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 adder\_4bit(Sum, Ovfl, A, B);

input[3:0] A, B;

output [3:0] Sum;

output Ovfl;

wire [3:0] Cout;

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

full\_adder\_1bit A1 (.S(Sum[0]),

.Cout(Cout[0]),

.A(A[0]),

.B(B[0]),

.Cin(1'b0)

);

full\_adder\_1bit A2 (.S(Sum[1]),

.Cout(Cout[1]),

.A(A[1]),

.B(B[1]),

.Cin(Cout[0])

);

full\_adder\_1bit A3 (.S(Sum[2]),

.Cout(Cout[2]),

.A(A[2]),

.B(B[2]),

.Cin(Cout[1]));

full\_adder\_1bit A4 (.S(Sum[3]),

.Cout(Cout[3]),

.A(A[3]),

.B(B[3]),

.Cin(Cout[2])

);

Endmodule

module PSA\_16bit (Sum, Error, A, B);

input [15:0] A, B; //Input values

output [15:0] Sum; //sum output

output Error; //To indicate overflows

wire [3:0] Ovfl;

adder\_4bit ll(.Sum(Sum[3:0]),

.Ovfl(Ovfl[0]),

.A(A[3:0]),

.B(B[3:0])

);

adder\_4bit l(.Sum(Sum[7:4]),

.Ovfl(Ovfl[1]),

.A(A[7:4]),

.B(B[7:4])

);

adder\_4bit h(.Sum(Sum[11:8]),

.Ovfl(Ovfl[2]),

.A(A[11:8]),

.B(B[11:8])

);

adder\_4bit hh(.Sum(Sum[15:12]),

.Ovfl(Ovfl[3]),

.A(A[15:12]),

.B(B[15:12])

);

assign Error = |Ovfl;

endmodule

module PSA\_tb();

reg [15:0] stimA;

reg [15:0] stimB;

wire [15:0] Sum;

reg [3:0] Ovfl;

wire Error;

reg [3:0] tempA,tempB,tempC,tempD;

PSA\_16bit DUT(.Sum(Sum[15:0]),

.Error(Error),

.A(stimA[15:0]),

.B(stimB[15:0])

);

initial begin

stimA = 16'b0;

stimB = 16'b0;

Ovfl = 4'b0;

repeat(100) begin

tempA = stimA[3:0] + stimB[3:0];

tempB = stimA[7:4] + stimB[7:4];

tempC = stimA[11:8] + stimB[11:8];

tempD = stimA[15:12] + stimB[15:12];

Ovfl[0] = (tempA[3] & ~stimA[3] & ~stimB[3]) | (~tempA[3] & stimA[3] & stimB[3]);

Ovfl[1] = (tempB[3] & ~stimA[7] & ~stimB[7]) | (~tempB[3] & stimA[7] & stimB[7]);

Ovfl[2] = (tempC[3] & ~stimA[11] & ~stimB[11]) | (~tempC[3] & stimA[11] & stimB[11]);

Ovfl[3] = (tempD[3] & ~stimA[15] & ~stimB[15]) | (~tempD[3] & stimA[15] & stimB[15]);

#20 begin

stimA = $random;

stimB = $random;

end

assert(Sum[3:0] == tempA & Sum[7:4] == tempB & Sum[11:8] == tempC & Sum[15:12] == tempD)

else

begin

$display("shoot! Sum is wrong, here is a list of info: Sum: %h, temp: %h %h %h %h",

Sum, tempA, tempB, tempC, tempD);

$stop;

end

assert(Error == |Ovfl)

else

begin

$display("shoot! Error is wrong, here is a list of info: Error: %h, Ovfl: %b",

Error, Ovfl);

$stop;

end

end

$display("All test passed!");

$finish;

end

endmodule

module Shifter (Shift\_Out, Shift\_In, Shift\_Val, Mode);

input [15:0] Shift\_In; //This is the number to perform shift operation on

input [3:0] Shift\_Val; //Shift amount (used to shift the ‘Shift\_In’)

input Mode; // To indicate SLL or SRA

output [15:0] Shift\_Out; //Shifter value

reg [15:0] sll, sra;

always@(Mode or Shift\_Val or Shift\_In) begin

case(Shift\_Val)

0: sll = Shift\_In;

1: sll = {Shift\_In[14:0], 1'b0};

2: sll = {Shift\_In[13:0], 2'b0};

3: sll = {Shift\_In[12:0], 3'b0};

4: sll = {Shift\_In[11:0], 4'b0};

5: sll = {Shift\_In[10:0], 5'b0};

6: sll = {Shift\_In[9:0], 6'b0};

7: sll = {Shift\_In[8:0], 7'b0};

8: sll = {Shift\_In[7:0], 8'b0};

9: sll = {Shift\_In[6:0], 9'b0};

10: sll = {Shift\_In[5:0], 10'b0};

11: sll = {Shift\_In[4:0], 11'b0};

12: sll = {Shift\_In[3:0], 12'b0};

13: sll = {Shift\_In[2:0], 13'b0};

14: sll = {Shift\_In[1:0], 14'b0};

15: sll = {Shift\_In[0], 15'b0};

default : sll = Shift\_In;

endcase

end

always@(Mode, Shift\_Val, Shift\_In) begin

case(Shift\_Val)

0: sra = Shift\_In;

1: sra = {Shift\_In[15], Shift\_In[15:1]};

2: sra = {{2{Shift\_In[15]}}, Shift\_In[15:2]};

3: sra = {{3{Shift\_In[15]}}, Shift\_In[15:3]};

4: sra = {{4{Shift\_In[15]}}, Shift\_In[15:4]};

5: sra = {{5{Shift\_In[15]}}, Shift\_In[15:5]};

6: sra = {{6{Shift\_In[15]}}, Shift\_In[15:6]};

7: sra = {{7{Shift\_In[15]}}, Shift\_In[15:7]};

8: sra = {{8{Shift\_In[15]}}, Shift\_In[15:8]};

9: sra = {{9{Shift\_In[15]}}, Shift\_In[15:9]};

10: sra = {{10{Shift\_In[15]}}, Shift\_In[15:10]};

11: sra = {{11{Shift\_In[15]}}, Shift\_In[15:11]};

12: sra = {{12{Shift\_In[15]}}, Shift\_In[15:12]};

13: sra = {{13{Shift\_In[15]}}, Shift\_In[15:13]};

14: sra = {{14{Shift\_In[15]}}, Shift\_In[15:14]};

15: sra = {{15{Shift\_In[15]}}, Shift\_In[15]};

default : sra = Shift\_In;

endcase

end

assign Shift\_Out = (Mode)? sra:sll;

endmodule

module shifter\_tb();

reg [15:0] Shift\_In; //This is the number to perform shift operation on

reg [3:0] Shift\_Val; //Shift amount (used to shift the ‘Shift\_In’)

reg Mode; // To indicate SLL or SRA

wire [15:0] Shift\_Out; //Shifter value

Shifter DUT(.Shift\_Out(Shift\_Out),

.Shift\_In(Shift\_In),

.Shift\_Val(Shift\_Val),

.Mode(Mode)

);

initial begin

Shift\_In = 16'b0000000000000001;

Shift\_Val = 4'b0001;

Mode = 0;

#100

assert(Shift\_Out == 16'b0000000000000010) $display("sll success: %h", Shift\_Out);

else

$display("sll failed: %h", Shift\_Out);

Mode = 0;

Shift\_In = 16'b0000000000000001;

Shift\_Val = 4'b0010;

#100

assert(Shift\_Out == 16'b0000000000000100) $display("sll success: %h", Shift\_Out);

else

$display("sll failed: %h", Shift\_Out);

Mode = 0;

Shift\_In = 16'b0000000000000001;

Shift\_Val = 4'b0011;

#100

assert(Shift\_Out == 16'b000000000001000) $display("sll success: %h", Shift\_Out);

else

$display("sll failed: %h", Shift\_Out);

Shift\_In = 16'b1000000000000000;

Shift\_Val = 4'b0001;

Mode = 1;

#100

assert(Shift\_Out == 16'b1100000000000000) $display("sll success: %h", Shift\_Out);

else

$display("sll failed: %h", Shift\_Out);

Mode = 1;

Shift\_In = 16'b1000000000000000;

Shift\_Val = 4'b0010;

#100

assert(Shift\_Out == 16'b1110000000000000) $display("sll success: %h", Shift\_Out);

else

$display("sll failed: %h", Shift\_Out);

Mode = 1;

Shift\_In = 16'b1000000000000000;

Shift\_Val = 4'b0011;

#100

assert(Shift\_Out == 16'b1111000000000000) $display("sll success: %h", Shift\_Out);

else

$display("sll failed: %h", Shift\_Out);

$stop;

end

endmodule