`timescale 1ns/1ps

//------------------ 1) 4-Bit Han-Carlson Adder ------------------

module hancarlson\_4bit(

input [3:0] A,

input [3:0] B,

input cin,

output [3:0] sum,

output cout

);

wire [3:0] g = A & B;

wire [3:0] p = A ^ B;

wire c0 = cin;

wire c1 = g[0] | (p[0] & c0);

wire c2 = g[1] | (p[1] & g[0]) | (p[1] & p[0] & c0);

wire c3 = g[2] | (p[2] & g[1]) | (p[2] & p[1] & g[0]) | (p[2] & p[1] & p[0] & c0);

wire c4 = g[3] | (p[3] & g[2]) | (p[3] & p[2] & g[1]) |

(p[3] & p[2] & p[1] & g[0]) | (p[3] & p[2] & p[1] & p[0] & c0);

assign sum[0] = p[0] ^ c0;

assign sum[1] = p[1] ^ c1;

assign sum[2] = p[2] ^ c2;

assign sum[3] = p[3] ^ c3;

assign cout = c4;

endmodule

//------------------ 2) 4-Bit Ling Adder ------------------

module ling\_4bit(

input [3:0] A,

input [3:0] B,

input cin,

output [3:0] sum,

output cout

);

wire [3:0] g = A & B;

wire [3:0] p = A | B;

wire c0 = cin;

wire c1 = g[0] | (p[0] & c0);

wire c2 = g[1] | (p[1] & g[0]) | (p[1] & p[0] & c0);

wire c3 = g[2] | (p[2] & g[1]) | (p[2] & p[1] & g[0]) | (p[2] & p[1] & p[0] & c0);

wire c4 = g[3] | (p[3] & g[2]) | (p[3] & p[2] & g[1]) |

(p[3] & p[2] & p[1] & g[0]) | (p[3] & p[2] & p[1] & p[0] & c0);

assign sum[0] = A[0] ^ B[0] ^ c0;

assign sum[1] = A[1] ^ B[1] ^ c1;

assign sum[2] = A[2] ^ B[2] ^ c2;

assign sum[3] = A[3] ^ B[3] ^ c3;

assign cout = c4;

endmodule

//------------------ 3) Binary to Excess-1 Converter ------------------

module bec\_4bit(

input [3:0] in,

output [3:0] out

);

assign out = in + 1;

endmodule

//------------------ 4) Weinberger Adder ------------------

module weinberger\_4bit(

input [3:0] A,

input [3:0] B,

input cin,

output [3:0] sum,

output cout

);

wire [3:0] sum0;

wire c1, c2, c3, c4;

wire c0 = 1'b0;

assign sum0[0] = A[0] ^ B[0] ^ c0;

assign c1 = (A[0] & B[0]) | ((A[0] ^ B[0]) & c0);

assign sum0[1] = A[1] ^ B[1] ^ c1;

assign c2 = (A[1] & B[1]) | ((A[1] ^ B[1]) & c1);

assign sum0[2] = A[2] ^ B[2] ^ c2;

assign c3 = (A[2] & B[2]) | ((A[2] ^ B[2]) & c2);

assign sum0[3] = A[3] ^ B[3] ^ c3;

assign c4 = (A[3] & B[3]) | ((A[3] ^ B[3]) & c3);

wire [3:0] sum1;

bec\_4bit U\_bec(.in(sum0), .out(sum1));

wire [3:0] dummy;

wire cc1, cc2, cc3, cc4;

assign dummy[0] = A[0] ^ B[0] ^ 1'b1;

assign cc1 = (A[0] & B[0]) | ((A[0] ^ B[0]) & 1'b1);

assign dummy[1] = A[1] ^ B[1] ^ cc1;

assign cc2 = (A[1] & B[1]) | ((A[1] ^ B[1]) & cc1);

assign dummy[2] = A[2] ^ B[2] ^ cc2;

assign cc3 = (A[2] & B[2]) | ((A[2] ^ B[2]) & cc2);

assign dummy[3] = A[3] ^ B[3] ^ cc3;

assign cc4 = (A[3] & B[3]) | ((A[3] ^ B[3]) & cc3);

assign sum = (cin == 1'b0) ? sum0 : sum1;

assign cout = (cin == 1'b0) ? c4 : cc4;

endmodule

//------------------ 5) Han-Carlson Adder + BEC ------------------

module hancarlson\_bec\_4bit(

input [3:0] A,

input [3:0] B,

input cin,

output [3:0] sum,

output cout

);

wire [3:0] sum0;

wire c0out;

hancarlson\_4bit hc0(

.A(A), .B(B), .cin(1'b0), .sum(sum0), .cout(c0out)

);

wire [3:0] sum1;

bec\_4bit U\_bec(.in(sum0), .out(sum1));

wire [3:0] dummy;

wire d1, d2, d3, d4;

assign dummy[0] = A[0] ^ B[0] ^ 1'b1;

assign d1 = (A[0] & B[0]) | ((A[0] ^ B[0]) & 1'b1);

assign dummy[1] = A[1] ^ B[1] ^ d1;

assign d2 = (A[1] & B[1]) | ((A[1] ^ B[1]) & d1);

assign dummy[2] = A[2] ^ B[2] ^ d2;

assign d3 = (A[2] & B[2]) | ((A[2] ^ B[2]) & d2);

assign dummy[3] = A[3] ^ B[3] ^ d3;

assign d4 = (A[3] & B[3]) | ((A[3] ^ B[3]) & d3);

assign sum = (cin == 1'b0) ? sum0 : sum1;

assign cout = (cin == 1'b0) ? c0out : d4;

endmodule

//------------------ 6) 16-Bit Hybrid Adder ------------------

module hybrid\_16bit\_adder(

input [15:0] A,

input [15:0] B,

input cin,

output [15:0] sum,

output cout

);

wire [3:0] sum0, sum1, sum2, sum3;

wire c0, c1, c2;

hancarlson\_4bit u0(.A(A[3:0]), .B(B[3:0]), .cin(cin), .sum(sum0), .cout(c0));

weinberger\_4bit u1(.A(A[7:4]), .B(B[7:4]), .cin(c0), .sum(sum1), .cout(c1));

weinberger\_4bit u2(.A(A[11:8]), .B(B[11:8]), .cin(c1), .sum(sum2), .cout(c2));

hancarlson\_bec\_4bit u3(.A(A[15:12]), .B(B[15:12]), .cin(c2), .sum(sum3), .cout(cout));

assign sum = {sum3, sum2, sum1, sum0};

endmodule

//==================== BOOTH MULTIPLIER (Signed) =====================

module PARTIALPRODUCT (

input signed [7:0] INPUT1,

input [2:0] SEGMENT,

output reg signed [15:0] OUTPUT1

);

always @(\*) begin

case (SEGMENT)

3'b000, 3'b111: OUTPUT1 = 16'sd0;

3'b001, 3'b010: OUTPUT1 = INPUT1;

3'b011: OUTPUT1 = INPUT1 <<< 1;

3'b100: OUTPUT1 = -(INPUT1 <<< 1);

3'b101, 3'b110: OUTPUT1 = -INPUT1;

endcase

end

endmodule

module booth\_multiplier(

input signed [7:0] A,

input signed [7:0] B,

output signed [15:0] PRODUCT

);

wire signed [15:0] P[0:3];

wire [2:0] SEGS[0:3];

assign SEGS[0] = {B[1:0], 1'b0};

assign SEGS[1] = {B[3:2], B[1]};

assign SEGS[2] = {B[5:4], B[3]};

assign SEGS[3] = {B[7:6], B[5]};

genvar i;

generate

for (i = 0; i < 4; i = i + 1) begin: partial\_gen

PARTIALPRODUCT pp\_inst (

.INPUT1(A),

.SEGMENT(SEGS[i]),

.OUTPUT1(P[i])

);

end

endgenerate

wire signed [15:0] PP0 = P[0];

wire signed [15:0] PP1 = P[1] <<< 2;

wire signed [15:0] PP2 = P[2] <<< 4;

wire signed [15:0] PP3 = P[3] <<< 6;

wire signed [15:0] SUM1, SUM2;

hybrid\_16bit\_adder ADD1 (.A(PP0), .B(PP1), .cin(1'b0), .sum(SUM1), .cout());

hybrid\_16bit\_adder ADD2 (.A(SUM1), .B(PP2), .cin(1'b0), .sum(SUM2), .cout());

hybrid\_16bit\_adder ADD3 (.A(SUM2), .B(PP3), .cin(1'b0), .sum(PRODUCT), .cout());

endmodule

**ADP = 2023.56**