module full\_adder\_1bit(S, Cout, A, B, Cin);

input A, B, Cin;

output Cout, S;

assign S = A ^ B ^ Cin;

assign Cout = (A & B) | (B & Cin) | (A & Cin);

endmodule

module addsub\_4bit(Sum, Ovfl, A, B, sub);

input[3:0] A, B;

input sub;

output [3:0] Sum;

output Ovfl;

wire [3:0] Cout;

wire [3:0] invB;

assign invB = sub? ~B + 1'b1: B;

assign Ovfl = assign Ovfl = sub ? (Sum[3] & ~A[3] & ~B[3]) | (~Sum[3] & A[3] & B[3]): (Sum[3] & ~A[3] & B[3]) | (~Sum[3] & A[3] & ~B[3]);

full\_adder\_1bit A1 (Sum[0], Cout[0], A[0], invB[0], 1'b0);

full\_adder\_1bit A2 (Sum[1], Cout[1], A[1], invB[1], Cout[0]);

full\_adder\_1bit A3 (Sum[2], Cout[2], A[2], invB[2], Cout[1]);

full\_adder\_1bit A4 (Sum[3], Cout[3], A[3], invB[3], Cout[2]);

endmodule

module addsub\_4bit\_tb();

reg [7:0] stim;

wire [3:0] Sum;

wire Ovfl;

reg sub;

reg [3:0] temp;

addsub\_4bit DUT(Sum, Ovfl, stim[3:0], stim[7:4], sub);

initial begin

assign sub = 1'b0;

stim = 8'b0;

repeat(100) begin

temp = stim[3:0] + stim[7:4];

#20 stim = $random;

assert(Sum == temp) $display("success");

else

begin

$display("sum is %d, it should be %d", Sum, stim[3:0] + stim[7:4]);

$stop;

end

end

stim = 8'b11111111;

assert(Ovfl == 1'b1) $display("overflow passed for add");

#10;

assign sub = 1'b1;

stim = 8'b0;

repeat(100) begin

temp = stim[3:0] - stim[7:4];

#20 stim = $random;

assert(Sum == temp) $display("success");

else

begin

$display("sum is %d, it should be %d", Sum, stim[3:0] + stim[7:4]);

$stop;

end

end

stim = 8'b11110111;

assert(Ovfl == 1'b1) $display("overflow passed for sub");

#10;

$display("all tests passed");

$stop;

end

endmodule

module ALU (ALU\_Out, Error, ALU\_In1, ALU\_In2, Opcode);

input [3:0] ALU\_In1, ALU\_In2;

input [1:0] Opcode;

output reg [3:0] ALU\_Out;

output Error; // Just to show overflow

wire [3:0] ALU\_result;

wire sub;

assign sub = (Opcode[0])? 1'b1:1'b0;

addsub\_4bit adder(ALU\_result, Error, ALU\_In1, ALU\_In2, sub);

always @(\*)

begin

casex(Opcode)

2'b00: // Add

assign ALU\_Out = ALU\_result;

2'b01:

assign ALU\_Out = ALU\_result;

2'b10: // Nand

assign ALU\_Out = ~(ALU\_In1 & ALU\_In2);

2'B11: // Xor

assign ALU\_Out = ALU\_In1 ^ ALU\_In2;

default:

assign ALU\_Out = 4'b0000;

endcase

end

endmodule

module ALU\_tb();

reg [1:0] ops;

wire [3:0] ALU\_Out;

reg [7:0] stim;

reg [3:0] temp;

wire Error;

ALU DUT(.ALU\_Out(ALU\_Out), .Error(Error), .ALU\_In1(stim[3:0]), .ALU\_In2(stim[7:4]), .Opcode(ops));

initial begin

assign ops = 2'b00;

stim = 8'b0;

repeat(100) begin

temp = stim[3:0] + stim[7:4];

#20 stim = $random;

assert(ALU\_Out == temp)

else

begin

$display("Result is %d, it should be %d", ALU\_Out, temp);

end

end

$display("Addition tests passed");

#10;

assign ops = 2'b01;

repeat(100) begin

temp = stim[3:0] - stim[7:4];

#20 stim = $random;

assert(ALU\_Out == temp)

else

begin

$display("Result is %d, it should be %d", ALU\_Out, temp);

end

end

$display("Subtraction tests passed");

#10;

assign ops = 2'b10;

repeat(100) begin

temp = ~(stim[3:0] & stim[7:4]);

#20 stim = $random;

assert(ALU\_Out == temp)

else

begin

$display("Result is %d, it should be %d", ALU\_Out, temp);

$stop;

end

end

$display("NAND tests passed");

#10;

assign ops = 2'b11;

repeat(100) begin

temp = stim[3:0] ^ stim[7:4];

#20 stim = $random;

assert(ALU\_Out == temp)

else

begin

$display("Result is %d, it should be %d", ALU\_Out, temp);

$stop;

end

end

$display("Xor tests passed");

#10;

$stop;

end

endmodule