CASEST

हैदराबाद विश्वविद्यालय
University of Hyderabad

Lab Assignment

Digital Signal Processing

Submitted by:

Full Name	**Enrolment No.**
Abhinav M	23PMMT11(@uohyd.ac.in)

Contents

Centre for Advanced Studies in Electronics Science and Technology

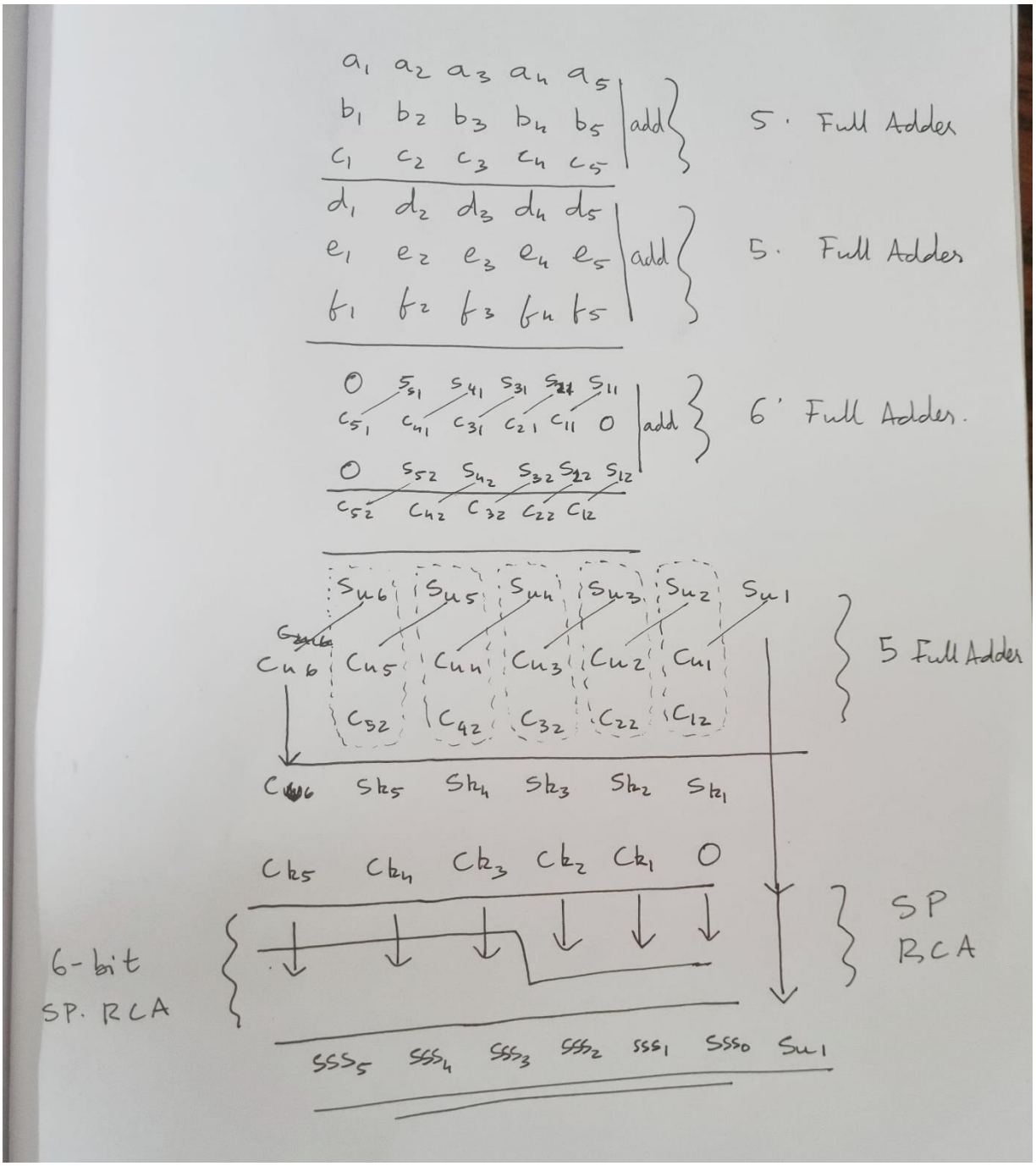
Question 1

To design a Wallace adder to add 6, five-bit numbers, report timing, utilization and power.

Logic

I used the following logic by using only full adders and no half-adders

Extra 0 bits were added when required to do the 3-bit sum.



Code

```
module fa1(
input a,
input b,
input cin,
output s,
output c);
assign s = a ^ b ^ cin;
assign c = (a & b) | (b & cin) | (cin & a);
endmodule


module df1 (Q,D,clk,reset);
output reg Q;
input D;
input clk,reset;
always @(posedge clk) begin
if(reset) Q<=0;
else Q<=D;
end
endmodule


module sprca_wallce_tree (
input [4:0] a,
input [4:0] b,
input [4:0] c,
input [4:0] d,
input [4:0] e,
input [4:0] f,
input clk,
input reset,
output [6:0] S
    );
wire s51,s41,s31,s21,s11;
wire c51,c41,c31,c21,c11;
wire s52,s42,s32,s22,s12;
wire c32,c42,c52,c22,c12;
wire su6,su5,su4,su3,su2,su1;
wire cu6,cu5,cu4,cu3,cu2;
wire cu1;
wire sk5,sk4,sk3,sk2,sk1;
wire ck5,ck4,ck3,ck2,ck1;
wire todff1,todff2,todff3;
wire ccc3,ccc4,ccc5,ccc6,ccc1,ccc2,delayccc3;
wire sss0,sss1,sss2,sss3,sss4,sss5;
wire x6,x5,x4,x3,x2,x1;
```

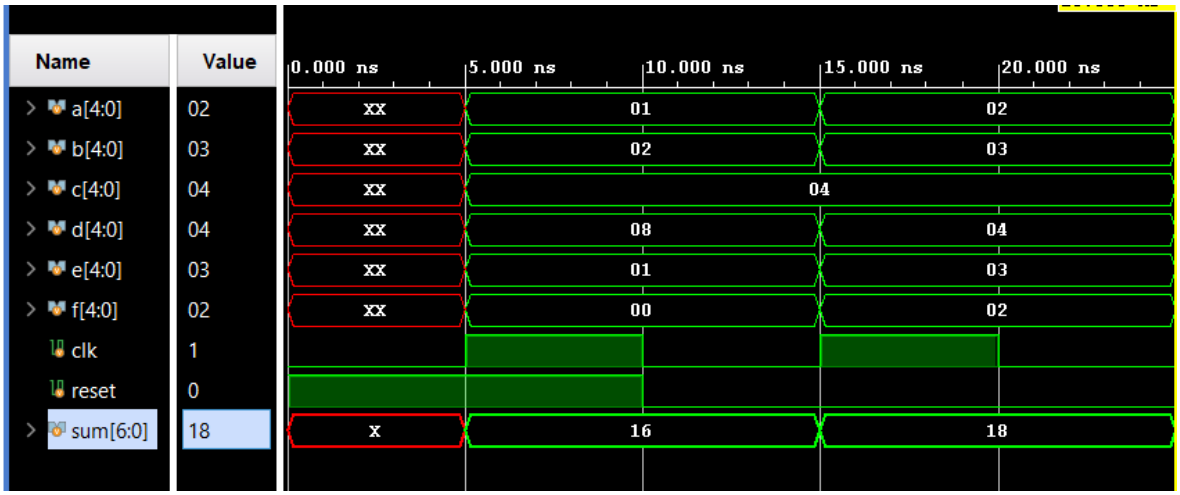
```
//STAGE 1 uses 10 FULL ADDERS

fa1
fa_array1[9:0]({a[4],a[3],a[2],a[1],a[0],d[4],d[3],d[2],d[1],d[0]},{b[4]
,b[3],b[2],b[1],b[0],e[4],e[3],e[2],e[1],e[0]},{c[4],c[3],c[2],c[1],c[0]
,f[4],f[3],f[2],f[1],f[0]},{s11,s21,s31,s41,s51,s12,s22,s32,s42,s52},{c1
1,c21,c31,c41,c51,c12,c22,c32,c42,c52});

//STAGE 2 uses 6 FULL ADDERS
fa1
fa_array2[5:0]({s11,s21,s31,s41,s51,1'b0},{1'b0,c11,c21,c31,c41,c51},{s1
2,s22,s32,s42,s52,1'b0},{su1,su2,su3,su4,su5,su6},{cu1,cu2,cu3,cu4,cu5,c
u6});

//STAGE 3 uses 5 FULL ADDERS
fa1
fa_array3[4:0]({su2,su3,su4,su5,su6},{cu1,cu2,cu3,cu4,cu5},{c12,c22,c32,
c42,c52},{sk1,sk2,sk3,sk4,sk5},{ck1,ck2,ck3,ck4,ck5});

//STAGE 4 - SEAMLESS RCA PART1
fa1
fa_array4[2:0]({1'b0,ck1,ck2},{sk1,sk2,sk3},{1'b0,ccc1,ccc2},{todff1,tod
ff2,todff3},{ccc1,ccc2,ccc3});

//STAGE 4 - SEAMLESS RCA PART2
df1 DFF_array1[2:0]({todff1,todff2,todff3},{sss0,sss1,sss2},clk,reset);
df1 ccc3dff(ccc3,delayccc3,clk,reset);

//STAGE 4 - SEAMLESS RCA PART3
df1
DFF_array2[5:0]({sk4,ck3,sk5,ck4,cu6,ck5},{x6,x5,x4,x3,x2,x1},clk,reset)
;

//STAGE 4 - SEAMLESS RCA PART4
fa1
fa_array5[2:0]({x6,x4,x2},{x5,x3,x1},{delayccc3,ccc4,ccc5},{sss3,sss4,ss
s5},{ccc4,ccc5,ccc6});

assign S = {sss5,sss4,sss3,sss2,sss1,sss0,su1};
endmodule
```

Test bench

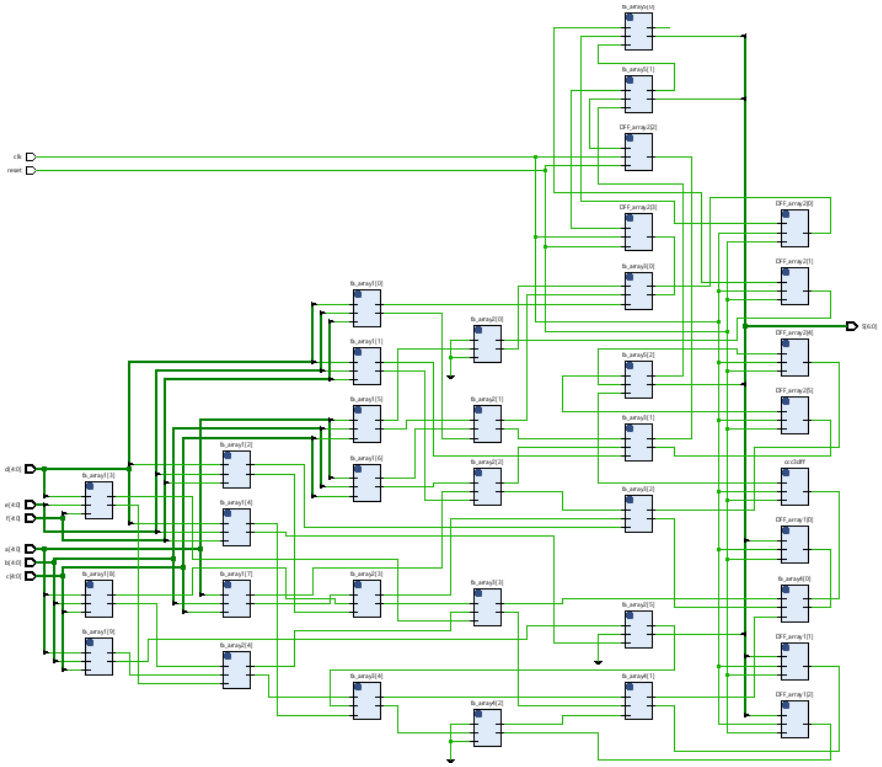
```
module sprca_wallace_tree_tb();

reg [4:0] a,b,c,d,e,f;
reg clk;
reg reset;
wire [6:0] sum;
k_wallce_tree uut(a,b,c,d,e,f,clk,reset,sum);

initial begin
  clk = 1'b0;
  forever #5 clk = ~clk;
end

initial begin
reset = 1;#5
```

```
    reset = 0;
    a=5'd1;b=5'd2;c=5'd4;d=5'd8;e=5'd1;f=5'd0;reset = 1;#5
    reset = 0;
    #5  a=5'd2;b=5'd3;c=5'd4;d=5'd4;e=5'd3;f=5'd2;#5 reset =1;
    reset = 0;#5
    #50 $finish;
end


endmodule
```

Waveform



Elaborated Design



Timing Summary (100ns clock)

Design Timing Summary

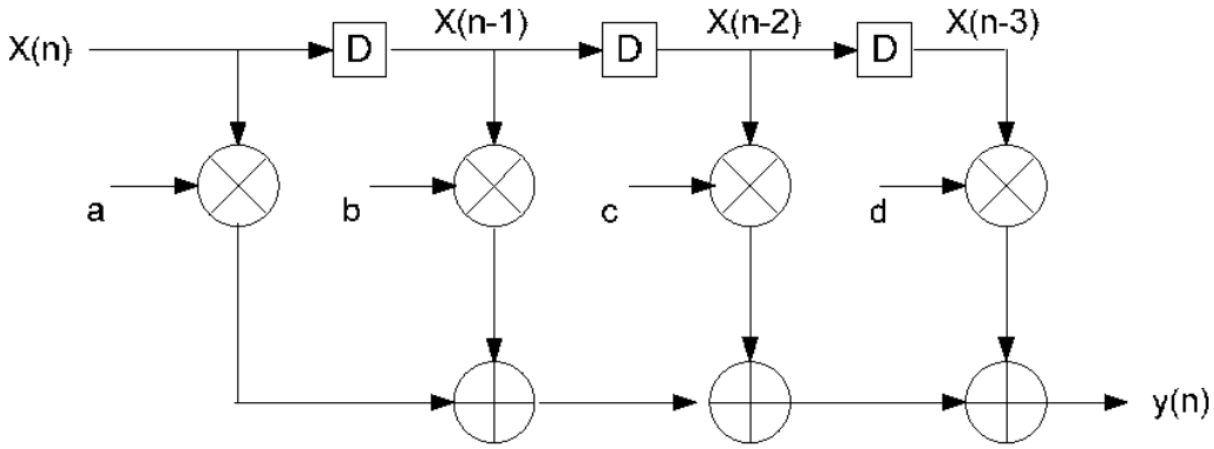
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	83.746 ns	Worst Hold Slack (WHS):	3.946 ns	Worst Pulse Width Slack (WPWS):	NA
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	NA
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	NA
Total Number of Endpoints:	1	Total Number of Endpoints:	1	Total Number of Endpoints:	NA

All user specified timing constraints are met.

Question 2

To design a 4- tap FIR filter, report timing, power and utilization.

Logic



Code

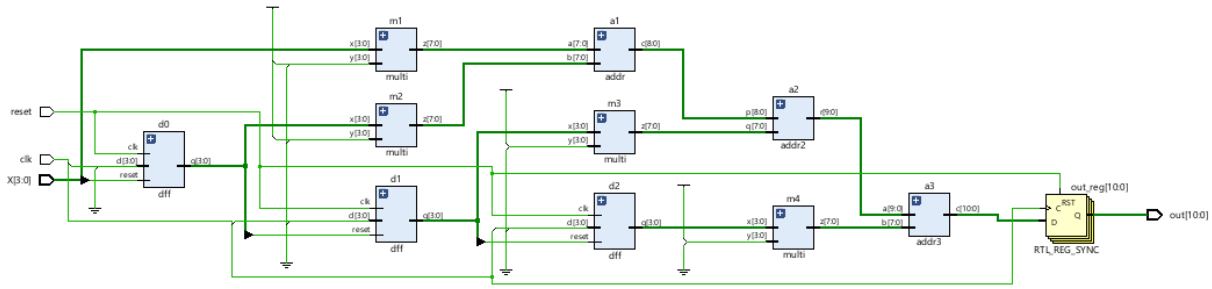
```
module tap_4_fir(
  input [3:0] X,
  input clk,
  input reset,
  output reg [10:0] out
);

  wire [3:0] H0,H1,H2,H3;
  wire [8:0] add_out1;
  wire [9:0] add_out2;
  wire [10:0] add_out3;
  wire [7:0] mul_out1,mul_out2,mul_out3,mul_out4;
  wire [3:0] X_delay1,X_delay2,X_delay3;

  // Giving h coefficient
assign H0 = 4'd1;
  assign H1 = 4'd2;
  assign H2 = 4'd3;
  assign H3 = 4'd4;

  multi m1(X,H0,mul_out1);
  multi m2(X_delay1,H1,mul_out2);
  multi m3(X_delay2,H2,mul_out3);
  multi m4(X_delay3,H3,mul_out4);
  dff d0(clk,reset,X,X_delay1);
  dff d1(clk,reset,X_delay1,X_delay2);
  dff d2(clk,reset,X_delay2,X_delay3);
  addr a1(mul_out1,mul_out2,add_out1);
  addr2 a2(add_out1,mul_out3,add_out2);
 addr3 a3(add_out2,mul_out4,add_out3);

  always @(posedge clk) begin
    if(reset) begin
```

```
      out<= 0;
    end
    else begin
      out <= add_out3;
    end

  end

endmodule


module multi (
  input [3:0] x,
  input [3:0] y,
  output [7:0] z);
  assign z = x*y;
endmodule

module addr (
  input [7:0] a,
  input [7:0] b,
  output [8:0] c);

  assign c = a + b;
endmodule

   module addr2(
   input [8:0] p,
   input [7:0] q,
   output [9:0] r);
   assign r = p + q;
   endmodule

   module addr3(
   input [9:0] a,
   input [7:0] b,
   output [10:0] c );
   assign c = a + b;
   endmodule


module dff (
input clk,
input reset,
  input [3:0] d,
  output reg [3:0] q
);


  always @(posedge clk) begin
    if(reset) begin
      q <= 0;
    end
    else begin
      q <= d;
    end
    end
endmodule
```

Testbench

```
module tb1;
  reg [3:0] X;
  reg clk;
  reg reset;
  wire [10:0] out;

  tap_4_fir uut(X,clk,reset,out);

  initial begin
    clk = 1'b0;
    forever #5 clk = ~clk;
  end


  initial begin
    #5 reset = 1;
    #8 reset = 0;
    X = 4'd1;
    //#5 X = 4'd7;

    #50 $finish;

  end

endmodule
```

Elaboration



Timing (100ns clock)

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	91.522 ns	Worst Hold Slack (WHS):	0.159 ns	Worst Pulse Width Slack (WPWS):	49.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	48	Total Number of Endpoints:	48	Total Number of Endpoints:	21

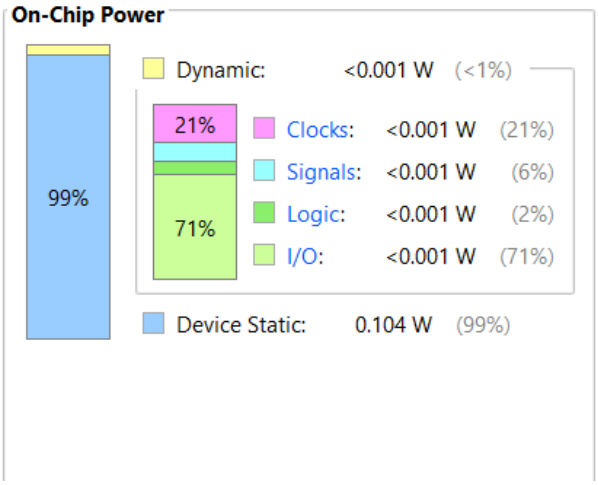
All user specified timing constraints are met.

Power

Summary

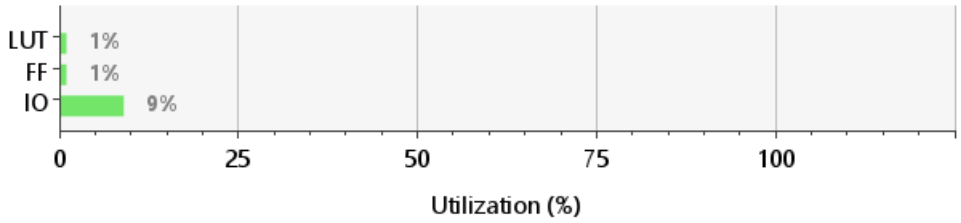
Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**0.104 W**
Design Power Budget:	**Not Specified**
Process:	typical
Power Budget Margin:	**N/A**
Junction Temperature:	**26.2°C**
Thermal Margin:	58.8°C (4.9 W)
Ambient Temperature:	25.0 °C
Effective ϑJA:	11.5°C/W



Utilization

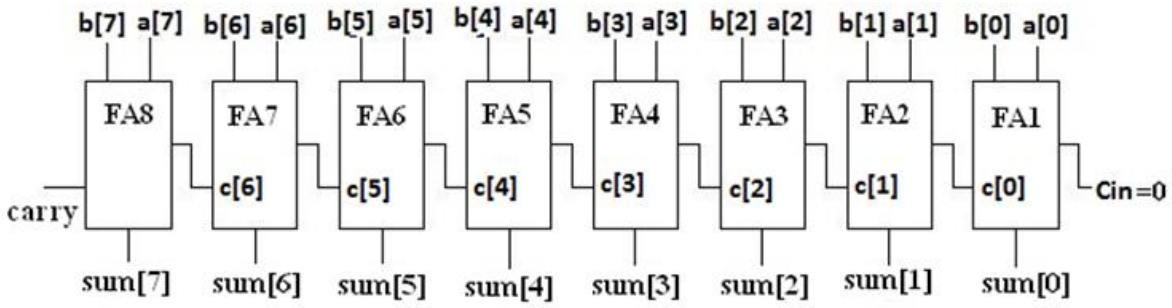
Resource	Utilization	Available	Utilization %
LUT	7	53200	0.01
FF	9	106400	0.01
IO	17	200	8.50



Question 3

To design a 8-bit Ripple Carry Adder, report timing, power and utilization.

Logic



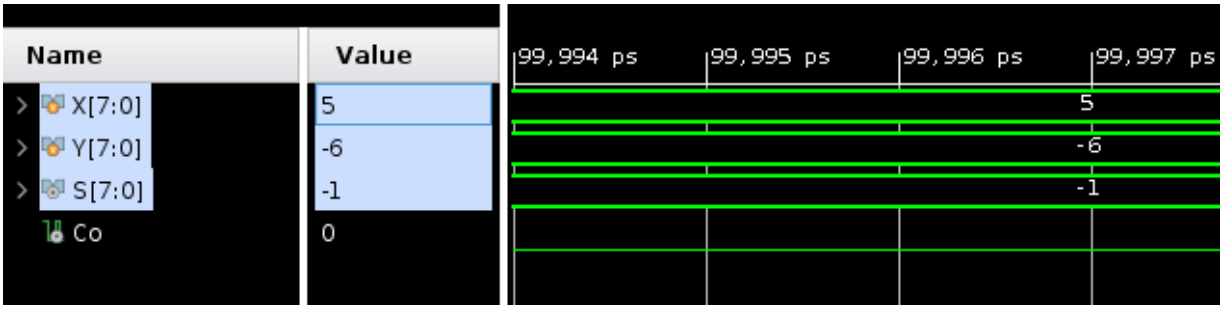
Code

```
module fad1(input x, y, ci, output s, co);
  wire w1,w2,w3;
  xor G1(w1, x, y);
  xor G2(s, w1, ci);
  and G3(w2, w1, ci);
  and G4(w3, x, y);
  or G5(co, w2, w3);
endmodule

module rca(X, Y, S, Co);
 input [7:0] X, Y;
 output [7:0] S;
 output Co;
 wire w1,w2,w3,w4,w5,w6,w7;

 fad1 u1(X[0], Y[0], 1'b0, S[0], w1);
 fad1 u2(X[1], Y[1], w1, S[1], w2);
 fad1 u3(X[2], Y[2], w2, S[2], w3);
 fad1 u4(X[3], Y[3], w3, S[3], w4);
 fad1 u5(X[4], Y[3], w4, S[4], w5);
 fad1 u6(X[5], Y[3], w5, S[5], w6);
 fad1 u7(X[6], Y[3], w6, S[6], w7);
 fad1 u8(X[7], Y[3], w7, S[7], Co);
endmodule
```

Testbench

```
module tb_rca();
reg[7:0]X,Y;
wire [7:0]S;
wire Co;
rca uut(X, Y, S, Co);
    initial begin
        X=8'd5;
        Y=-8'd6;
        #100 $finish;
    end
endmodule
```

Simulation



Timing (100ns clock)

Design Timing Summary

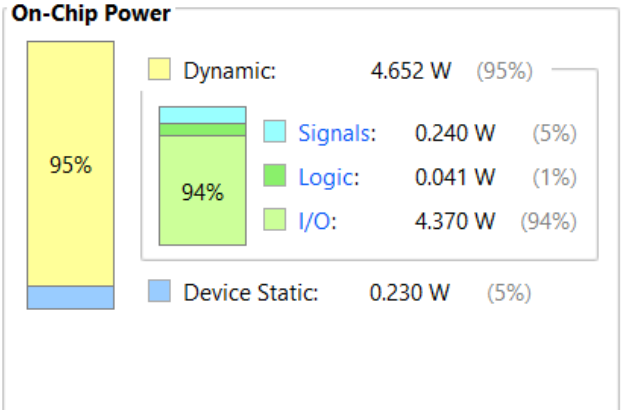
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	89.561 ns	Worst Hold Slack (WHS):	3.944 ns	Worst Pulse Width Slack (WPWS):	NA
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	NA
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	NA
Total Number of Endpoints:	9	Total Number of Endpoints:	9	Total Number of Endpoints:	NA

All user specified timing constraints are met.

Power

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

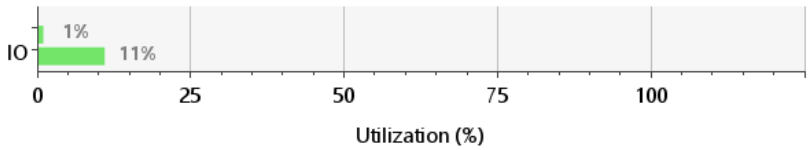
Total On-Chip Power:	**4.881 W**
Design Power Budget:	**Not Specified**
Process:	typical
Power Budget Margin:	**N/A**
Junction Temperature:	**70.6°C**
Thermal Margin:	14.4°C (1.2 W)



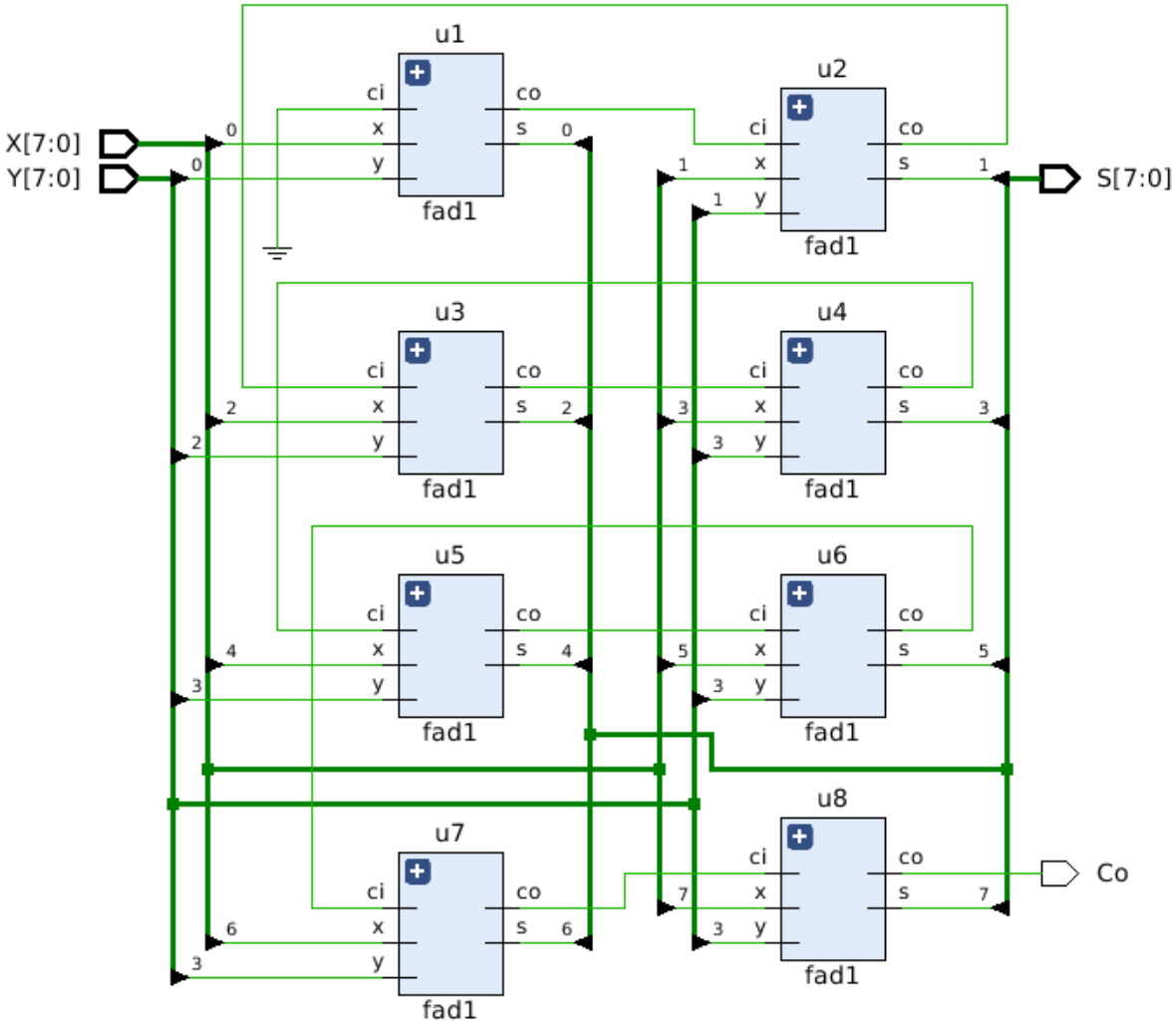
Utilization

Summary

Resource	Utilization	Available	Utilization %
LUT	8	53200	0.02
IO	21	200	10.50



Elaborated Design



Question 4

To do seamless pipelining of the RCA, report timing, power and utilization.

Logic

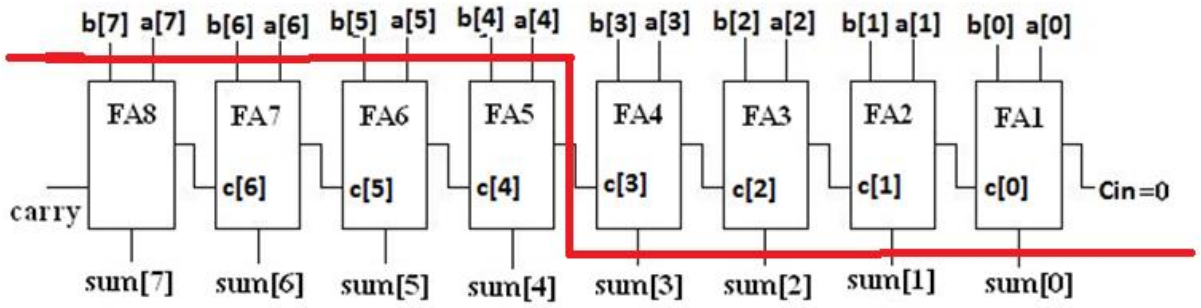


Figure 1: Pipelining strategy by adding delays along the cut set

Code

```
module Seamless_pipeline_RCA #(parameter N = 8)(
input clk,
input reset,
input [N-1:0] A,
input [N-1:0] B,
input Cin,
output [N-1:0] Sum,
output Cout
    );

 wire [7:0] Cout1;
 wire [3:0] sum_w;
 wire [7:0] a_w,b_w;

 fa1  f1 [3:0]
(A[3:0],B[3:0],{Cout1[2],Cout1[1],Cout1[0],Cin},{Sum[3],Sum[2],Sum[1],Sum[0
]},{Cout1[3],Cout1[2],Cout1[1],Cout1[0]});
 dff d1
({Sum[3],Sum[2],Sum[1],Sum[0]},clk,reset,{sum_w[3],sum_w[2],sum_w[1],sum_w[
0]});

 /*dff d2 (A[7:4],clk,reset,a_w[7:4]);
 dff  b1 (B[7:4],clk,reset,b_w[7:4]);*/
 dff d2_a (A[4],clk,reset,a_w[4]);
 dff d2_b(B[4],clk,reset,b_w[4]);
 dff d3_a(A[5],clk,reset,a_w[5]);
 dff d3_b(B[5],clk,reset,b_w[5]);
 dff d4_a(A[6],clk,reset,a_w[6]);
 dff d4_b(B[6],clk,reset,b_w[6]);
 dff d5_a(A[7],clk,reset,a_w[7]);
 dff d5_b(B[7],clk,reset,b_w[7]);

 fa1 fa4(a_w[4],b_w[4],Cout1[3],Sum[4],Cout1[4]);
 fa1 fa5(a_w[5],b_w[5],Cout1[4],Sum[5],Cout1[5]);

 fa1 fa6(a_w[6],b_w[6],Cout1[5],Sum[6],Cout1[6]);

 fa1 fa7(a_w[7],b_w[7],Cout1[6],Sum[7],Cout1[7]);
```

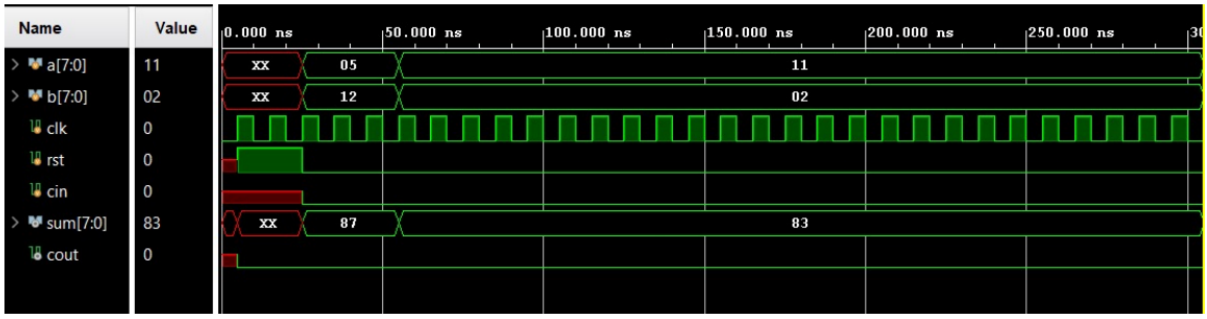
```
 assign a_w[3:0] = {4'b0};
 assign b_w[3:0] = {4'b0};

 assign Sum[3:0] = sum_w[3:0];
 assign Cout = Cout1[7];

endmodule

module dff(
input [3:0] d,
input clk,
input reset,
output reg [3:0]  q);
wire qbar;

always @(posedge clk) begin
if(reset) begin
  q <= 0;
end
else begin
  q <= d;
end
end
endmodule


module fa1(
input a,
input b,
input cin,
output sum,
output carry);

assign sum = a ^ b ^ cin;
assign carry = (a & b) | (b & cin) | (cin & a);
endmodule
```

Testbench

```
module RCA_seamless_tb();
reg [7:0] a;
reg [7:0] b;
reg clk,rst;
reg cin;
wire [7:0] sum;
wire cout;


RCA_seamless uut(a,b,cin,clk,rst,sum,cout);
initial
begin
clk = 1'b0;
forever #5 clk = ~clk;
end
initial
begin
#5 rst = 1;
#10 rst =0; a = 8'd5; b = 8'd8; cin = 0;
//#10 rst = 0; a=8'd17; b = 8'd2; cin = 0;
# 250 $finish;
```

```
end
endmodule
```

Simulation



Timing (250ns clock)

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	230.582 ns	Worst Hold Slack (WHS):	0.879 ns	Worst Pulse Width Slack (WPWS):	124.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	21	Total Number of Endpoints:	21	Total Number of Endpoints:	9

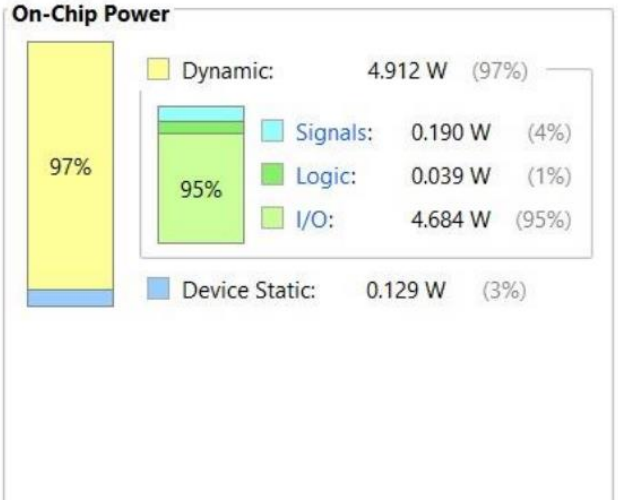
All user specified timing constraints are met.

Power

Summary

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

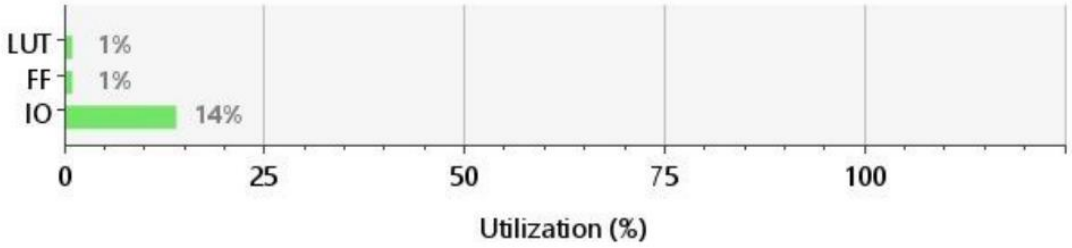
Total On-Chip Power:	**5.041 W**
Design Power Budget:	**Not Specified**
Process:	typical
Power Budget Margin:	**N/A**
Junction Temperature:	**40.9°C**
Thermal Margin:	59.1°C (13.9 W)
Ambient Temperature:	20.0 °C
Effective ϑJA:	4.2°C/W



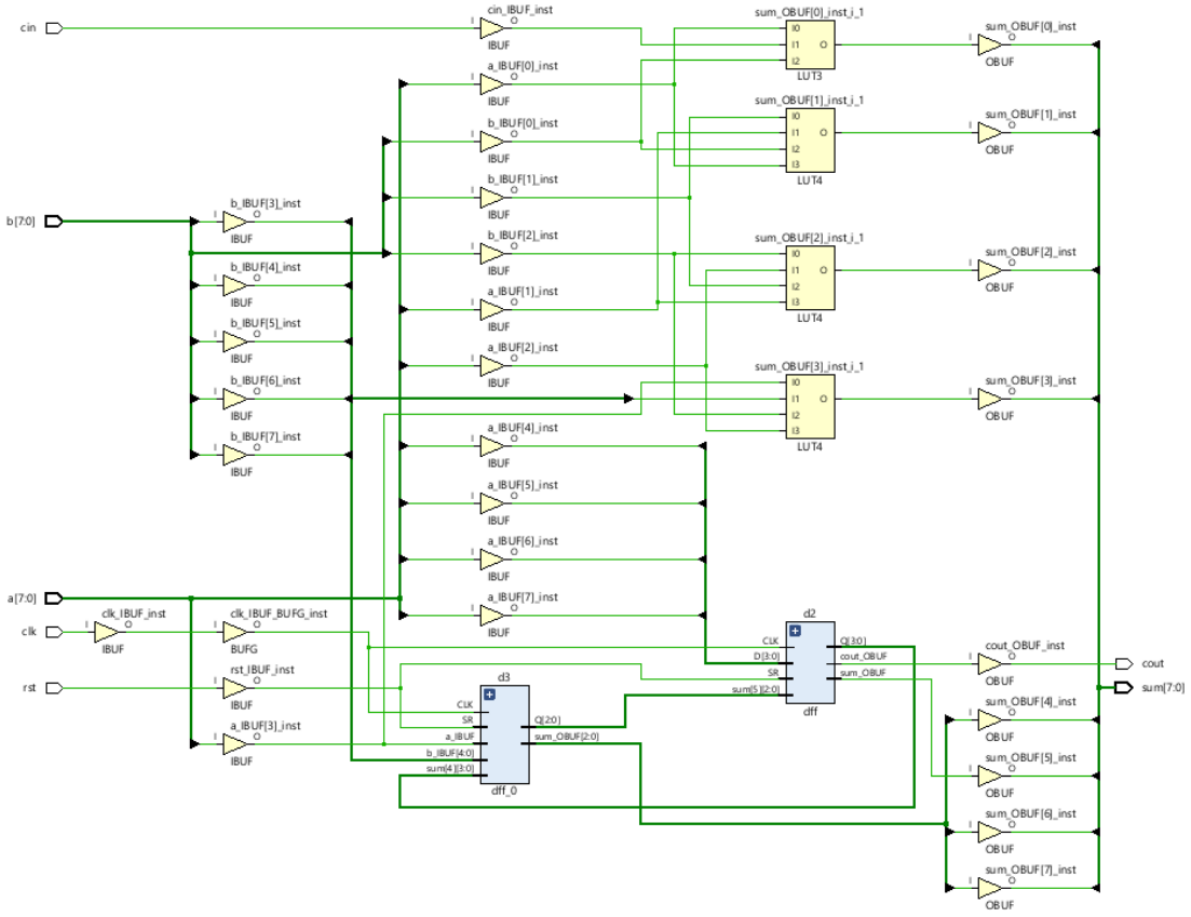
Utilization

Summary

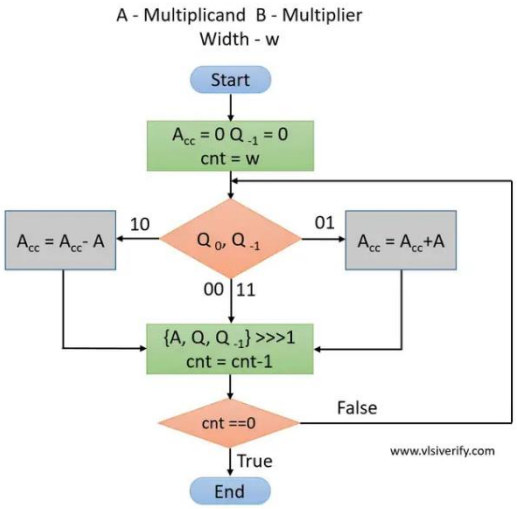
Resource	Utilization	Available	Utilization %
LUT	8	53200	0.02
FF	8	106400	0.01
IO	28	200	14.00



Synthesised Design



Booth Multiplier



Booth's Multiplier

4-bit booth multiplier Code

```
module BoothMulti(X, Y, Z);
      input signed [3:0] X, Y;
       output signed [7:0] Z;
       reg signed [7:0] Z;
       reg [1:0] temp;
       integer i;
       reg E1;
       reg [3:0] Y1;
       always @ (X, Y)
       begin
       Z = 8'd0;
       E1 = 1'd0;
       for (i = 0; i < 4; i = i + 1)
       begin
       temp = {X[i], E1};
       Y1 = - Y;
  case (temp)
       2'd2 : Z [7 : 4] = Z [7 : 4] + Y1;
       2'd1 : Z [7 : 4] = Z [7 : 4] + Y;
       default : begin end
       endcase
       Z = Z >> 1;

       Z[7] = Z[6];


       E1 = X[i];
           end
       if (Y == 4'd8)


           begin
               Z = - Z;
           end
       end
```

```
endmodule
```

Booth Multiplier with Wallace tree adder in structural

I tried to modify the Modified booth multiplier + Wallace adder architecture given in this IEEE paper to work with 2 4-bit inputs (https://doi.org/10.1051/e3sconf/202339101025)

I also used structural code from this repository: (https://github.com/pareddy113/Design-of-various-multiplier-Array-Booth-Wallace-/)

Modified Code

```
module hybrid_4bit(x,y,p);
input [3:0] x,y;
output [7:0] p;

wire [3:0] i,j,k,l,n,o,q,r;
wire [5:0] fp,sp,tp,fop,m;
wire [3:0] one,two,sign;
wire [2:0] cry,z;
// carry generation
// 1st
xor n9(z[0],one[0],two[0]);
and n10(cry[0],z[0],sign[0]);
//2nd
xor n13(z[1],one[1],two[1]);
and n14(cry[1],z[1],sign[1]);
//3rd
xor n15(z[2],one[2],two[2]);
and n16(cry[2],z[2],sign[2]);

code_4bit e1(one[0],two[0],sign[0],y[1],y[0],1'b0);
code_4bit e2(one[1],two[1],sign[1],y[3],y[2],y[1]);

//first product generation
product_4bit p0(x[0],sign[0],cry[0],one[0],two[0],sign[0],fp[0],i[0],n[0]);
product_4bit p1(x[1],i[0],n[0],one[0],two[0],sign[0],fp[1],i[1],n[1]);
product_4bit p2(x[2],i[1],n[1],one[0],two[0],sign[0],fp[2],i[2],n[2]);
product_4bit p3(x[3],i[2],n[2],one[0],two[0],sign[0],fp[3],i[3],n[3]);
xor x1(m[0],i[3],n[3]);
and a1(m[1],two[0],i[3]);
and a2(m[2],one[0],m[0]);
or o1(fp[4],m[1],m[2]);
not n1(fp[5],fp[4]);
assign p[0]=fp[0];

//second product generation
product_4bit q0(x[0],sign[1],cry[1],one[1],two[1],sign[1],sp[0],j[0],o[0]);
product_4bit q1(x[1],j[0],o[0],one[1],two[1],sign[1],sp[1],j[1],o[1]);
product_4bit q2(x[2],j[1],o[1],one[1],two[1],sign[1],sp[2],j[2],o[2]);
product_4bit q3(x[3],j[2],o[2],one[1],two[1],sign[1],sp[3],j[3],o[3]);
xor x2(m[3],j[3],o[3]);
and a3(m[4],two[1],j[3]);
and a4(m[5],one[1],m[3]);
or o2(sp[4],m[4],m[5]);
not n2(sp[5],sp[4]);
assign p[1]=fp[1];

//addition
// 1st lvl add
```

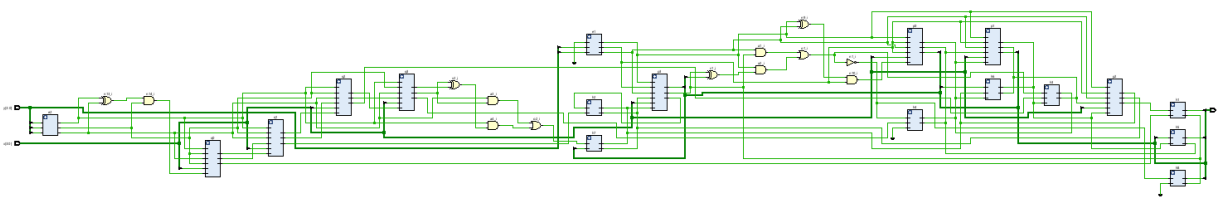
```
HAd h1(fp[2],sp[0],n[3],i[3]);
HAd h2(fp[3],sp[1],n[2],i[2]);
FAd h3(fp[4],sp[2],n[1],n[0],i[1]);
FAd h4(fp[5],sp[3],1'b0,n[0],i[0]);
assign p[2]=n[3];

// 2nd lvl add
HAd h5(fp[0],n[1],n[3],i[0]);
FAd h6(fp[1],n[2],n[0],n[1],i[1]);
HAd h7(sp[4],n[3],n[2],i[2]);
HAd h8(fp[5],1'b0,n[3],i[3]);

endmodule

// generation of codes
module code_4bit(one,two,sign,y2,y1,y0);
input y2,y1,y0;
output one,two,sign;
wire [1:0]k;
xor x1(one,y0,y1);
xor x2(k[1],y2,y1);
not n1(k[0],one);
and a1(two,k[0],k[1]);
assign sign=y2;
endmodule

//generation of inner products
module product_4bit(x1,x0,x2,one,two,sign,p,i,ca);
input x1,x0,x2,sign,one,two;
output p,i,ca;
wire [2:0] k;
xor xo1(i,x1,sign);
and a1(k[1],i,one);
and a0(k[0],x0,two);
or o0(k[2],k[1],k[0]);
xor xo2(p,k[2],x2);
and a2(ca,k[2],x2);
endmodule

//adders design
module HAd(a,b,c,s);
input a,b;
output c,s;
xor x1(s,a,b);
and a1(c,a,b);
endmodule

module FAd(a,b,c,cy,sm);
input a,b,c;
output cy,sm;
wire x,y;
xor x1(x,a,b);
xnor x2(y,a,b);
MUX m1(x,y,c,sm);
MUX m2(a,c,x,cy);
endmodule

module MUX(i0,i1,s,o);
input i0,i1,s;
output o;
wire t,p,q;
```

```
and a1(t,s,i1);
not n0(p,s);
and a2(q,p,i0);
or a3(o,t,q);
endmodule

module FA(a,b,c,cy,sm);
input a,b,c;
output cy,sm;
wire x,y,z;
xor x1(x,a,b);
xor x2(sm,x,c);
and a1(y,a,b);
and a2(z,x,c);
or o1(cy,y,z);
endmodule

module cla(n,z,o,a,b);
input [3:0] a,b;
input o;
wire [3:0] p,g,c;
wire [9:0]m;
output [3:0]n;
output z;
xor (p[0],a[0],b[0]);
and (g[0],a[0],b[0]);
xor (p[1],a[1],b[1]);
and (g[1],a[1],b[1]);
xor (p[2],a[2],b[2]);
and (g[2],a[2],b[2]);
xor (p[3],a[3],b[3]);
and (g[3],a[3],b[3]);

and (m[0],o,p[0]);
or (c[0],m[0],g[0]);
and (m[1],g[0],p[1]);
and (m[2],o,p[0],p[1]);
or (c[1],g[1],m[1],m[2]);
and (m[3],g[1],p[2]);
and (m[4],g[0],p[1],p[2]);
and (m[5],o,p[1],p[2],p[0]);
or (c[2],g[2],m[3],m[4],m[5]);
and (m[6],g[2],p[3]);
and (m[7],g[1],p[2],p[3]);
and (m[8],g[0],p[1],p[2],p[3]);
and (m[9],o,p[0],p[1],p[2],p[3]);
or (c[3],g[3],m[6],m[7],m[8],m[9]);
xor (n[0],p[0],o);
xor (n[1],p[1],c[0]);
xor (n[2],p[2],c[1]);
xor (n[3],p[3],c[2]);
assign z=c[3];
endmodule
```

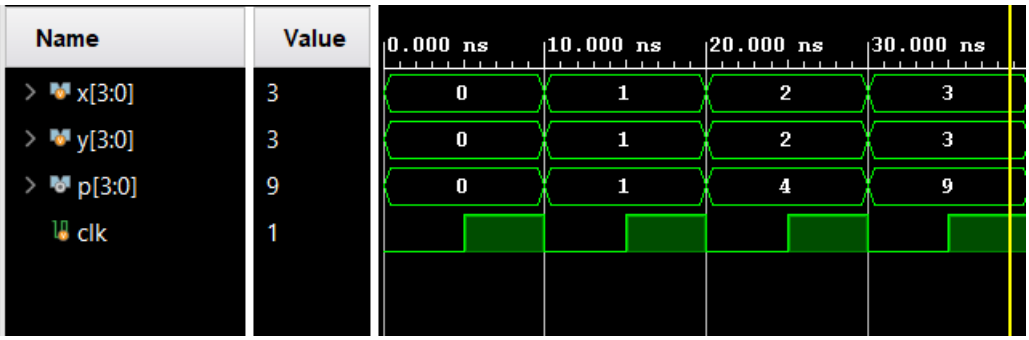
Test bench

```
module hybrid_tb();

    // Inputs
    reg [3:0] x;
    reg [3:0] y;

    // Outputs
    wire [3:0] p;

    // Instantiate the hybrid module
    hybrid dut (
        .x(x),
        .y(y),
        .p(p)
    );

    // Clock generation
    reg clk = 0;
    always #5 clk = ~clk;

    // Stimulus
    initial begin
        // Initialize inputs
        x = 8'b00000000;
        y = 8'b00000000;

        // Apply stimulus
        #10 x = 4'b0001; y = 4'b0001;
        #10 x = 4'b0010; y = 4'b0010;
        #10 x = 4'b0011; y = 4'b0011;
        #10 x = 4'b0100; y = 4'b0100;
        #10 x = 4'b0101; y = 4'b0101;
        #10 $finish;
    end

    // Monitor
    always @(posedge clk) begin
        // Display outputs
        $display("x = %b, y = %b, p = %b", x, y, p);
    end

endmodule
```

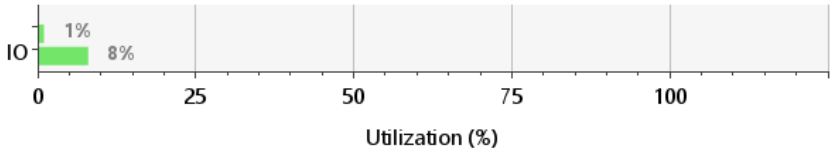
Elaborated Design



Simulation



Resource	Utilization	Available	Utilization %
LUT	21	53200	0.04
IO	16	200	8.00



Summary

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**1.927 W**
Design Power Budget:	**Not Specified**
Process:	typical
Power Budget Margin:	**N/A**
Junction Temperature:	**47.2°C**
Thermal Margin:	37.8°C (3.1 W)
Ambient Temperature:	25.0 °C
Effective θJA:	11.5°C/W