module tb\_fb\_cpu;

parameter TEST\_CASE = 1;

parameter ADDRESS\_WIDTH = 6;

parameter DATA\_WIDTH = 10;

reg clk = 1;

reg rst;

wire [ADDRESS\_WIDTH-1:0] addr\_toRAM;

wire [DATA\_WIDTH-1:0] data\_toRAM, data\_fromRAM;

wire [ADDRESS\_WIDTH-1:0] pCounter;

wire wrEn;

always clk = #5 !clk;

reg error;

initial begin

rst = 1;

error = 0;

#100;

rst <= #1 0;

#5000;

if(TEST\_CASE == 1)

memCheck(52,15);

else if(TEST\_CASE == 2)

memCheck(52,50);

else if(TEST\_CASE == 3)

memCheck(52,50);

#100;

$finish;

end

fb\_cpu #(

ADDRESS\_WIDTH,

DATA\_WIDTH

) fb\_cpu\_Inst(

.clk(clk),

.rst(rst),

.MDRIn(data\_toRAM),

.RAMWr(wrEn),

.MAR(addr\_toRAM),

.MDROut(data\_fromRAM),

.PC(pCounter)

);

blram #(ADDRESS\_WIDTH, 64, TEST\_CASE) blram(

.clk(clk),

.rst(rst),

.i\_we(wrEn),

.i\_addr(addr\_toRAM),

.i\_ram\_data\_in(data\_toRAM),

.o\_ram\_data\_out(data\_fromRAM)

);

task memCheck;

input [5:0] memLocation;

input [9:0] expectedValue;

begin

if(blram.memory[memLocation] != expectedValue) begin

error = 1;

end

end

endtask

endmodule

module blram(clk, rst, i\_we, i\_addr, i\_ram\_data\_in, o\_ram\_data\_out);

parameter SIZE = 6;

parameter DEPTH = 64;

parameter TEST\_CASE = 1;

input clk;

input rst;

input i\_we;

input [SIZE-1:0] i\_addr;

input [9:0] i\_ram\_data\_in;

output reg [9:0] o\_ram\_data\_out;

reg [9:0] memory[0:DEPTH-1];

always @(posedge clk) begin

o\_ram\_data\_out <= #1 memory[i\_addr[SIZE-1:0]];

if (i\_we)

memory[i\_addr[SIZE-1:0]] <= #1 i\_ram\_data\_in;

end

initial begin

if(TEST\_CASE == 1) begin

memory[0] = 10'b0000110010;

memory[1] = 10'b0010110011;

memory[2] = 10'b0001110100;

memory[3] = 10'b1001000000;

memory[50] = 10'b0000000101;

memory[51] = 10'b0000001010;

end else if(TEST\_CASE == 2) begin

memory[0] = 10'b0000110010;

memory[1] = 10'b0100110011;

memory[2] = 10'b0001110100;

memory[3] = 10'b1001000000;

memory[50] = 10'b0000000101;

memory[51] = 10'b0000001010;

end else if(TEST\_CASE == 3) begin

memory[0]= 10'b0000110011;

memory[1]= 10'b0011110001;

memory[2]= 10'b0111001010;

memory[3]= 10'b0000110000;

memory[4]= 10'b0010110010;

memory[5]= 10'b0001110000;

memory[6]= 10'b0000110001;

memory[7]= 10'b0010101110;

memory[8]= 10'b0001110001;

memory[9]= 10'b0110000000;

memory[10]= 10'b0000110000;

memory[11]= 10'b0001110100;

memory[12]= 10'b1001000000;

memory[46]= 10'b1;

memory[48]= 10'b0;

memory[49]= 10'b0;

memory[50]= 10'b0000000101;

memory[51]= 10'b0000001010;

end

end

endmodule

module fb\_cpu #(

parameter ADDRESS\_WIDTH = 6,

parameter DATA\_WIDTH = 10

)(

input clk,

input rst,

output reg [DATA\_WIDTH-1:0] MDRIn,

output reg RAMWr,

output reg [ADDRESS\_WIDTH-1:0] MAR,

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

output reg [5:0] PC

);

reg [DATA\_WIDTH - 1:0] IR, IRNext;

reg [5:0] PCNext;

reg [9:0] ACC, ACCNext;

reg [2:0] state, stateNext;

always@(posedge clk) begin

state <= #1 stateNext;

PC <= #1 PCNext;

IR <= #1 IRNext;

ACC <= #1 ACCNext;

end

always@(\*) begin

stateNext = state;

PCNext = PC;

IRNext = IR;

ACCNext = ACC;

MAR = 0;

RAMWr = 0;

MDRIn = 0;

if(rst) begin

stateNext = 0;

PCNext = 0;

MAR = 0;

RAMWr = 0;

IRNext = 0;

ACCNext = 0;

MDRIn = 0;

end else begin

case(state)

0: begin

MAR = PC;

RAMWr = 0;

stateNext = state + 1;

end

1: begin

IRNext = MDROut;

PCNext = PC + 1;

stateNext = state + 1;

end

2: begin

case(IR[9:6])

4'b0000: begin

MAR = IR[5:0];

stateNext = 3;

end

4'b0001: begin

MAR = IR[5:0];

MDRIn = ACC;

RAMWr = 1;

stateNext = 0;

end

4'b0010: begin

MAR = IR[5:0];

stateNext = 3;

end

4'b0011: begin

MAR = IR[5:0];

stateNext = 3;

end

4'b0100: begin

MAR = IR[5:0];

stateNext = 3;

end

4'b0110: begin

PCNext = IR[5:0];

stateNext = 0;

end

4'b0111: begin

if (ACC == 0)

PCNext = IR[5:0];

stateNext = 0;

end

4'b1001: begin

stateNext = 4;

end

endcase

end

3: begin

case(IR[9:6])

4'b0000: begin

ACCNext = MDROut;

stateNext = 0;

end

4'b0010: begin

ACCNext = ACC + MDROut;

stateNext = 0;

end

4'b0011: begin

ACCNext = ACC - MDROut;

stateNext = 0;

end

4'b0100: begin

ACCNext = ACC \* MDROut;

stateNext = 0;

end

endcase

end

4: begin

stateNext = 4;

end

endcase

end

end

endmodule

module top (

input clk,

input rst,

input [15:0] switches,

input btnU,

input btnD,

input btnL,

input btnR,

input btnM,

output reg [15:0] leds,

output reg [7:0] ss3, ss2, ss1, ss0,

output reg red, green, blue

);

tb\_fb\_cpu tb\_fb\_cpuInst();

endmodule