EHB436E

DIGITAL SYSTEM DESIGN APPLICATIONS

Muhammed Erkmen

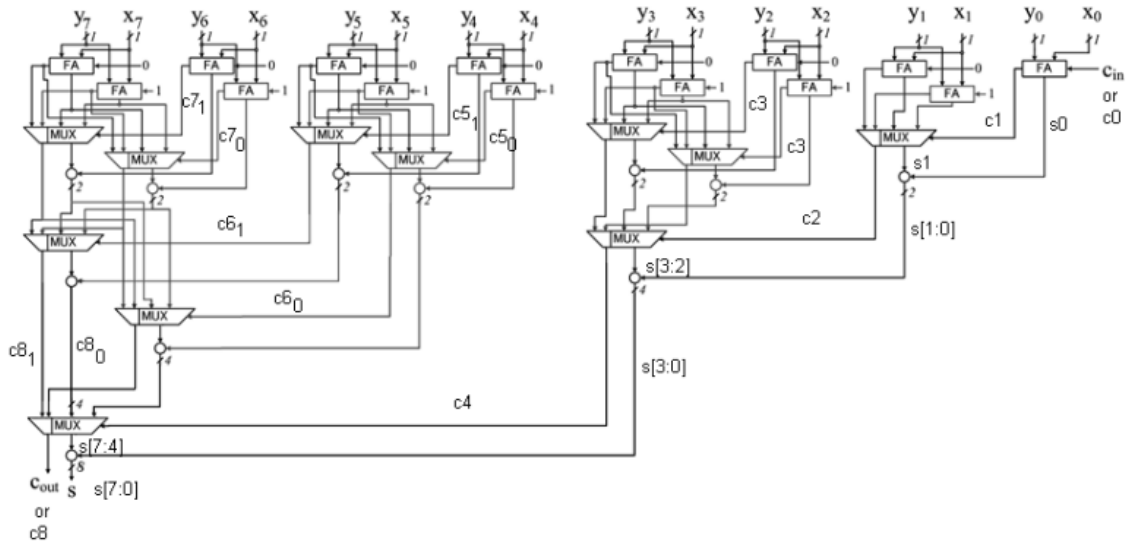
040170049

32-BIT CONDITIONAL SUM ADDER

PROJECT-1 REPORT

Conditional Sum Adder:

Idea of conditional sum adder is, calculating every possible carry and sum outputs for each one of 2-bits, then operating a selection with the previous bits' output.



First step is calculating every possible outputs for every 2-bit addition. To implement this operation, we use full adders. After that, for example for the x3 and y3 addition, we need to select which carry will come, so we use 2 multiplexers and give to these multiplexers as a selection bit x2 and y2 output carry. Then, we'll need to select which carry will come to x2 and y2 adding operation. And that is the third multiplexer's selection input is x1 and y1 adding operation's carry out which selected by x0 and y0.

My designing steps:

So i designed this 32-bit conditional sum adder, first i designed an 8-bit conditional sum adder. If i designed it just 32-bit conditional without parsing the process, i would need more multiplexers. Because first 8 bit stage has 7 multiplexers, but if i designed it with connecting more than 8 bits, i would need more than 7*4=28 multiplexers to give output. This design has less utilization, also i think the same amount of delay because if it was 32 bit connected, the next stage of 3 already needs to wait first 8 bits cout. So I think this design is better.

First stage is calculating the every possible output available, i did that by using full adders.

I put 2*k'th bits outputs to the muxes, then i selected the value i will need with k'th bits couts and added kth bits sum to the selected sum.

I did same thing in 3rd stage mux too. So i get 8 bit sum and 1 bit carry out.

After that I used 4 8-bit conditional sum adder by connected. Then connected the previous carry out to the next one. Included their sum values in a 32 bit wire respectively. And carry out of 4th conditional sum adder is the carry out of the system. **But because of i am designing this signed, this carry out has no meanings i guess.**

Adding overflow output is kind of changing the design. Because overflow output equals to last 2 carry's xor result. But in our design 2nd last carry comes from selections and its source is too above of the circuit.

Because of that reason i made this part behavioral, not structural.

```
always @(*) begin
if( (x+y > 2147483646) || (x+y<-2147483647))
v<=1;
else
v<=0;
end
```

Preparing a testbench:

I designed a testbench with 100 random binary pairs by using $readmemb() function in testbench. I get random binary numbers from: https://www.browserling.com/tools/random-bin. I can write a python or c code or try to use $random command in verilog testbench, but this would cost more time than using this website. But in assignment files a python file is needed so i created a python function to create binary numbers too.

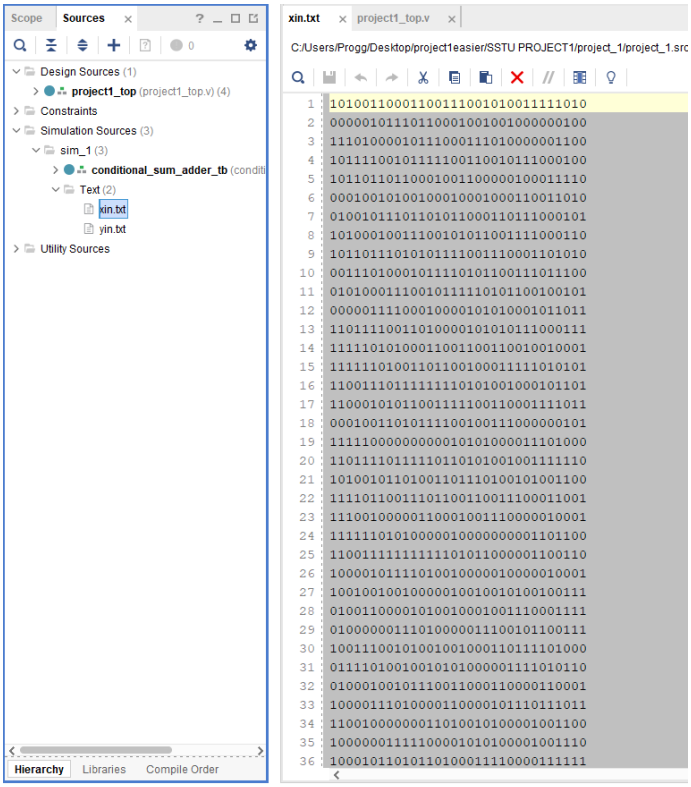
Python Code:

```
import random

def randombinary(b):
    reg = ""
    for i in range(b):
        temp = str(random.randint(0, 1))
        reg += temp
    return(reg)
file = open("output.txt","w")

# Driver Code
n = 32
for i in range(100):
    randombinarynumber = randombinary(n)
    file.write(randombinarynumber + "\n")
    print(randombinarynumber)
```

After getting these numbers, made copy-paste to .txt files named xin,yin. Then added these files to simulation sources in Vivado.



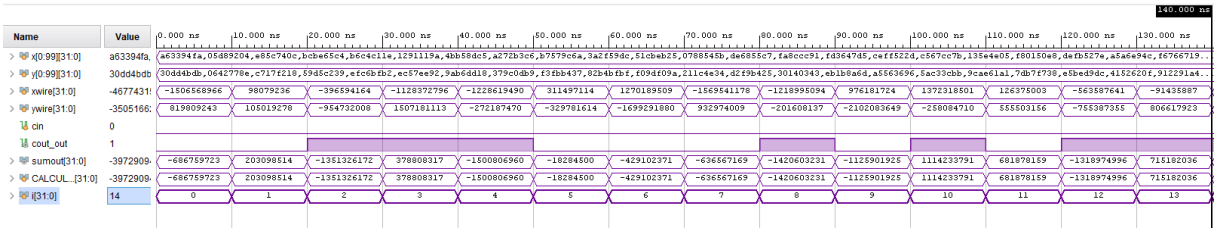
I read these datas and assigned values to [31:0] x [0:99] and [31:0] y [0:99] signed registers.

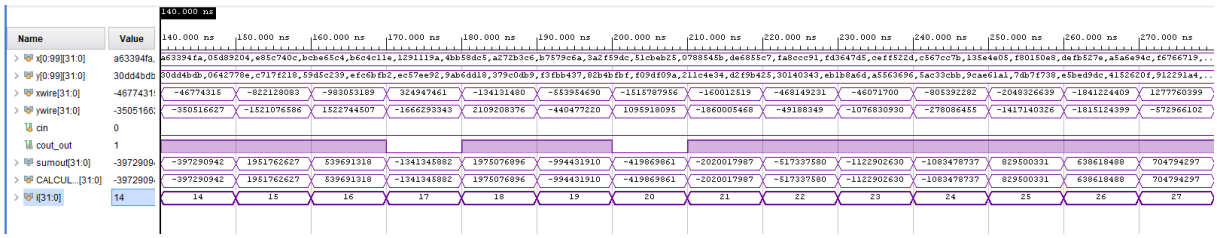
```
initial $readmemb("xin.txt",x);
initial $readmemb("yin.txt",y);
```

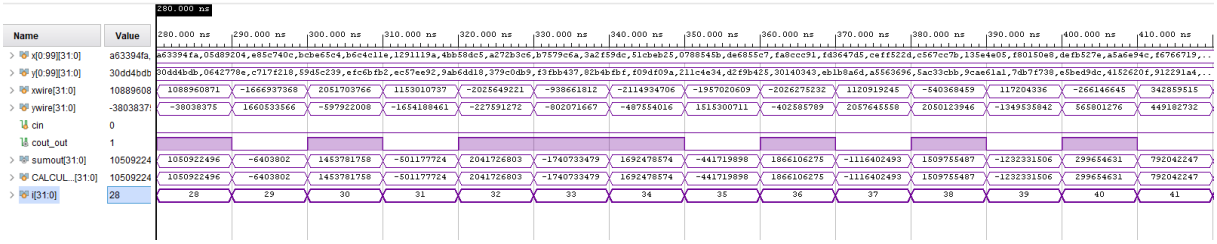
Just initial statements are left. I set a CALCULATOR register which gives directly x+y+cin value in testbench. In every for loop, x and y values goes to my top module and calculator calculates the true result. Then an if block controls that is the output of module (sumout) equal to CALCULATOR value. If these values are equal, there will be a line in TCL console as: x: *xvalue*, y: *yvalue*, result: *sumout*

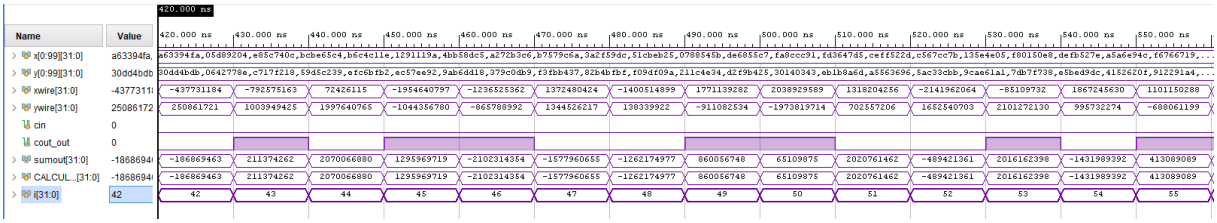
```
for (i=0;i<=99;i=i+1)
begin
 xwire = x[i];
 ywire = y[i];
 CALCULATOR = x[i]+y[i]+cin;
 #5;
 if(CALCULATOR == sumout)
 $display ("x:%d , y: %d, result: %d", xwire,ywire,sumout);
 else
 $display("operation is wrong");
 #5;
end
end
```

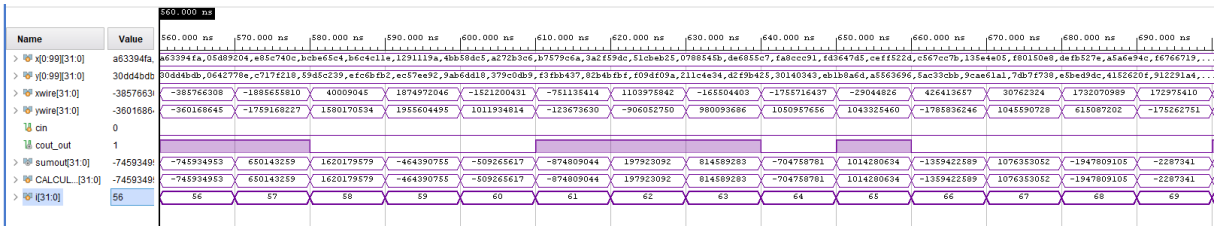
Waveform outputs:

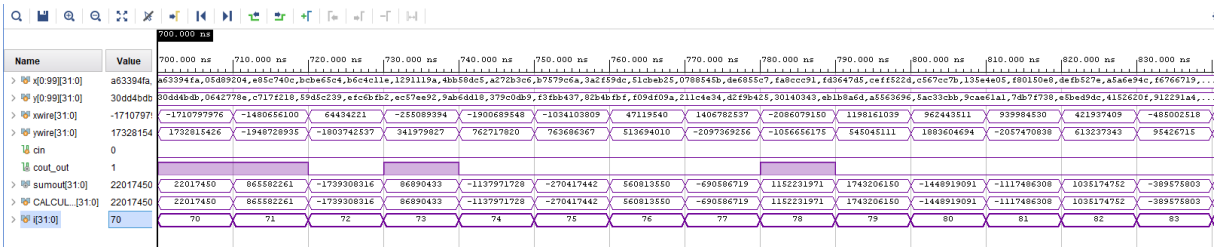


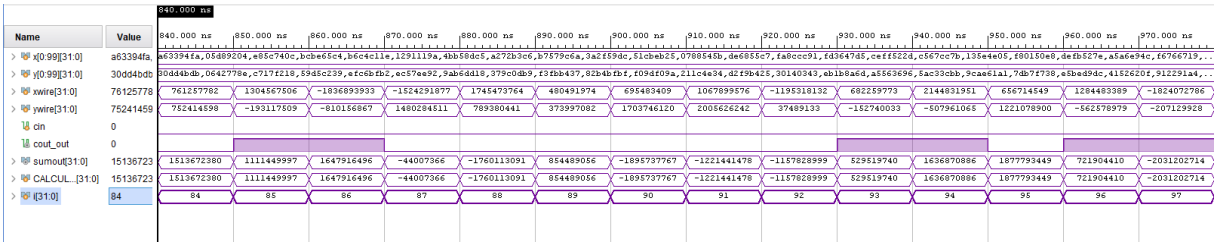


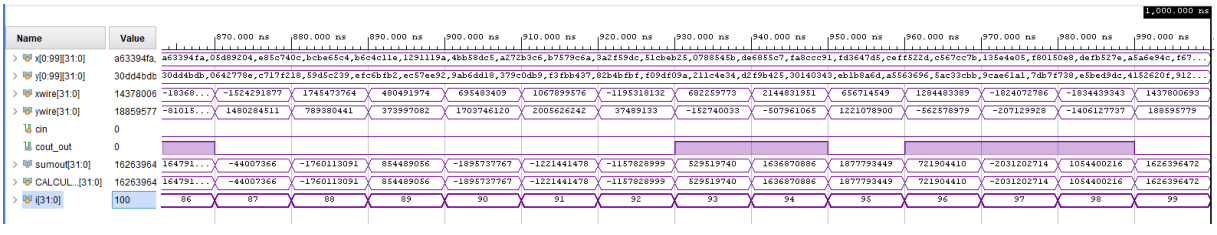

280.000 ns
Name
Value
x[0:99][31:0]
y[0:99][31:0]
xwire[31:0]
ywire[31:0]
cin
cout_out
sumout[31:0]
CALCUL...[31:0]
i[31:0]
28
29
30
31
32
33
34
35
36
37
38
39
40
41


420.000 ns
Name
Value
x[0:99][31:0]
y[0:99][31:0]
xwire[31:0]
ywire[31:0]
cin
cout_out
sumout[31:0]
CALCUL...[31:0]
i[31:0]
42
43
44
45
46
47
48
49
50
51
52
53
54
55

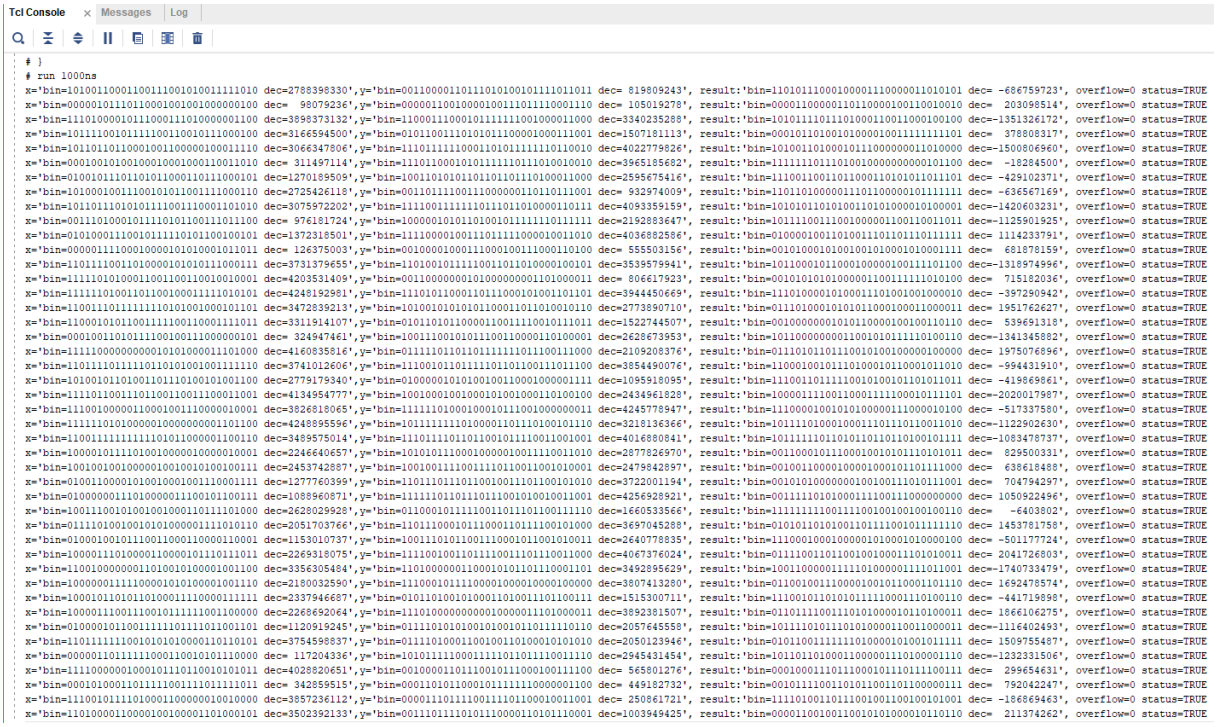

560.000 ns
Name
Value
x[0:99][31:0]
y[0:99][31:0]
xwire[31:0]
ywire[31:0]
cin
cout_out
sumout[31:0]
CALCUL...[31:0]
i[31:0]
56
57
58
59
60
61
62
63
64
65
66
67
68
69


700.000 ns
Name
Value
x[0:99][31:0]
y[0:99][31:0]
xwire[31:0]
ywire[31:0]
cin
cout_out
sumout[31:0]
CALCUL...[31:0]
i[31:0]
70
71
72
73
74
75
76
77
78
79
80
81
82
83


840.000 ns
Name
Value
x[0:99][31:0]
y[0:99][31:0]
xwire[31:0]
ywire[31:0]
cin
cout_out
sumout[31:0]
CALCUL...[31:0]
i[31:0]
84
85
86
87
88
89
90
91
92
93
94
95
96
97


1,000.000 ns
Name
Value
x[0:99][31:0]
y[0:99][31:0]
xwire[31:0]
ywire[31:0]
cin
cout_out
sumout[31:0]
CALCUL...[31:0]
i[31:0]
100
86
87
88
89
90
91
92
93
94
95
96
97
98
99

TCL Console output when simulation applied:

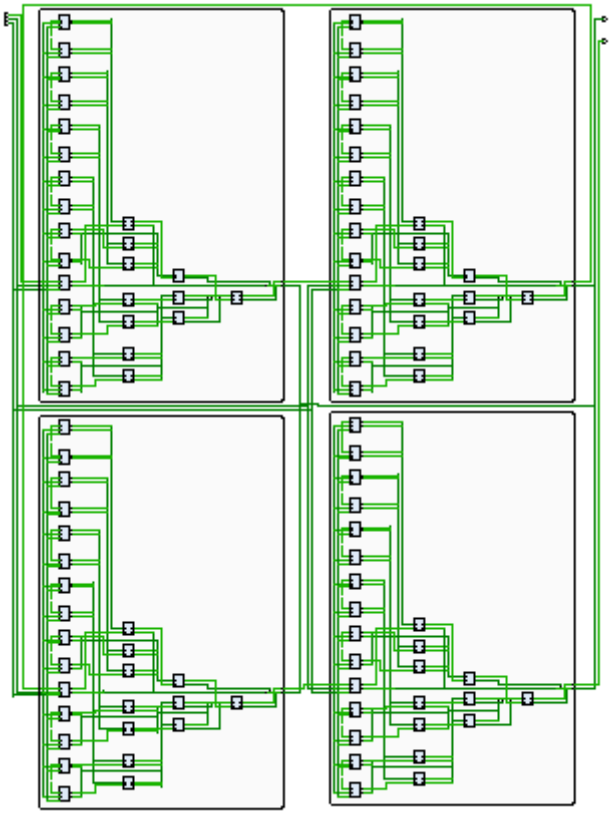


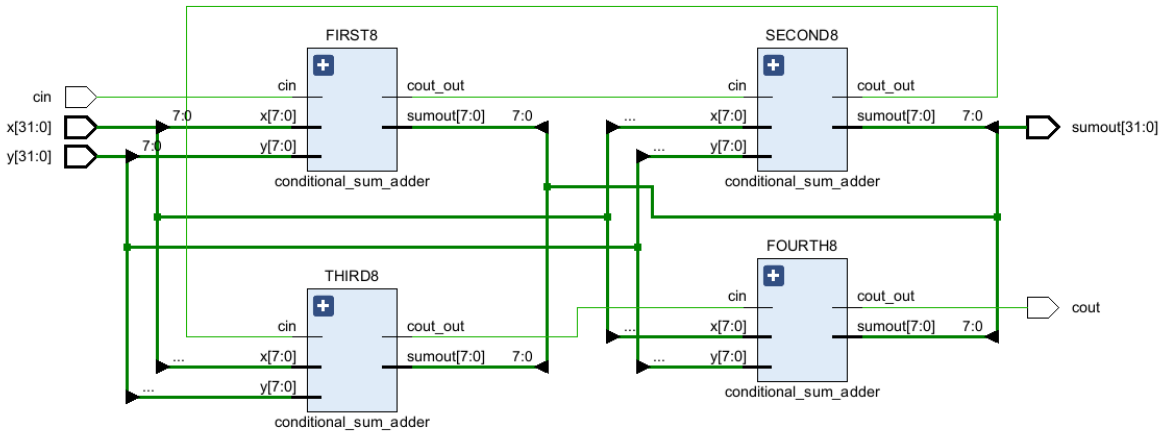
I writed these outputs to a txt file too.

Implementation Reports:

So i put top of my module (* DONT_TOUCH = "TRUE" *) statement then i started implementation

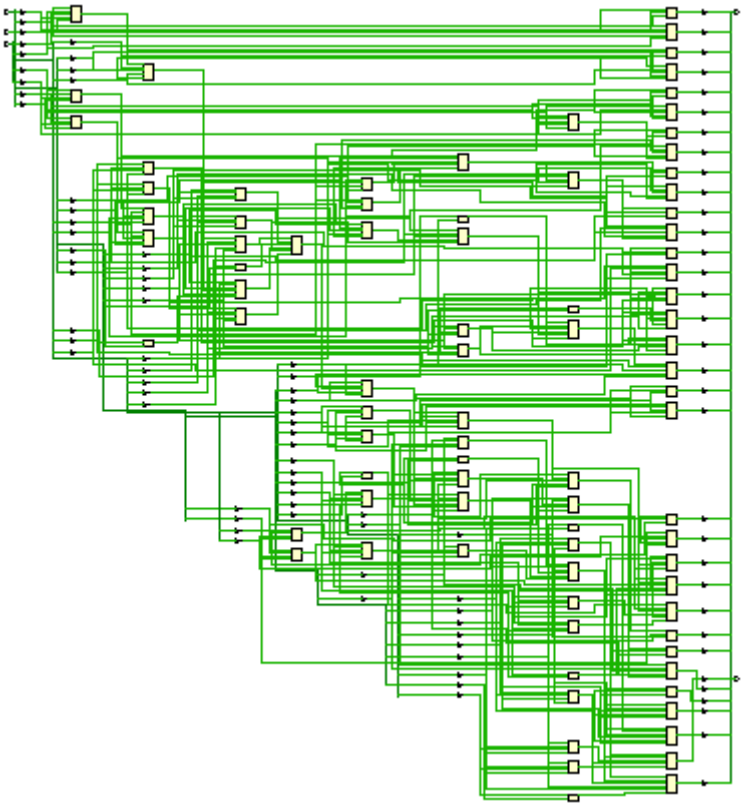
RTL Schematic:





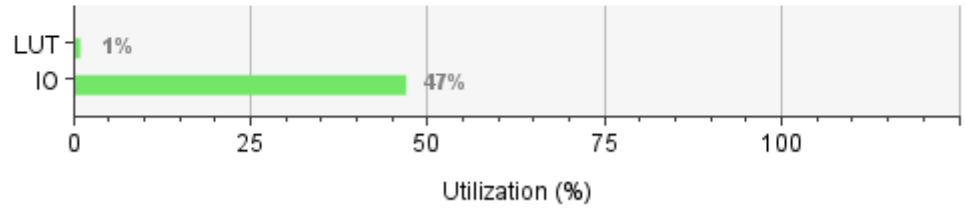
.

Technology Schematic:



Utilization Report:

Resource	Utilization	Available	Utilization %
LUT	59	32600	0.18
IO	98	210	46.67



59 LUT used in this design

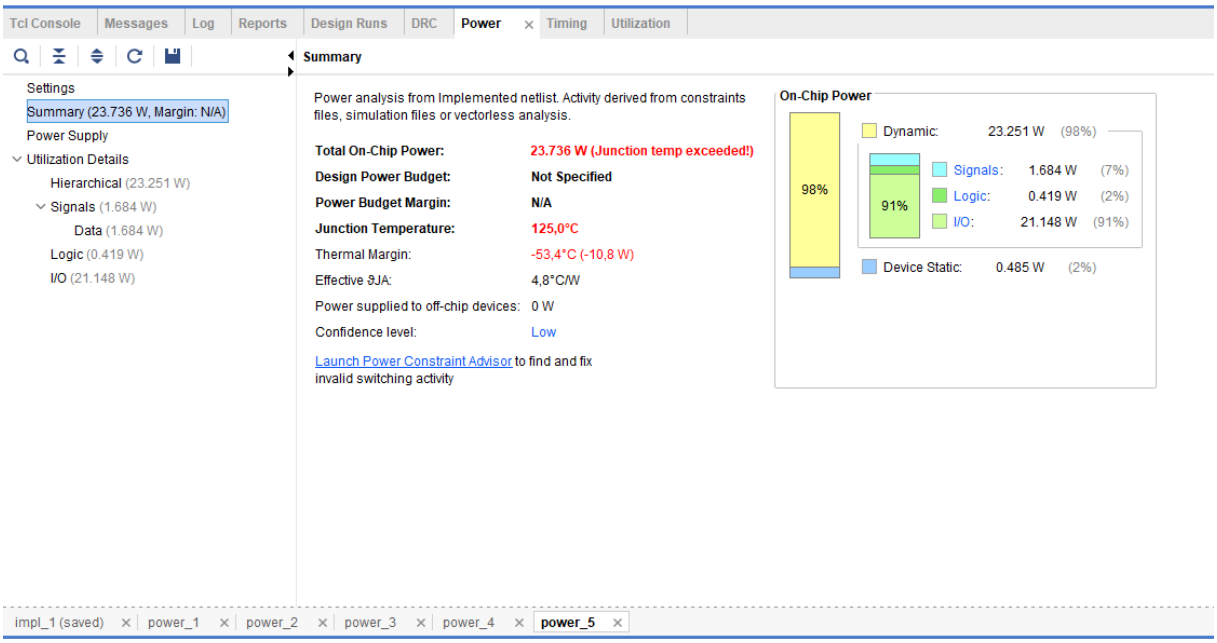
Timing Report:

Combinational Delays

From Port	To Port	Ma 1	Max Process Corner	Min Delay	Min Proces Corner
x[1]	sumout[28]	18.519	SLOW	6.894	FAST
x[0]	sumout[28]	18.366	SLOW	6.857	FAST
x[1]	sumout[29]	18.270	SLOW	6.793	FAST
x[5]	sumout[28]	18.175	SLOW	6.559	FAST
x[0]	sumout[29]	18.117	SLOW	6.756	FAST
x[1]	sumout[31]	18.001	SLOW	6.719	FAST
x[4]	sumout[28]	17.950	SLOW	6.474	FAST
x[5]	sumout[29]	17.926	SLOW	6.458	FAST
x[1]	sumout[30]	17.895	SLOW	6.675	FAST
x[7]	sumout[28]	17.888	SLOW	6.328	FAST
x[0]	sumout[31]	17.848	SLOW	6.682	FAST
x[0]	sumout[30]	17.743	SLOW	6.637	FAST
x[4]	sumout[29]	17.701	SLOW	6.373	FAST
x[5]	sumout[31]	17.657	SLOW	6.384	FAST
x[12]	sumout[28]	17.655	SLOW	6.229	FAST
x[7]	sumout[29]	17.639	SLOW	6.227	FAST
x[5]	sumout[30]	17.552	SLOW	6.340	FAST
cin	sumout[28]	17.524	SLOW	6.523	FAST
x[4]	sumout[31]	17.432	SLOW	6.299	FAST
x[1]	sumout[27]	17.418	SLOW	6.484	FAST

So maximum delay is 18.519 nanosecond. That means our period must be bigger than 18.519 nS. Lets assume it 18.52 nS, maximum clock frequency is 1/((18.52)*(10e-9)) almost 54 MHz.

Power Report:



When we use don't touch, power consumption is too much.

Utilization	Name	I/O Type	I/O Standard	Drive Strength	Input Pins	Output Pins	Bidir Pins	IO LOGIC SERDES	IO DELAY	IBUF LOW PWR	Input Term
21.148 W (89% of total)	project1_top										
20.403 W (86% of total)	sumout	HR	LVCMOS18	12.000	0	32	0	No	Off	No	NONE
0.485 W (2% of total)	cout	HR	LVCMOS18	12.000	0	1	0	No	Off	No	NONE
0.128 W (1% of total)	x	HR	LVCMOS18	N/A	32	0	0	No	Off	No	RTT_NONE
0.128 W (1% of total)	y	HR	LVCMOS18	N/A	32	0	0	No	Off	No	RTT_NONE
0.004 W (<1% of total)	cin	HR	LVCMOS18	N/A	1	0	0	No	Off	No	RTT_NONE

Average power consumption for Leaf Cells is 23.251/185 = 0.125 watt
Average power consumption for LUTs is 23,251/59 = 0.394 watt

Utilization	Name	Signals (W)	Data (W)	Logic (W)	I/O (W)
23.251 W (98% of total)	project1_top				
23.251 W (98% of total)	Leaf Cells (185)				

Source Codes:

Full Adder:

```
// ------------------------------
module HA (
input x,
input y,
output cout,
output sum
);
assign sum = ((~x)&&y) || (x&&(~y));
assign cout = x&&y;
endmodule
//------------------------------
module FA (
input x,
input y,
input cin,
output cout,
output sum);
wire sum1;
wire cout1;
wire sum2;
wire cout2;
HA HA1 (.x(x),.y(y),.sum(sum1),.cout(cout1));
HA HA2 (.x(sum1),.y(cin),.sum(sum2),.cout(cout2));
assign sum = sum2;
assign cout = cout2 || cout1;
endmodule
```

Level1 Mux:

```
module level1MUX(
input [3:0] D,
input S,
output reg selected_sum,
output reg selected_cout
    );


always @(S,D) begin
case(S)
    1'b0 :
    begin
    selected_sum<= D[0];
    selected_cout<= D[1];
    end
    1'b1:
    begin
    selected_sum <= D[2];
    selected_cout <= D[3];
    end
endcase
end
endmodule
```

Level2 Mux:

```
// ------------------------------
module level2MUX(
input [5:0] D,
input S,
output reg [1:0] selectedsum,
output reg selectedcout
    );
always @(S,D) begin
case(S)
    1'b1 :
    begin
    selectedsum<= D[1:0];
    selectedcout<= D[2];
    end
    1'b0:
    begin
    selectedsum <= D[4:3];
    selectedcout<= D[5];
    end
endcase
end
endmodule
```

Level3 Mux:

```
`timescale 1ns / 1ps
module level3MUX(
input [9:0] D,
input S,
output reg [3:0] selectedsum,
output reg selectedcout
    );
always @(S,D) begin
case(S)
    1'b1 :
    begin
    selectedsum<= D[3:0];
    selectedcout<= D[4];
    end
    1'b0:
    begin
    selectedsum <= D[8:5];
    selectedcout<= D[9];
    end
endcase
end
endmodule
```

8-bit Conditional Sum Adder:

```
module conditional_sum_adder(
    input [7:0] x,
    input [7:0] y,
    input cin,
    output [7:0] sumout,
    output cout_out
    );
wire s0,s1;
genvar i;
wire firstselect;
wire [1:0] muxselects_1 [2:0];
wire [1:0] mastersums_1 [2:0];
wire [3:0] muxinputs_1 [2:0];
wire [1:0] cselection_1 [7:1];
wire [1:0] sumwilladd_1 [7:1];
wire [1:0] cout [7:1];
wire [1:0] sum [7:1];
// LEVEL 1

FA fa0 (.x(x[0]),.y(y[0]),.cin(cin),.cout(c1),.sum(s0));
FA fa1_0 (.x(x[1]),.y(y[1]),.cin(0),.cout(cout[1][0]),.sum(sum[1][0]));
FA fa1_1 (.x(x[1]),.y(y[1]),.cin(1),.cout(cout[1][1]),.sum(sum[1][1]));
FA fa2_0
(.x(x[2]),.y(y[2]),.cin(0),.cout(cselection_1[2][0]),.sum(sumwilladd_1[2][0]
));
FA fa2_1
(.x(x[2]),.y(y[2]),.cin(1),.cout(cselection_1[2][1]),.sum(sumwilladd_1[2][1]
));
FA fa3_0 (.x(x[3]),.y(y[3]),.cin(0),.cout(cout[3][0]),.sum(sum[3][0]));
FA fa3_1 (.x(x[3]),.y(y[3]),.cin(1),.cout(cout[3][1]),.sum(sum[3][1]));
FA fa4_0
(.x(x[4]),.y(y[4]),.cin(0),.cout(cselection_1[4][0]),.sum(sumwilladd_1[4][0]
));
FA fa4_1
(.x(x[4]),.y(y[4]),.cin(1),.cout(cselection_1[4][1]),.sum(sumwilladd_1[4][1]
));
FA fa5_0 (.x(x[5]),.y(y[5]),.cin(0),.cout(cout[5][0]),.sum(sum[5][0]));
FA fa5_1 (.x(x[5]),.y(y[5]),.cin(1),.cout(cout[5][1]),.sum(sum[5][1]));
FA fa6_0
(.x(x[6]),.y(y[6]),.cin(0),.cout(cselection_1[6][0]),.sum(sumwilladd_1[6][0]
));
FA fa6_1
(.x(x[6]),.y(y[6]),.cin(1),.cout(cselection_1[6][1]),.sum(sumwilladd_1[6][1]
));
FA fa7_0 (.x(x[7]),.y(y[7]),.cin(0),.cout(cout[7][0]),.sum(sum[7][0]));
FA fa7_1 (.x(x[7]),.y(y[7]),.cin(1),.cout(cout[7][1]),.sum(sum[7][1]));

wire [3:0] mux0_in = {cout[1][1],sum[1][1],cout[1][0],sum[1][0]};
wire [3:0] mux1_in = {cout[3][1],sum[3][1],cout[3][0],sum[3][0]};
wire [3:0] mux2_in = {cout[3][1],sum[3][1],cout[3][0],sum[3][0]};
wire [3:0] mux3_in = {cout[5][1],sum[5][1],cout[5][0],sum[5][0]};
wire [3:0] mux4_in = {cout[5][1],sum[5][1],cout[5][0],sum[5][0]};
wire [3:0] mux5_in = {cout[7][1],sum[7][1],cout[7][0],sum[7][0]};
wire [3:0] mux6_in = {cout[7][1],sum[7][1],cout[7][0],sum[7][0]};

wire [1:0] selsum [7:1];
wire [1:0] selcout [7:1];
level1MUX M0 (.D(mux0_in),.S(c1),.selected_sum(s1),.selected_cout(c2));
```

```
level1MUX M1
(.D(mux1_in),.S(cselection_1[2][0]),.selected_sum(selsum[3][0]),.selected_co
ut(selcout[3][0]));
level1MUX M2
(.D(mux2_in),.S(cselection_1[2][1]),.selected_sum(selsum[3][1]),.selected_co
ut(selcout[3][1]));
level1MUX M3
(.D(mux3_in),.S(cselection_1[4][0]),.selected_sum(selsum[5][0]),.selected_co
ut(selcout[5][0]));
level1MUX M4
(.D(mux4_in),.S(cselection_1[4][1]),.selected_sum(selsum[5][1]),.selected_co
ut(selcout[5][1]));
level1MUX M5
(.D(mux5_in),.S(cselection_1[6][0]),.selected_sum(selsum[7][0]),.selected_co
ut(selcout[7][0]));
level1MUX M6
(.D(mux6_in),.S(cselection_1[6][1]),.selected_sum(selsum[7][1]),.selected_co
ut(selcout[7][1]));

wire [5:0] mux0_2_in =
{selcout[3][0],selsum[3][0],sumwilladd_1[2][0],selcout[3][1],selsum[3][1],su
mwilladd_1[2][1]};
wire [5:0] mux1_2_in =
{selcout[7][0],selsum[7][0],sumwilladd_1[6][0],selcout[7][1],selsum[7][1],su
mwilladd_1[6][1]};
wire [5:0] mux2_2_in =
{selcout[7][0],selsum[7][0],sumwilladd_1[6][0],selcout[7][1],selsum[7][1],su
mwilladd_1[6][1]};

wire [1:0] s2;
wire c3;
wire [1:0] mux2sumoutput0,mux2sumoutput1;
wire mux2cout0,mux2cout1;
level2MUX M20 (.D(mux0_2_in),.S(c2),.selectedsum(s2),.selectedcout(c3));
level2MUX M21_0
(.D(mux1_2_in),.S(selcout[5][0]),.selectedsum(mux2sumoutput0),.selectedcout(
mux2cout0));
level2MUX M21_1
(.D(mux2_2_in),.S(selcout[5][1]),.selectedsum(mux2sumoutput1),.selectedcout(
mux2cout1));



wire [9:0] mux3inputs =
{mux2cout0,mux2sumoutput0,selsum[5][0],sumwilladd_1[4][0],mux2cout1,mux2sumo
utput1,selsum[5][1],sumwilladd_1[4][1]};

wire clast;
wire [3:0] lastsum;
level3MUX LASTMUX
(.D(mux3inputs),.S(c3),.selectedsum(lastsum),.selectedcout(cout_out));


assign sumout = {lastsum,s2,s1,s0};
endmodule
```

32-bit Conditional Sum Adder TOP Module:

```
`timescale 1ns / 1ps

(* DONT_TOUCH = "TRUE" *)
module project1_top(
        input signed [31:0] x,
        input signed [31:0] y,
        input cin,
        output [31:0] sumout,
        output cout,
        output reg v
    );

wire [7:0] sum1,sum2,sum3,sum4;
conditional_sum_adder FIRST8
(.x(x[7:0]),.y(y[7:0]),.cin(cin),.sumout(sum1),.cout_out(co1));
conditional_sum_adder SECOND8
(.x(x[15:8]),.y(y[15:8]),.cin(co1),.sumout(sum2),.cout_out(co2));
conditional_sum_adder THIRD8
(.x(x[23:16]),.y(y[23:16]),.cin(co2),.sumout(sum3),.cout_out(co3));
conditional_sum_adder FOURTH8
(.x(x[31:24]),.y(y[31:24]),.cin(co3),.sumout(sum4),.cout_out(cout));

assign sumout = {sum4,sum3,sum2,sum1};

always @(*) begin
if( (x+y > 2147483646) || (x+y<-2147483647))
v<=1;
else
v<=0;
end

endmodule
```

Work package of this report:

Research	Muhammed Erkmen
Understanding	Muhammed Erkmen
Design	Muhammed Erkmen
Code	Muhammed Erkmen
Test	Muhammed Erkmen
Report	Muhammed Erkmen