Assignment –3 Part_B

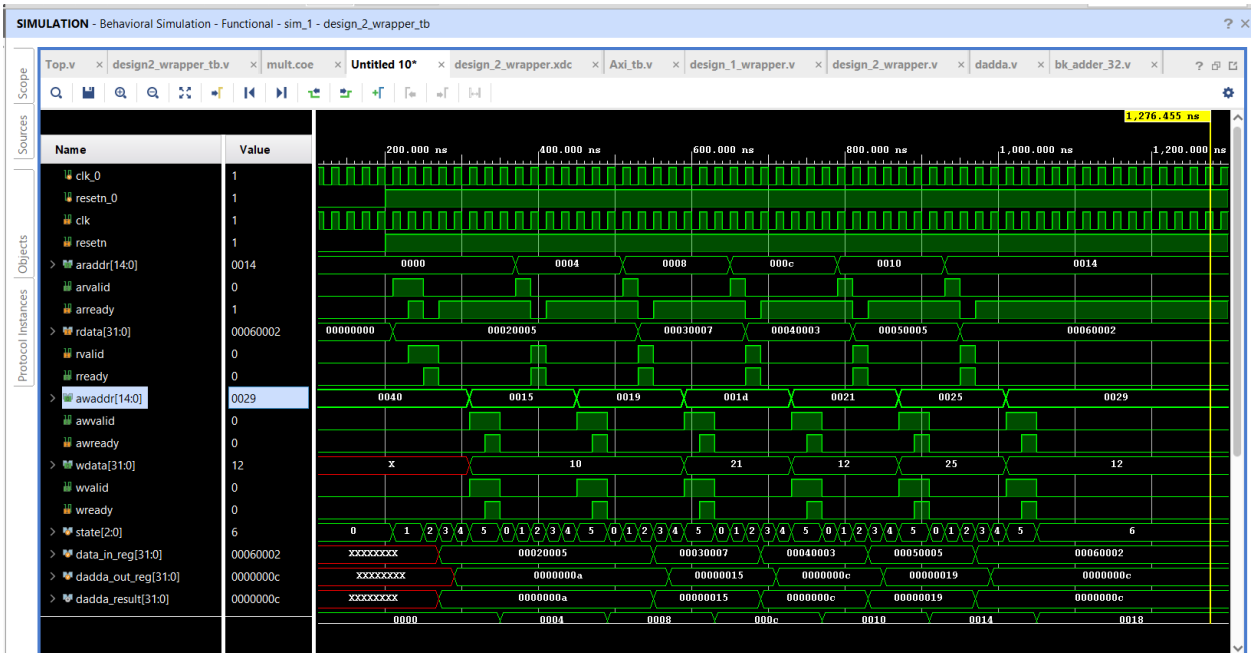
VLSI Design Lab

Manish Ranjan

24M1176

3 B.

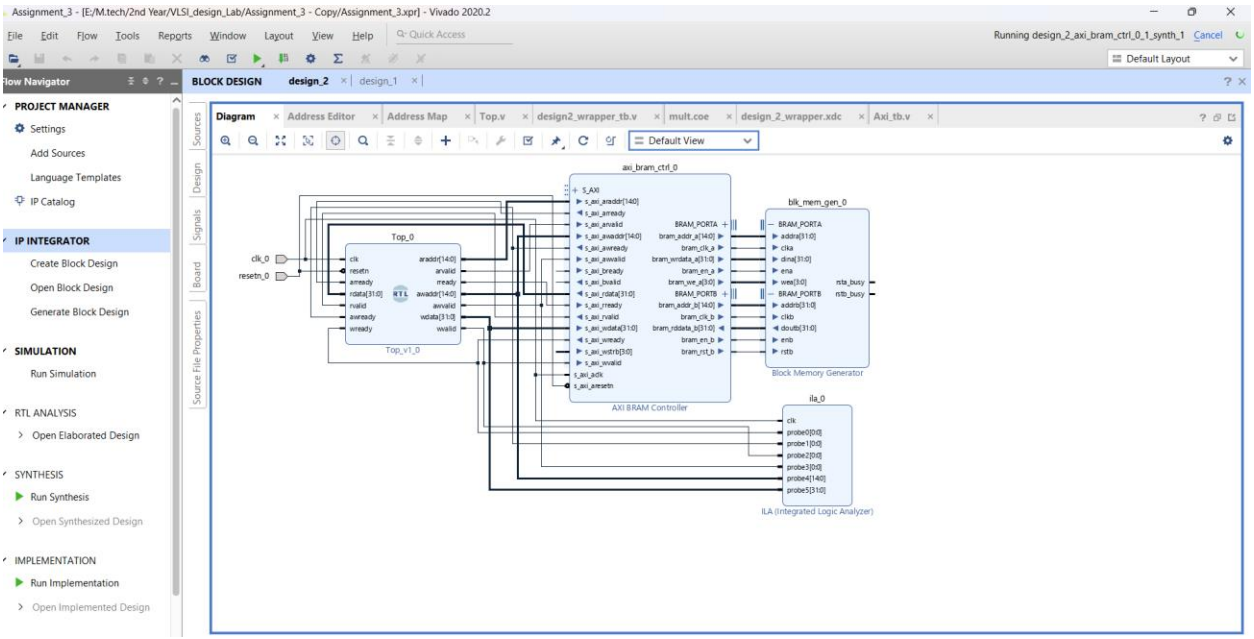
Behavioural Simulation:



Coe File content:

memory_initialization_radix=16; memory_initialization_vector=00020005 00030007 00040003 00050005 00060002 00000000 00000000 00000000 00000000 00000000 00000000;

Block diagram



Verilog codes:

`timescale 1ns / 1ps

module Top (input clk, // AXI clock input resetn, // AXI reset (active-low) // AXI Read Address Channel output reg [14:0] araddr,

output reg arvalid,

input arready,

// AXI Read Data Channel input [31:0] rdata,

input rvalid,

output reg rready,

// AXI Write Address Channel output reg [14:0] awaddr,

output reg awvalid,

input awready,

// AXI Write Data Channel output reg [31:0] wdata,

output reg wvalid,

```
 input wready

);

parameter IDLE = 3'b000, READ_ADDR = 3'b001, READ_DATA = 3'b010,
          Mult = 3'b011, WRITE_ADDR = 3'b100, WRITE_DATA = 3'b101,
DONE = 3'b110;

reg [2:0] state;
reg [31:0] data_in_reg;
reg [31:0] dadda_out_reg;
wire [31:0] dadda_result;

reg [14:0] raddr;
reg [14:0] waddr;
reg [2:0] rw_count;  // Counter to limit reads/writes to 5

// Instantiate Dadda Multiplier
d_mult d1 (
    .r_data(data_in_reg),
    .res(dadda_result)
);

always @(posedge clk) begin
    if (~resetn) begin
        state <= IDLE;

arvalid <= 0;
        araddr <= 0;  // Start at address 0
        rready <= 0;
        awvalid <= 0;
        awaddr <= 64;  // Start at address 64
        wvalid <= 0;
        rw_count <= 0; // Reset counter
        raddr <= 0;
        waddr <= 21;
    end else begin
        case (state)
            IDLE: begin
                if (rw_count < 6) begin  // Perform only 5 reads
```

```
        arvalid <= 1;
        araddr <= raddr;
        state <= READ_ADDR;
    end else begin
        state <= DONE;  // Stop operation after 5 cycles
    end
end

READ_ADDR: begin
    if (arready) begin
        arvalid <= 0;
        rready <= 1;
        state <= READ_DATA;
    end
end

READ_DATA: begin
    if (rvalid) begin
        data_in_reg <= rdata[31:0];
        rready <= 0;
        state <= Mult;
    end
end

Mult: begin
    dadda_out_reg <= dadda_result;
    state <= WRITE_ADDR;
end

WRITE_ADDR: begin
    awvalid <= 1;
    awaddr <= waddr;
    wdata <= dadda_out_reg;
    wvalid <= 1;
    state <= WRITE_DATA;
end


 WRITE_DATA: begin
```

```
                if (wready && awready) begin
                    awvalid <= 0;
                    wvalid <= 0;

                    // Increment read/write addresses
                    raddr <= raddr + 4;
                    waddr <= waddr + 4;
                    rw_count <= rw_count + 1;  // Increment counter
after each write

                    // Ensure exactly 5 reads and 5 writes (fix
condition)
                    if (rw_count < 5) begin
                            state <= IDLE;
                    end else begin
                        state <= DONE;
                    end
                end
            end



            DONE: begin

            end

            default: state <= IDLE;
        endcase
    end
end


Endmodule
```

Dadda Multiplier : .

`timescale 1ns / 1ps

module d_mult(//input [15:0] A, //input [15:0] B, input [31:0] r_data, output [31:0] res); wire [15:0] A,B; assign {A,B}=r_data; reg [30:0] pp[15:0]; integer i, j; reg [1:0] ha_tmp; reg [1:0] fa_tmp, fa_tmp1, fa_tmp2, fa_tmp3; reg c_tmp, c_tmp1, c_tmp2, c_tmp3; reg [31:0] s_temp, s_temp1;

task half_adder; input p, q; output [1:0] result; begin result[0] = p ^ q; result[1] = p & q; end endtask

task full_adder; input m, n, cin; output [1:0] result; begin result[0] = m ^ n ^ cin; result[1] = (m & n) | (n & cin) | (m & cin); end endtask

always @(*) begin // Initialize partial products to zero for (i = 0; i < 16; i = i + 1) begin pp[i] = 31'd0; end

```
// Partial products calculation
for (i = 0; i < 16; i = i + 1)
begin
    for (j = 0; j < 16; j = j + 1)
    begin
        pp[i][j + i] = B[i] & A[j];
    end
end

// Creating triangle structure for better visualization and reducing
complexity
for (j = 16; j < 31; j = j + 1)
begin
    for (i = (j - 15); i < 16; i = i + 1)
    begin
        pp[i - (j - 15)][j] = pp[i][j];
        pp[i][j] = 0;
    end
end

// Reduction to d = 13
half_adder(pp[12][13], pp[13][13], ha_tmp);
pp[12][13] = ha_tmp[0];
c_tmp = ha_tmp[1];

full_adder(pp[12][14], pp[13][14], pp[14][14], fa_tmp);
```

```
half_adder(fa_tmp[0], c_tmp, ha_tmp);
pp[12][14] = ha_tmp[0];
c_tmp = fa_tmp[1];
c_tmp1 = ha_tmp[1];

full_adder(pp[13][15], pp[14][15], pp[15][15], fa_tmp1);
full_adder(pp[12][15], c_tmp1, c_tmp, fa_tmp2);
half_adder(fa_tmp1[0], fa_tmp2[0], ha_tmp);
pp[12][15] = ha_tmp[0];
c_tmp = fa_tmp1[1];
c_tmp1 = fa_tmp2[1];
c_tmp2 = ha_tmp[1];

full_adder(pp[12][16], pp[13][16], pp[14][16], fa_tmp1);
full_adder(c_tmp2, c_tmp1, c_tmp, fa_tmp2);
half_adder(fa_tmp1[0], fa_tmp2[0], ha_tmp);
pp[12][16] = ha_tmp[0];
c_tmp = fa_tmp1[1];
c_tmp1 = fa_tmp2[1];
c_tmp2 = ha_tmp[1];

full_adder(c_tmp2, pp[12][17], pp[13][17], fa_tmp1);
full_adder(fa_tmp1[0], c_tmp1, c_tmp, fa_tmp2);
pp[12][17] = fa_tmp2[0];
c_tmp = fa_tmp1[1];
c_tmp1 = fa_tmp2[1];

full_adder(pp[12][18], c_tmp, c_tmp1, fa_tmp1);
pp[12][18] = fa_tmp1[0];
pp[12][19] = fa_tmp1[1];

// Reduction to d = 9
half_adder(pp[8][9], pp[9][9], ha_tmp);
pp[8][9] = ha_tmp[0];
c_tmp = ha_tmp[1];

full_adder(pp[8][10], pp[9][10], pp[10][10], fa_tmp);
half_adder(fa_tmp[0], c_tmp, ha_tmp);
pp[8][10] = ha_tmp[0];
```

```
c_tmp = fa_tmp[1];
c_tmp1 = ha_tmp[1];

full_adder(pp[9][11], pp[10][11], pp[11][11], fa_tmp1);
full_adder(pp[8][11], c_tmp1, c_tmp, fa_tmp2);
half_adder(fa_tmp1[0], fa_tmp2[0], ha_tmp);
pp[8][11] = ha_tmp[0];
c_tmp = fa_tmp1[1];
c_tmp1 = fa_tmp2[1];
c_tmp2 = ha_tmp[1];

full_adder(pp[10][12], pp[11][12], pp[12][12], fa_tmp);
full_adder(pp[8][12], pp[9][12], c_tmp, fa_tmp1);
full_adder(fa_tmp[0], c_tmp1, c_tmp2, fa_tmp2);
half_adder(fa_tmp1[0], fa_tmp2[0], ha_tmp);
pp[8][12] = ha_tmp[0];
c_tmp = fa_tmp[1];
c_tmp1 = fa_tmp1[1];
c_tmp2 = fa_tmp2[1];
c_tmp3 = ha_tmp[1];

for (j = 13; j < 20; j = j + 1)
begin
    full_adder(pp[10][j], pp[11][j], pp[12][j], fa_tmp);
    full_adder(pp[8][j], pp[9][j], c_tmp, fa_tmp1);
    full_adder(fa_tmp[0], fa_tmp1[0], c_tmp1, fa_tmp2);
    full_adder(fa_tmp2[0], c_tmp2, c_tmp3, fa_tmp3);
    pp[8][j] = fa_tmp3[0];
    c_tmp = fa_tmp[1];
    c_tmp1 = fa_tmp1[1];
    c_tmp2 = fa_tmp2[1];
    c_tmp3 = fa_tmp3[1];
end

full_adder(pp[8][20], pp[9][20], pp[10][20], fa_tmp);
full_adder(fa_tmp[0], c_tmp, c_tmp1, fa_tmp1);
full_adder(fa_tmp1[0], c_tmp2, c_tmp3, fa_tmp2);
pp[8][20] = fa_tmp2[0];
c_tmp = fa_tmp[1];
```

```
c_tmp1 = fa_tmp1[1];
c_tmp2 = fa_tmp2[1];

full_adder(c_tmp, pp[8][21], pp[9][21], fa_tmp);
full_adder(fa_tmp[0], c_tmp1, c_tmp2, fa_tmp1);
pp[8][21] = fa_tmp1[0];
c_tmp = fa_tmp[1];
c_tmp1 = fa_tmp1[1];

full_adder(c_tmp, c_tmp1, pp[8][22], fa_tmp);
pp[8][22] = fa_tmp[0];
pp[8][23] = fa_tmp[1];

// Reduction to d = 6
half_adder(pp[5][6], pp[6][6], ha_tmp);
pp[5][6] = ha_tmp[0];
c_tmp = ha_tmp[1];

full_adder(pp[5][7], pp[6][7], pp[7][7], fa_tmp);
half_adder(fa_tmp[0], c_tmp, ha_tmp);
pp[5][7] = ha_tmp[0];
c_tmp = fa_tmp[1];
c_tmp1 = ha_tmp[1];

full_adder(pp[6][8], pp[7][8], pp[8][8], fa_tmp1);
full_adder(pp[5][8], c_tmp1, c_tmp, fa_tmp2);
half_adder(fa_tmp1[0], fa_tmp2[0], ha_tmp);
pp[5][8] = ha_tmp[0];
c_tmp = fa_tmp1[1];
c_tmp1 = fa_tmp2[1];
c_tmp2 = ha_tmp[1];

for (j = 9; j < 24; j = j + 1)
begin
    full_adder(pp[8][j], pp[7][j], pp[6][j], fa_tmp);
    full_adder(pp[5][j], c_tmp, c_tmp1, fa_tmp1);
    full_adder(fa_tmp[0], fa_tmp1[0], c_tmp2, fa_tmp2);
    pp[5][j] = fa_tmp2[0];
    c_tmp = fa_tmp[1];
```

```
    c_tmp1 = fa_tmp1[1];
    c_tmp2 = fa_tmp2[1];
end

full_adder(pp[6][24], pp[5][24], c_tmp, fa_tmp);
full_adder(fa_tmp[0], c_tmp1, c_tmp2, fa_tmp1);
pp[5][24] = fa_tmp1[0];
c_tmp = fa_tmp[1];
c_tmp1 = fa_tmp1[1];

full_adder(pp[5][25], c_tmp, c_tmp1, fa_tmp);
pp[5][25] = fa_tmp[0];
pp[5][26] = fa_tmp[1];

// Reduction to d = 4
half_adder(pp[3][4], pp[4][4], ha_tmp);
pp[3][4] = ha_tmp[0];
c_tmp = ha_tmp[1];

full_adder(pp[3][5], pp[4][5], pp[5][5], fa_tmp);
half_adder(fa_tmp[0], c_tmp, ha_tmp);
pp[3][5] = ha_tmp[0];
c_tmp = fa_tmp[1];
c_tmp1 = ha_tmp[1];

for (j = 6; j < 27; j = j + 1)
begin
    full_adder(pp[3][j], pp[4][j], pp[5][j], fa_tmp);
    full_adder(c_tmp, c_tmp1, fa_tmp[0], fa_tmp1);
    pp[3][j] = fa_tmp1[0];
    c_tmp = fa_tmp[1];
    c_tmp1 = fa_tmp1[1];
end

full_adder(pp[3][27], c_tmp, c_tmp1, fa_tmp);
pp[3][27] = fa_tmp[0];
pp[3][28] = fa_tmp[1];

// Reduction to d = 3
```

```
half_adder(pp[2][3], pp[3][3], ha_tmp);
pp[2][3] = ha_tmp[0];
c_tmp = ha_tmp[1];

for (j = 4; j < 29; j = j + 1)
begin
    full_adder(pp[3][j], pp[2][j], c_tmp, fa_tmp);
    pp[2][j] = fa_tmp[0];
    c_tmp = fa_tmp[1];
end
pp[2][29] = c_tmp;

// Reduction to d = 2
half_adder(pp[1][2], pp[2][2], ha_tmp);
pp[1][2] = ha_tmp[0];
c_tmp = ha_tmp[1];

for (j = 3; j < 30; j = j + 1)
begin
    full_adder(pp[1][j], pp[2][j], c_tmp, fa_tmp);
    pp[1][j] = fa_tmp[0];
    c_tmp = fa_tmp[1];
end
pp[1][30] = c_tmp;

for (j = 0; j < 31; j = j + 1)
begin
    s_temp[j] = pp[0][j];
end

for (j = 0; j < 31; j = j + 1)
begin
    s_temp1[j] = pp[1][j];
end

s_temp[31] = 0;
s_temp1[31] = 0;
```

end

wire cout; bkadder_32 add (.A(s_temp), .B(s_temp1), .cin(1'b0), .Y(res), .cout(cout));

endmodule

Brent Kung adder code:

`timescale 1ns / 1ps // // Company: // Engineer: // // Create Date: 01/24/2025 12:37:16 AM // Design Name: // Module Name: bk_adder_32 // Project Name: // Target Devices: // Tool Versions: // Description: // // Dependencies: // // Revision: // Revision 0.01 - File Created // Additional Comments: // //

module bkadder_32(

```
input [31:0] A ,         // Input A
input [31:0] B,             // Input B
input cin,                   // Carry-in
output [31:0] Y,    // 32-bit sum
output cout         // Final carry-out
        );

//reg [31:0] A, B;
//reg cin;


wire [31:0] p, g;           // Initial propagate and generate signals
wire [31:0] c;              // Intermediate carry signals

// Declaring intermediate wires for the carry tree
wire p10, g10, p32, g32, p54, g54, p76, g76, p98, g98, p1110, g1110,
p1312, g1312, p1514, g1514;
wire p1716, g1716, p1918, g1918, p2120, g2120, p2322, g2322, p2524,
g2524, p2726, g2726, p2928, g2928, p3130, g3130;

wire p30, g30, p74, g74, p118, g118, p1512, g1512, p1916, g1916,
p2320, g2320, p2724, g2724, p3128, g3128;

wire p70, g70, p158, g158, p2316, g2316, p3124, g3124;
```

```
wire p150,g150,p3116,g3116;

wire p310,g310;
wire p230,g230;

wire p110,g110,p190,g190,p270,g270;

wire
p50,g50,p90,g90,p130,g130,p170,g170,p210,g210,p250,g250,p290,g290;

wire
p20,p40,p60,p80,p100,g100,p120,p140,p160,p180,p200,p220,p240,p260,p280
,p300;
wire
g20,g40,g60,g80,g120,g140,g160,g180,g200,g220,g240,g260,g280,g300;


// Generate initial propagate and generate signals
assign p = A ^ B;
assign g = A & B;

// Carry propagation tree
//level 1
pg_gen i1 (p[1], p[0], g[1], g[0], p10, g10);
pg_gen i2 (p[3], p[2], g[3], g[2], p32, g32);
pg_gen i3 (p[5], p[4], g[5], g[4], p54, g54);
pg_gen i4 (p[7], p[6], g[7], g[6], p76, g76);
pg_gen i5 (p[9], p[8], g[9], g[8], p98, g98);
pg_gen i6 (p[11], p[10], g[11], g[10], p1110, g1110);
pg_gen i7 (p[13], p[12], g[13], g[12], p1312, g1312);
pg_gen i8 (p[15], p[14], g[15], g[14], p1514, g1514);
pg_gen i9 (p[17], p[16], g[17], g[16], p1716, g1716);
pg_gen i10 (p[19], p[18], g[19], g[18], p1918, g1918);
pg_gen i11 (p[21], p[20], g[21], g[20], p2120, g2120);
pg_gen i12 (p[23], p[22], g[23], g[22], p2322, g2322);
pg_gen i13 (p[25], p[24], g[25], g[24], p2524, g2524);
pg_gen i14 (p[27], p[26], g[27], g[26], p2726, g2726);
pg_gen i15 (p[29], p[28], g[29], g[28], p2928, g2928);
```

```
pg_gen i16 (p[31], p[30], g[31], g[30], p3130, g3130);
```

//level 2 pg_gen i17 (p32, p10, g32, g10, p30, g30); pg_gen i18 (p76, p54, g76, g54, p74, g74); pg_gen i19 (p1110, p98, g1110, g98, p118, g118); pg_gen i20 (p1514, p1312, g1514, g1312, p1512, g1512); pg_gen i21 (p1918, p1716, g1918, g1716, p1916, g1916); pg_gen i22 (p2322, p2120, g2322, g2120, p2320, g2320); pg_gen i23 (p2726, p2524, g2726, g2524, p2724, g2724); pg_gen i24 (p3130, p2928, g3130, g2928, p3128, g3128);

//level 3 pg_gen i25 (p74, p30, g74, g30, p70, g70); pg_gen i26 (p1512, p118, g1512, g118, p158, g158); pg_gen i27 (p2320, p1916, g2320, g1916, p2316, g2316); pg_gen i28 (p3128, p2724, g3128, g2724, p3124, g3124);

```
//level 4

pg_gen i29(p158,p70,g158,g70,p150,g150);
pg_gen i30(p3124,p2316,g3124,g2316,p3116,g3116);

//level 5

pg_gen i31(p3116,p150,g3116,g150,p310,g310);

//level 6
pg_gen i32(p2316,p150,g2316,g150,p230,g230);
//level 7
pg_gen i33(p118,p70,g118,g70,p110,g110);
pg_gen i34(p1916,p150,g1916,g150,p190,g190);
pg_gen i35(p2724,p230,g2724,g230,p270,g270);

//level 8

pg_gen i36(p54,p30,g54,g30,p50,g50);
pg_gen i37(p98,p70,g98,g70,p90,g90);
pg_gen i38(p1312,p110,g1312,g110,p130,g130);
pg_gen i39(p1716,p150,g1716,g150,p170,g170);
pg_gen i40(p2120,p190,g2120,g190,p210,g210);
pg_gen i41(p2524,p230,g2524,g230,p250,g250);
pg_gen i42(p2928,p270,g2928,g270,p290,g290);
```

```
//level 9

pg_gen i43(p[2],p10,g[2],g10,p20,g20);
pg_gen i44(p[4],p30,g[4],g30,p40,g40);
pg_gen i45(p[6],p50,g[6],g50,p60,g60);
pg_gen i46(p[8],p70,g[8],g70,p80,g80);
pg_gen i47(p[10],p90,g[10],g90,p100,g100);
pg_gen i48(p[12],p110,g[12],g110,p120,g120);
pg_gen i49(p[14],p130,g[14],g130,p140,g140);
pg_gen i50(p[16],p150,g[16],g150,p160,g160);
pg_gen i490(p[18],p170,g[12],g170,p180,g180);
pg_gen i501(p[20],p190,g[20],g190,p200,g200);
pg_gen i51(p[22],p210,g[22],g210,p220,g220);
pg_gen i52(p[24],p230,g[24],g230,p240,g240);
pg_gen i53(p[26],p250,g[25],g250,p260,g260);
pg_gen i54(p[28],p270,g[28],g270,p280,g280);
pg_gen i55(p[30],p290,g[30],g290,p300,g300);
```

```
// Carry signal computation
assign c[0] = g[0] | (cin & p[0]);
genvar i;
generate
    for (i = 1; i < 32; i = i + 1) begin
        assign c[i] = g[i] | (p[i] & c[i-1]);
    end
endgenerate

// Final carry-out
assign cout = c[31];

// Sum computation
assign Y = p ^ {c[30:0], cin};
```

endmodule

module pg_gen(input Px, input Py, input Gx, input Gy, output Pxy, output Gxy); assign Gxy = Gx | (Px & Gy); assign Pxy = Px & Py; endmodule