**Digital Signal Design Lab**

Lab CEL-442

Lab Journal: 11



Name: Omar

Class: BCE-06

Enrollment No: 01-132182-024

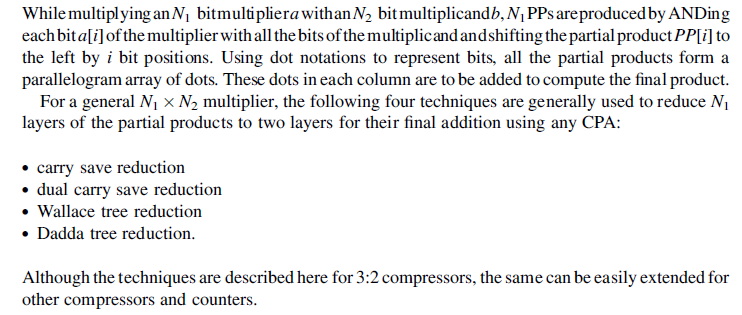
**Lab # 11**

**Basic Building Block Optimization Fast Multiplier design - I**

**Objective** In this lab we are going to design a Fast Multiplier by using Carry Save Reduction scheme.

**Introduction**

**Partial Product Reduction:**

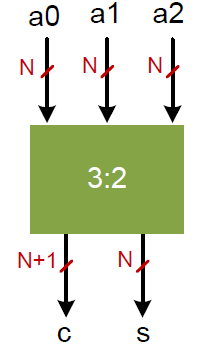
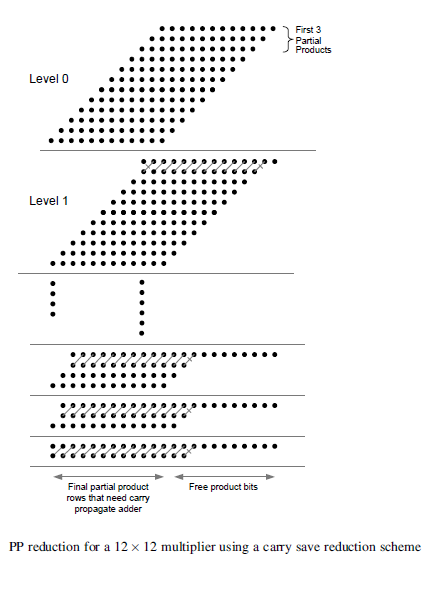


**Carry Save Reduction:**

The first three layers of the PPs are reduced to two layers using carry save addition (CSA). In this reduction process, while generating the next level of logic, isolated bits in a column, in the selected three layers, are simply dropped down to the same column, columns with two bits are reduced to two bits using half adders and the columns with three bits are reduced to two bits using full adders. While adding bits using HAs or FAs, the dot representing the sum bit is dropped down in the same column whereas the dot representing the carry bit is placed in the next significant bit column.

Once the first three partial products are reduced to two layers, the fourth partial product in the original composition is grouped with them to make a new group of three layers. These three layers are again reduced to two layers using the CSA technique. The process is repeated until the entire array is reduced to two layers of numbers. This scheme results in a few least significant product bits (called free product bits), and the rest of the bits appear in two layers, which are then added using any CPA to get the rest of the product bits.

A CSA reduction scheme is shown in the figure below:



**LAB TASK:**

1. Implement an NxN multiplier
2. Implement an NxN Carry Save Adder (CSA)
3. Use the CSA to build (optimize) the NxN multiplier
4. Optimize the multiplier in part 1 using **Dual Carry Save Adder**
5. Optimize the multiplier in part 1 using **Wallace Tree Reduction Technique**

**Task 1:**

Implement a multiplier

**Module:**

`timescale 1ns / 1ps

module mul\_4x4(

input reset,start,

input[3:0] A,B, output reg [7:0] Z,

output wire Finish

);

//reg [7:0] Z; //wire Finish; wire Phi0,Phi1;// 2 phase clocking

wire m1,m2,m3,m4; // state machine reg[3:0] State; // Accumulator reg [8:0] ACC; // Accumulator // logic to create 2 phase clocking when starting nand u0(m1,start,m2); buf #20 u1(m2,m1); buf #10 u2(Phi0,m1);// First phase clocking not #2 u5(m4,Phi0);

assign m3=~m1; and #2 u4(Phi1,m3,m4);// Second phase clocking assign Finish = (State==9)? 1'b1:1'b0; // Finish Flag

// FSM always @(posedge Phi0 or posedge Phi1 or posedge reset) begin if(reset) begin State <= 0;

ACC <= 0; Z <= 0;

end else if((Phi0==1'b1) || (Phi1==1'b1)) begin // 2 phase clocking if(State==0) begin

ACC[8:4] <= 5'b00000; // begin cycle

ACC[3:0] <= A; // Load A State <= 1; end else if(State==1 || State == 3 || State ==5 || State ==7)

// add/shift State

begin if(ACC[0] == 1'b1) begin // add multiplicand

ACC[8:4] <= {1'b0,ACC[7:4]} + B; State <= State + 1; end else begin

ACC <= {1'b0,ACC[8:1]};// shift right

State <= State + 2;

end end else if(State==2 || State == 4 || State ==6 || State ==8)

// shift State

begin

ACC <= {1'b0,ACC[8:1]}; // shift right State <= State + 1; end else if(State == 9) begin

State <= 0; Z <= ACC[7:0]; end end end endmodule

**Testbench:**

`timescale 1ns / 1ps

module test;

Inputs reg reset; reg start; reg [3:0] A; reg [3:0] B;

// Outputs

wire [7:0] Z; wire Finish;

// Instantiate the Unit Under Test (UUT)

mul\_4x4 uut (

.reset(reset),

.start(start),

.A(A), .B(B),

.Z(Z),

.Finish(Finish)

);

initial begin // Initialize Inputs

reset = 1;

#40 start = 0;

#40 A = 10; #40 B = 15;

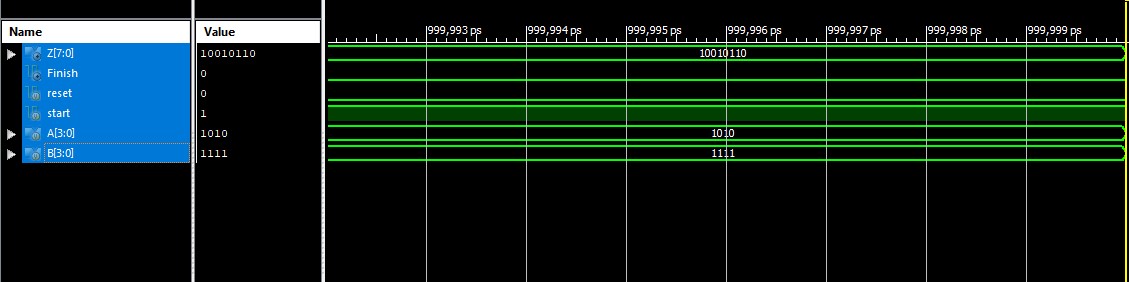
#400 reset = 0;

#40 start = 1;

end

endmodule

**Output:**



**Task 2**

Implement a Carry Save Adder (CSA)

**Module:**

`timescale 1ns / 1ps module HA(a, b, s, c); input a; input b; output wire s; output wire c;

assign {c,s}=a+b;

endmodule

module FA(a,b,cin,s,c); input a,b,cin; output reg s,c;

always @(\*) begin assign {c,s} = a+b+cin; end endmodule

module carrysave(p0,p1,p2,p3,p4,p5,s,c,a,b); output [5:0]p0,p1,p2,p3,p4,p5; output [10:0]s; output [7:0]c; input [5:0]a,b;

wire

d,d1,d2,d3,d4,d5,d6,d7,d8,d9,d10,d11,d12,d13,d14,d15,d16,d17,e1,e2,e3,e4,e5,e6,e7,e8,e9,e10,e11,e1 3,e14,e15,e16,e17;

assign p0=b[0]?a:0; assign p1=b[1]?a:0; assign p2=b[2]?a:0; assign p3=b[3]?a:0; assign p4=b[4]?a:0; assign p5=b[5]?a:0; assign s[0]=p0[0];

HA h1(s[1],d,p0[1],p1[0]);

HA h2(e5,d5,p1[5],p2[4]);

FA m1(e1,d1,p0[2],p1[1],p2[0]);

FA m2(e2,d2,p0[3],p1[2],p2[1]);

FA m3(e3,d3,p0[4],p1[3],p2[2]); FA m4(e4,d4,p0[5],p1[4],p2[3]);

HA h3(e6,d6,p3[1],p4[0]);

HA h4(e11,d11,p4[5],p5[4]);

FA m5(e7,d7,p3[2],p4[1],p5[0]);

FA m6(e8,d8,p3[3],p4[2],p5[1]);

FA m7(e9,d9,p3[4],p4[3],p5[2]);

FA m8(e10,d10,p3[5],p4[4],p5[3]);

HA h5(s[2],d12,d,e1);

FA m9(e13,d13,d1,e2,p3[0]);

FA m10(e14,d14,d2,e3,e6);

FA m11(e15,d15,d3,e4,e7);

FA m12(e16,d16,d4,e5,e8);

FA m13(e17,d17,d5,e6,p2[5]);

HA h6(s[3],c[0],d12,e13);

HA h7(s[4],c[1],d13,e14);

HA h8(s[9],c[6],d10,e11);

HA h9(s[10],c[7],d11,p5[5]);

FA m14(s[5],c[2],d6,d14,e15);

FA m15(s[6],c[3],d7,d15,e16);

FA m16(s[7],c[4],d8,d16,e17);

FA m17(s[8],c[5],d9,d17,e10);

Endmodule

**Testbench**

`timescale 1ns / 1ps

module test;

Inputs reg [5:0] a; reg [5:0] b;

// Outputs wire [5:0] p0; wire [5:0] p1; wire [5:0] p2; wire [5:0] p3; wire [5:0] p4; wire [5:0] p5; wire [10:0] s; wire [7:0] c;

// Instantiate the Unit Under Test (UUT) carrysave uut (

.p0(p0),

.p1(p1),

.p2(p2),

.p3(p3),

.p4(p4),

.p5(p5),

.s(s),

.c(c),

.a(a),

.b(b)

);

initial begin // Initialize Inputs

a = 101010; b = 110011;

// Wait 100 ns for global reset to finish

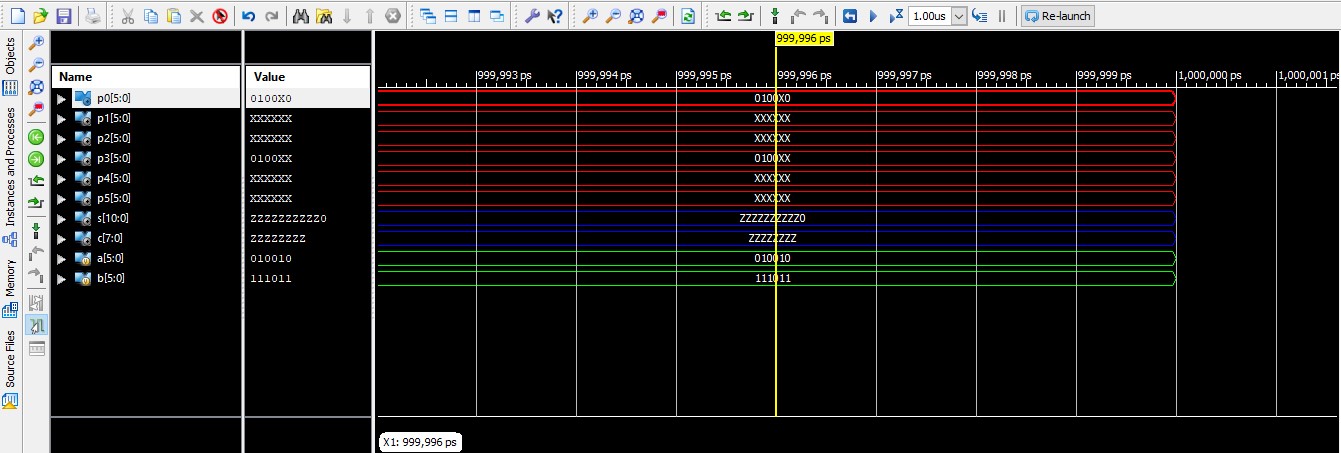
#100;

// Add stimulus here

end

endmodule

**Output:**



**Task 3**

**Module:**

`timescale 1ns / 1ps

module HA(A,B,s,c);

input A,B;

output s;

output c;

assign {c,s}=A+B;

endmodule

module FA(A,B,cin,sum,cout);

input A,B;

input cin;

output sum;

output cout;

wire w1,w3;

wire w2;

HA n1(A,B,w2,w1);

HA n2(cin,w2,sum,w3);

assign cout=w1|w3;

endmodule

module RCA4\_bit(a,b,cin,sum,cout);

input[3:0]a,b;

input cin;

output [3:0]sum;

output cout;

wire c4,c5,c6;

FA F1(a[0],b[0],cin,sum[0],c4),

F2(a[1],b[1],c4,sum[1],c5),

F3(a[2],b[2],c5,sum[2],c6),

F4(a[3],b[3],c6,sum[3],cout);

endmodule

module CSA(A,B,cin,sum,cout);

input [11:0]A,B;

input cin;

output [11:0]sum;

output cout;

wire w1,w2;

wire [11:0]s;

RCA4\_bit A1(A[3:0],B[3:0],cin,s[3:0],w1);

RCA4\_bit A2(A[7:4],B[7:4],w1,s[7:4],w2);

RCA4\_bit A3(A[11:8],B[11:8],w2,s[11:8],cout1);

wire t1,t2,c,c1,c2,cout1,cout2;

wire [11:0]s1;

RCA4\_bit A5(A[3:0],B[3:0],cin,s1[3:0],t1);

RCA4\_bit A6(A[7:4],B[7:4],t1,s1[7:4],t2);

RCA4\_bit A7(A[11:8],B[11:8],t2,s1[11:8],cout2);

assign sum[3:0]=cin?s[3:0]:s1[3:0];

assign c=cin?w1:t1;

assign sum[7:4]=c?s[7:4]:s1[7:4];

assign c1=c?w2:t2;

assign sum[11:8]=c1?s[11:8]:s1[11:8];

assign cout=c1?cout1:cout2;

endmodule

**Testbench**

`timescale 1ns / 1ps

module Task\_3\_Test;

// Inputs

reg [11:0] A;

reg [11:0] B;

reg cin;

// Outputs

wire [11:0] sum;

wire cout;

// Instantiate the Unit Under Test (UUT)

CSA uut (

.A(A),

.B(B),

.cin(cin),

.sum(sum),

.cout(cout)

);

initial begin

// Initialize Inputs

A = 12'b001010001100;

B = 12'b111110010101;

cin = 0;

// Wait 100 ns for global reset to finish

#100;

A = 12'b001010001100;

B = 12'b111110010101;

cin = 1;

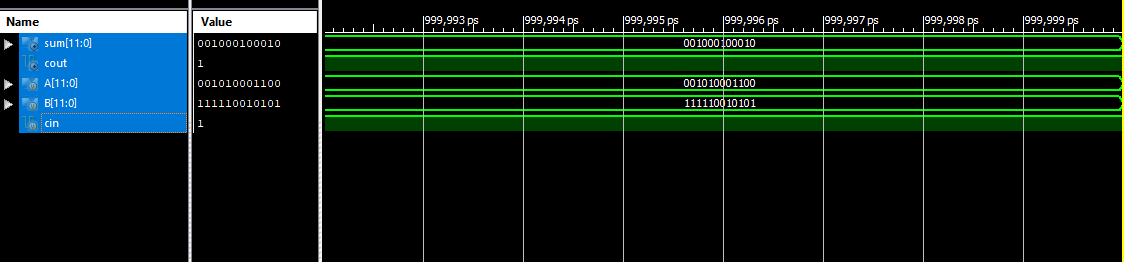
// Wait 100 ns for global reset to finish

#100;

end

endmodule

**OUTPUT**



**Task 4**

**Module:**

`timescale 1ns / 1ps

module HA(A,B,s,c);

input A,B;

output s;

output c;

assign {c,s}=A+B;

endmodule

module FA(A,B,cin,sum,cout);

input A,B;

input cin;

output sum;

output cout;

wire w1,w3;

wire w2;

HA n1(A,B,w2,w1);

HA n2(cin,w2,sum,w3);

assign cout=w1|w3;

endmodule

module RCA4\_bit(a,b,cin,sum,cout);

input[3:0]a,b;

input cin;

output [3:0]sum;

output cout;

wire c4,c5,c6;

FA F1(a[0],b[0],cin,sum[0],c4),

F2(a[1],b[1],c4,sum[1],c5),

F3(a[2],b[2],c5,sum[2],c6),

F4(a[3],b[3],c6,sum[3],cout);

endmodule

module CSA(A,B,cin,sum,cout);

input [11:0]A,B;

input cin;

output [11:0]sum;

output cout;

wire w1,w2;

wire [11:0]s;

RCA4\_bit A1(A[3:0],B[3:0],cin,s[3:0],w1);

RCA4\_bit A2(A[7:4],B[7:4],w1,s[7:4],w2);

RCA4\_bit A3(A[11:8],B[11:8],w2,s[11:8],cout1);

wire t1,t2,c,c1,c2,cout1,cout2;

wire [11:0]s1;

RCA4\_bit A5(A[3:0],B[3:0],cin,s1[3:0],t1);

RCA4\_bit A6(A[7:4],B[7:4],t1,s1[7:4],t2);

RCA4\_bit A7(A[11:8],B[11:8],t2,s1[11:8],cout2);

assign sum[3:0]=cin?s[3:0]:s1[3:0];

assign c=cin?w1:t1;

assign sum[7:4]=c?s[7:4]:s1[7:4];

assign c1=c?w2:t2;

assign sum[11:8]=c1?s[11:8]:s1[11:8];

assign cout=c1?cout1:cout2;

endmodule

**Testbench**

`timescale 1ns / 1ps

module Task\_4Test;

// Inputs

reg [11:0] A;

reg [11:0] B;

reg cin;

// Outputs

wire [11:0] sum;

wire cout;

// Instantiate the Unit Under Test (UUT)

CSA uut (

.A(A),

.B(B),

.cin(cin),

.sum(sum),

.cout(cout)

);

initial begin

// Initialize Inputs

A = 12'b000000000000;

B = 12'b000000000000;

cin = 0;

#10;

A = 12'b000000000000;

B = 12'b000000000000;

cin = 1;

#10;

A = 12'b000000111111;

B = 12'b111111000000;

cin = 0;

#10;

A = 12'b000000111111;

B = 12'b111111000000;

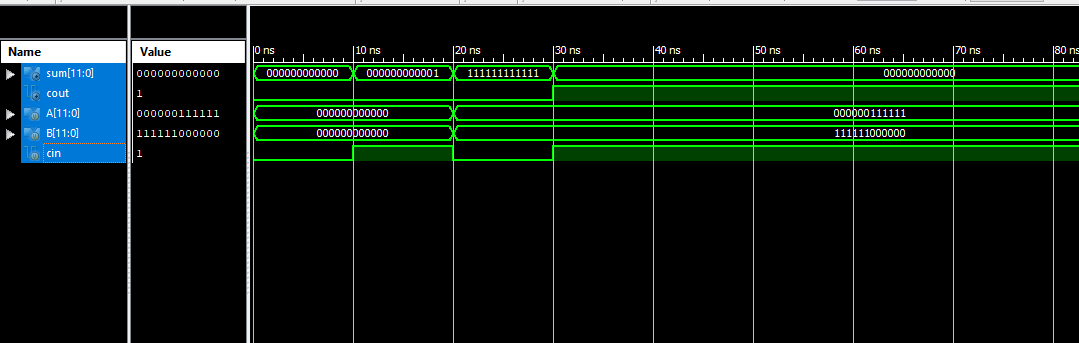
cin = 1;

#10;

end

endmodule

**OUTPUT**



**Task 5**

**Module:**

`timescale 1ns / 1ps

module Ha(a,b,sum,carry);

input a,b;

output sum,carry;

assign sum=a^b;

assign carry=a&b;

endmodule

module Fa(a,b,cin,sum,carry);

input a,b,cin;

output sum,carry;

assign {carry,sum} = a+b+cin;

endmodule

module Wallace(a,b,p,cout);

input [8:0]a,b;

output [17:0]p;

output cout;

reg [8:0] pp [8:0];

wire g1,g2,g3,g4,g5,g6,g7,g8,g9,g10,f1,f2,f3,f4,f5,f6,f7,f8,f9;

wire o1,o2,o3,o4,o5,o6,o7,o8,o9,p1,p2,p3,p4,p5,p6,p7,p8,p9;

wire s1,s2,s3,s4,s5,s6,s7,s8,s9,c1,c2,c3,c4,c5,c6,c7,c8,c9;

wire m1,m2,m3,m4,m5,m6,m7,m8,m9,n1,n2,n3,n4,n5,n6,n7,n8,n9;

wire h1,h2,h3,h4,h5,h6,h7,h8,h9,h10,h11,i1,i2,i3,i4,i5,i6,i7,i8,i9,i10,i11;

wire u1,u2,u3,u4,u5,u6,u7,u8,ju9,u10,u11,u12,u13,v1,v2,v3,v4,v5,v6,v7,v8,v9,v10,v11,v12,v13;

wire r1,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11,r12;

wire j1,j2,j3,j4,j5,j6,j7,j8,j9,j10,j11,k1,k2,k3,k4,k5,k6,k7,k8,k9,k10,k11;

integer i,j;

always @\*

begin

for(i=0; i<9; i= i+1)

begin

for(j=0;j<9; j=j+1)

begin

pp[i][j]=a[i]\*b[j];

end

end

end

assign p[0]=pp[0][0];

Ha HA1(pp[0][1],pp[1][0],p[1],c1);

Fa FA1(pp[0][2],pp[1][1],pp[2][0],s2,c2),

FA2(pp[0][3],pp[1][2],pp[2][1],s3,c3),

FA3(pp[0][4],pp[1][3],pp[2][2],s4,c4),

FA4(pp[0][5],pp[1][4],pp[2][3],s5,c5),

FA5(pp[0][6],pp[1][5],pp[2][4],s6,c6),

FA6(pp[0][7],pp[1][6],pp[2][5],s7,c7),

FA7(pp[0][8],pp[1][7],pp[2][6],s8,c8);

Ha HA2(pp[1][8],pp[2][7],s9,c9);

Ha ha3(pp[3][1],pp[4][0],m1,n1);

Fa fa8(pp[3][2],pp[4][1],pp[5][0],m2,n2),

fa9(pp[3][3],pp[4][2],pp[5][1],m3,n3),

fa10(pp[3][4],pp[4][3],pp[5][2],m4,n4),

fa11(pp[3][5],pp[4][4],pp[5][3],m5,n5),

fa12(pp[3][6],pp[4][5],pp[5][4],m6,n6),

fa13(pp[3][7],pp[4][6],pp[5][5],m7,n7),

fa14(pp[3][8],pp[4][7],pp[5][6],m8,n8);

Ha ha4(pp[4][8],pp[5][7],m9,n9);

Ha ha5(pp[6][1],pp[7][0],g1,f1);

Fa fa15(pp[6][2],pp[7][1],pp[8][0],g2,f2),

fa16(pp[6][3],pp[7][2],pp[8][1],g3,f3),

fa17(pp[6][4],pp[7][3],pp[8][2],g4,f4),

fa18(pp[6][5],pp[7][4],pp[8][3],g5,f5),

fa19(pp[6][6],pp[7][5],pp[8][4],g6,f6),

fa20(pp[6][7],pp[7][6],pp[8][5],g7,f7),

fa21(pp[6][8],pp[7][7],pp[8][6],g8,f8);

Ha ha6(pp[7][8],pp[8][7],g9,f9);

Ha ha7(s2,c1,p[2],p1);

Fa fa22(s3,c2,pp[3][0],o2,p2),

fa23(s4,c3,m1,o3,p3),

fa24(s5,c4,m2,o4,p4),

fa25(s6,c5,m3,o5,p5),

fa26(s7,c6,m4,o6,p6),

fa27(s8,c7,m5,o7,p7),

fa28(s9,c8,m6,o8,p8),

fa29(pp[2][8],c9,m7,o9,p9);

Ha ha8(n2,pp[6][0],h1,i1),

ha9(n3,g1,h2,i2);

Fa fa30(n4,g2,f1,h3,i3),

fa31(n5,g3,f2,h4,i4),

fa32(n6,g4,f3,h5,i5),

fa33(n7,g5,f4,h6,i6),

fa34(n8,g6,f5,h7,i7),

fa35(n9,g7,f6,h8,i8);

Ha ha10(g8,f7,h9,i9),

ha11(g9,f8,h10,i10),

ha12(pp[8][8],f9,h11,i11);

Ha ha13(o2,p1,p[3],k1),

ha14(o3,p2,j2,k2);

Fa fa36(o4,p3,n1,j3,k3),

fa37(o5,p4,h1,j4,k4),

fa38(o6,p5,h2,j5,k5),

fa39(o7,p6,h3,j6,k6),

fa40(o8,p7,h4,j7,k7),

fa41(o9,p8,h5,j8,k8),

fa42(m8,p9,h6,j9,k9);

Ha ha15(m9,h7,j10,k10),

ha16(pp[5][8],h8,j11,k11);

Ha ha17(j2,k1,p[4],v1),

ha18(j3,k2,u2,v2),

ha19(j4,k3,u3,v3);

Fa fa43(j5,k4,i1,u4,v4),

fa44(j6,k5,i2,u5,v5),

fa45(j7,k6,i3,u6,v6),

fa46(j8,k7,i4,u7,v7),

fa47(j9,k8,i5,u8,v8),

fa48(j10,k9,i6,u9,v9),

fa49(j11,k10,i7,u10,v10),

fa50(h9,k11,i8,u11,v11);

Ha ha20(h10,i9,u12,v12),

ha21(h11,i10,u13,v13);

Ha A1(u2,v1,p[5],r1);

Fa A2(u3,v2,r1,p[6],r2),

A3(u4,v3,r2,p[7],r3),

A4(u5,v4,r3,p[8],r4),

A5(u6,v5,r4,p[9],r5),

A6(u7,v6,r5,p[10],r6),

A7(u8,v7,r6,p[11],r7),

A8(u9,v8,r7,p[12],r8),

A9(u10,v9,r8,p[13],r9),

A10(u11,v10,r9,p[14],r10),

A11(u12,v11,r10,p[15],r11),

A12(u13,v12,r11,p[16],r12),

A13(i11,v13,r12,p[17],cout);

endmodule

**Testbench**

`timescale 1ns / 1ps

module Task\_5\_Test;

// Inputs

reg [8:0] a;

reg [8:0] b;

// Outputs

wire [17:0] p;

wire cout;

// Instantiate the Unit Under Test (UUT)

Wallace uut (

.a(a),

.b(b),

.p(p),

.cout(cout)

);

initial begin

// Initialize Inputs

a = 5;

b = 4;

// Wait 100 ns for global reset to finish

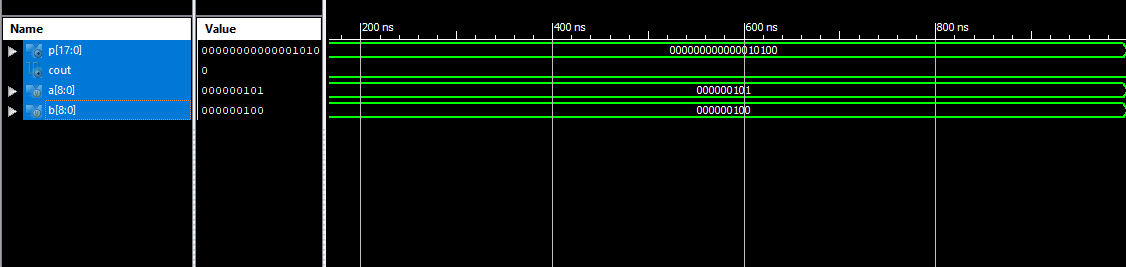
#100;

// Add stimulus here

end

endmodule

**OUTPUT**



**Conclusion: -**

In this lab we learned to design a Fast Multiplier by using Carry Save Reduction scheme.