8\*8 Wallace Tree Multiplier

Understanding the problem:

* Our goal is to design 8\*8 Wallace Tree Multiplier which would multiply two 8-bit inputs.
* It was observed on multiplication that maximum output bits are 16.
* Hence, output array is created as 16-bit bus.

Devising the plan:

* We used AND gates to generate partial products.
* We added these partial products as well as carrys of previous stages using full adders and half adders.
* Here, ‘stage’ refers to a column of a half adder and full adders whose final output is a bit of the final product. For example, p[0].
* For every stage, first two partial products were added using a half adder. Then, remaining partial products were added using immediate lower full adders.
* This ensured that set of half adders would work simultaneously as well as immediate lower set of full adders would work simultaneously with minimum delay.
* Also, Carrys of the previous stages were distributed in lower sets of full adders as they were already waiting for the sum to arrive from the upper set of full adders. Thus, timings of the inputs were almost matched.
* Thus, we ensured that series cascading of adders would occur in lower stages only where number of adders is already less
* Hence, total delay decreased.

Carrying out the plan:

Verilog Code:

module FA(a,b,c\_in,sum,c\_out);

input a;

input b;

input c\_in;

output sum;

output c\_out;

wire y1;

reg y2,y3,y4;

xor ( y1,a,b); // Structural Modelling

xor (sum,y1,c\_in);

always @ (a,b,c\_in) // Behavioural Modelling

begin

y2=a&b;

y3=b&c\_in;

y4=a&c\_in;

end

assign c\_out=y2|y3|y4; // Dataflow Modelling

endmodule

module HA(

input a,

input b,

output reg sum,

output reg c\_out

);

always@(a,b)

begin

sum=a^b;

c\_out=a&b;

end

endmodule

module WTM(

input [7:0] x,

input [7:0] y,

output [15:0] p

);

wire a,b1,b2,c1,c2,c3,d1,d2,d3,d4,e1,e2,e3,e4,e5,f1,f2,f3,f4,f5,f6,g1,g2,g3,g4,g5,g6,g7,h1,h2,h3,h4,h5,h6,h7,h8,i1,i2,i3,i4,i5,i6,i7,j1,j2,j3,j4,j5,j6,k1,k2,k3,k4,k5,l1,l2,l3,l4,m1,m2,m3,n1,n2,o1;

wire ca1,s1,cb1,s2,cc1,cb2,s3,cc2,cc3,s4,cd1,s5,cd2,s6,cd3,cd4,s7,ce1,s8,ce2,s9,ce3,s10,ce4,ce5,s11,cf1,s12,cf2,s13,cf3,s14,cf4,s15,cf5;

wire cf6,s16,cg1,s17,cg2,s18,cg3,s19,cg4,s20,cg5,s21,cg6,cg7,s22,ch1,s23,ch2,s24,ch3,s25,ch4,s26,ch5,s27,ch6,ch7,s28,ci1,s29,ci2;

wire s30,ci3,s31,ci4,s32,ci5,s33,ci6,ci7,s34,cj1,s35,cj2,s36,cj3,s37,cj4,s38,cj5,cj6,s39,ck1,s40,ck2,s41,ck3,s42,ck4;

wire ck5,s43,cl1,s44,cl2,s45,cl3,cl4,s46,cm1,s47,cm2,cm3,s48,cn1,cn2,co1;

assign a=x[0]&y[0];

assign b1=x[1]&y[0];

assign b2=x[0]&y[1];

assign c1=x[2]&y[0];

assign c2=x[1]&y[1];

assign c3=x[0]&y[2];

assign d1=x[3]&y[0];

assign d2=x[2]&y[1];

assign d3=x[1]&y[2];

assign d4=x[0]&y[3];

assign e1=x[4]&y[0];

assign e2=x[3]&y[1];

assign e3=x[2]&y[2];

assign e4=x[1]&y[3];

assign e5=x[0]&y[4];

assign f1=x[5]&y[0];

assign f2=x[4]&y[1];

assign f3=x[3]&y[2];

assign f4=x[2]&y[3];

assign f5=x[1]&y[4];

assign f6=x[0]&y[5];

assign g1=x[6]&y[0];

assign g2=x[5]&y[1];

assign g3=x[4]&y[2];

assign g4=x[3]&y[3];

assign g5=x[2]&y[4];

assign g6=x[1]&y[5];

assign g7=x[0]&y[6];

assign h1=x[7]&y[0];

assign h2=x[6]&y[1];

assign h3=x[5]&y[2];

assign h4=x[4]&y[3];

assign h5=x[3]&y[4];

assign h6=x[2]&y[5];

assign h7=x[1]&y[6];

assign h8=x[0]&y[7];

assign i1=x[7]&y[1];

assign i2=x[6]&y[2];

assign i3=x[5]&y[3];

assign i4=x[4]&y[4];

assign i5=x[3]&y[5];

assign i6=x[2]&y[6];

assign i7=x[1]&y[7];

assign j1=x[7]&y[2];

assign j2=x[6]&y[3];

assign j3=x[5]&y[4];

assign j4=x[4]&y[5];

assign j5=x[3]&y[6];

assign j6=x[2]&y[7];

assign k1=x[7]&y[3];

assign k2=x[6]&y[4];

assign k3=x[5]&y[5];

assign k4=x[4]&y[6];

assign k5=x[3]&y[7];

assign l1=x[7]&y[4];

assign l2=x[6]&y[5];

assign l3=x[5]&y[6];

assign l4=x[4]&y[7];

assign m1=x[7]&y[5];

assign m2=x[6]&y[6];

assign m3=x[5]&y[7];

assign n1=x[7]&y[6];

assign n2=x[6]&y[7];

assign o1=x[7]&y[7];

assign p[0]=a;

HA inst\_1(b1,b2,p[1],ca1);

HA inst\_2(c1,c2,s1,cb1);

FA inst\_3(s1,c3,ca1,p[2],cb2);

HA inst\_4(d1,d2,s2,cc1);

FA inst\_5(s2,d4,d3,s3,cc2);

FA inst\_6(cb2,cb1,s3,p[3],cc3);

HA inst\_7(e1,e2,s4,cd1);

FA inst\_8(s4,e4,e3,s5,cd2);

FA inst\_9(s5,e5,cc1,s6,cd3);

FA inst\_10(cc2,cc3,s6,p[4],cd4);

HA inst\_11(f1,f2,s7,ce1);

FA inst\_12(s7,f4,f3,s8,ce2);

FA inst\_13(s8,f5,f6,s9,ce3);

FA inst\_14(s9,cd1,cd2,s10,ce4);

FA inst\_15(s10,cd3,cd4,p[5],ce5);

HA inst\_16(g1,g2,s11,cf1);

FA inst\_17(s11,g3,g4,s12,cf2);

FA inst\_18(s12,g5,g6,s13,cf3);

FA inst\_19(s13,g7,ce1,s14,cf4);

FA inst\_20(s14,ce2,ce3,s15,cf5);

FA inst\_21(s15,ce4,ce5,p[6],cf6);

HA inst\_22(h1,h2,s16,cg1);

FA inst\_23(s16,h3,h4,s17,cg2);

FA inst\_24(s17,h5,h6,s18,cg3);

FA inst\_25(s18,h7,h8,s19,cg4);

FA inst\_26(s19,cf1,cf2,s20,cg5);

FA inst\_27(s20,cf3,cf4,s21,cg6);

FA inst\_28(s21,cf5,cf6,p[7],cg7);

HA inst\_29(i1,i2,s22,ch1);

FA inst\_30(s22,i3,i4,s23,ch2);

FA inst\_31(s23,i5,i6,s24,ch3);

FA inst\_32(s24,i7,cg1,s25,ch4);

FA inst\_33(s25,cg2,cg3,s26,ch5);

FA inst\_34(cg4,cg5,s26,s27,ch6);

FA inst\_35(s27,cg6,cg7,p[8],ch7);

HA inst\_36(j1,j2,s28,ci1);

FA inst\_37(s28,j3,j4,s29,ci2);

FA inst\_38(s29,j5,j6,s30,ci3);

FA inst\_39(s30,ch1,ch2,s31,ci4);

FA inst\_40(s31,ch3,ch4,s32,ci5);

FA inst\_41(ch5,ch6,s32,s33,ci6);

HA inst\_42(s33,ch7,p[9],ci7);

HA inst\_43(k1,k2,s34,cj1);

FA inst\_44(s34,k3,k4,s35,cj2);

FA inst\_45(s35,k5,ci1,s36,cj3);

FA inst\_46(s36,ci2,ci3,s37,cj4);

FA inst\_47(s37,ci4,ci5,s38,cj5);

FA inst\_48(s38,ci6,ci7,p[10],cj6);

HA inst\_49(l1,l2,s39,ck1);

FA inst\_50(s39,l3,l4,s40,ck2);

FA inst\_51(s40,cj1,cj2,s41,ck3);

FA inst\_52(s41,cj3,cj4,s42,ck4);

FA inst\_53(cj5,cj6,s42,p[11],ck5);

HA inst\_54(m1,m2,s43,cl1);

FA inst\_55(s43,m3,ck1,s44,cl2);

FA inst\_56(s44,ck2,ck3,s45,cl3);

FA inst\_57(s45,ck4,ck5,p[12],cl4);

HA inst\_58(n1,n2,s46,cm1);

FA inst\_59(s46,cl1,cl2,s47,cm2);

FA inst\_60(s47,cl3,cl4,p[13],cm3);

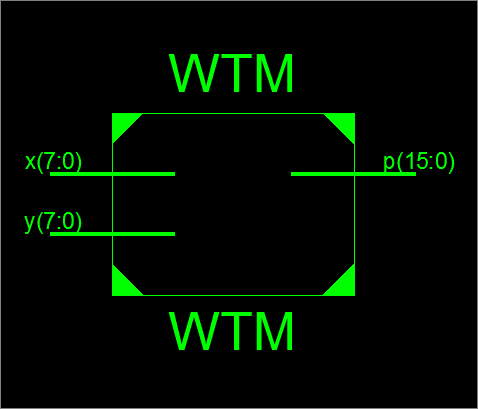
HA inst\_61(o1,cm1,s48,cn1);

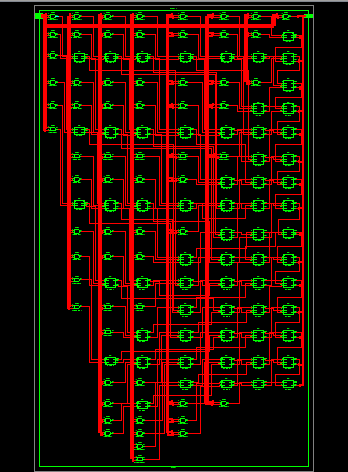
FA inst\_62(s48,cm2,cm3,p[14],cn2);

HA inst\_63(cn1,cn2,p[15],co1);

endmodule

RTL schematic:





Test bench:

module WTM\_tb;

// Inputs

reg [7:0] x;

reg [7:0] y;

// Outputs

wire [15:0] p;

// Instantiate the Unit Under Test (UUT)

WTM uut (

.x(x),

.y(y),

.p(p)

);

initial begin

// Initialize Inputs

x = 0;

y = 0;

// Wait 100 ns for global reset to finish

#100;

x=8'b11111111;

y=8'b11111111;

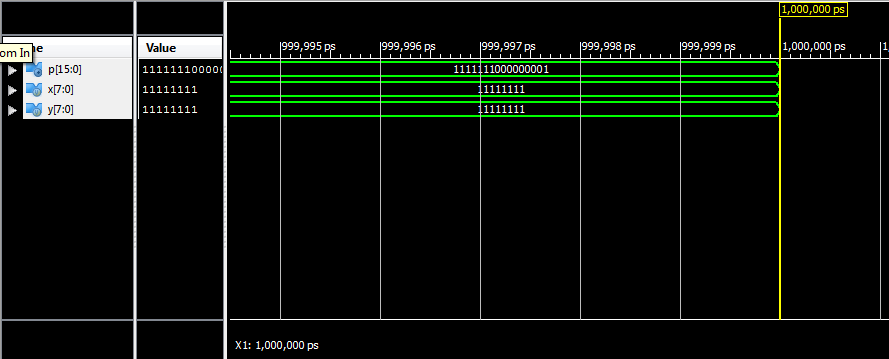
#100;

// Add stimulus here

end

endmodule

Output:



Looking Back:

* Earlier, for every stage, we used lower carrys of previous stages in upper full adders and distributed partial products throughout the stage.
* This design resulted in total delay of around 45ns.
* But, when we used above mentioned designing strategy, it decreased total delay to 25.875ns.
* Thus, we understood the importance of matching of arrival time of all the inputs in every stage.
* This design worked correctly for the overflow condition as seen above.

Synthesis Report:

Design Statistics

# IOs : 32

Cell Usage :

# BELS : 138

# LUT2 : 17

# LUT3 : 49

# LUT4 : 72

# IO Buffers : 32

# IBUF : 16

# OBUF : 16

Device utilization summary:

---------------------------

Selected Device : 3s200pq208-5

Number of Slices: 79 out of 1920 4%

Number of 4 input LUTs: 138 out of 3840 3%

Number of IOs: 32

Number of bonded IOBs: 32 out of 141 22%

Delay: 25.875ns (Levels of Logic = 15)

Source: x<1> (PAD)

Destination: p<14> (PAD)

Data Path: x<1> to p<14>

Gate Net

Cell:in->out fanout Delay Delay Logical Name (Net Name)

---------------------------------------- ------------

IBUF:I->O 14 0.715 1.304 x\_1\_IBUF (x\_1\_IBUF)

LUT4:I0->O 2 0.479 1.040 inst\_1/c\_out1 (ca1)

LUT4:I0->O 2 0.479 1.040 inst\_3/c\_out1 (cb2)

LUT3:I0->O 2 0.479 1.040 inst\_6/c\_out1 (cc3)

LUT3:I0->O 2 0.479 1.040 inst\_10/c\_out1 (cd4)

LUT3:I0->O 2 0.479 1.040 inst\_15/c\_out1 (ce5)

LUT3:I0->O 2 0.479 1.040 inst\_21/c\_out1 (cf6)

LUT3:I0->O 3 0.479 0.830 inst\_28/c\_out1 (cg7)

LUT4:I2->O 2 0.479 1.040 inst\_42/c\_out1 (ci7)

LUT3:I0->O 2 0.479 1.040 inst\_48/c\_out1 (cj6)

LUT3:I0->O 2 0.479 1.040 inst\_53/c\_out1 (ck5)

LUT3:I0->O 2 0.479 1.040 inst\_57/c\_out1 (cl4)

LUT3:I0->O 2 0.479 0.804 inst\_60/c\_out1 (cm3)

LUT4:I2->O 1 0.479 0.681 inst\_62/Mxor\_sum\_xo<0>1 (p\_14\_OBUF)

OBUF:I->O 4.909 p\_14\_OBUF (p<14>)

----------------------------------------

Total **25.875ns** (11.851ns logic, 14.024ns route)

(45.8% logic, 54.2% route)