EHB436E

DIGITAL SYSTEM DESIGN APPLICATIONS

Group: **M**ost **S**ignificant **B**it

Enes Şentürk 040170065

&

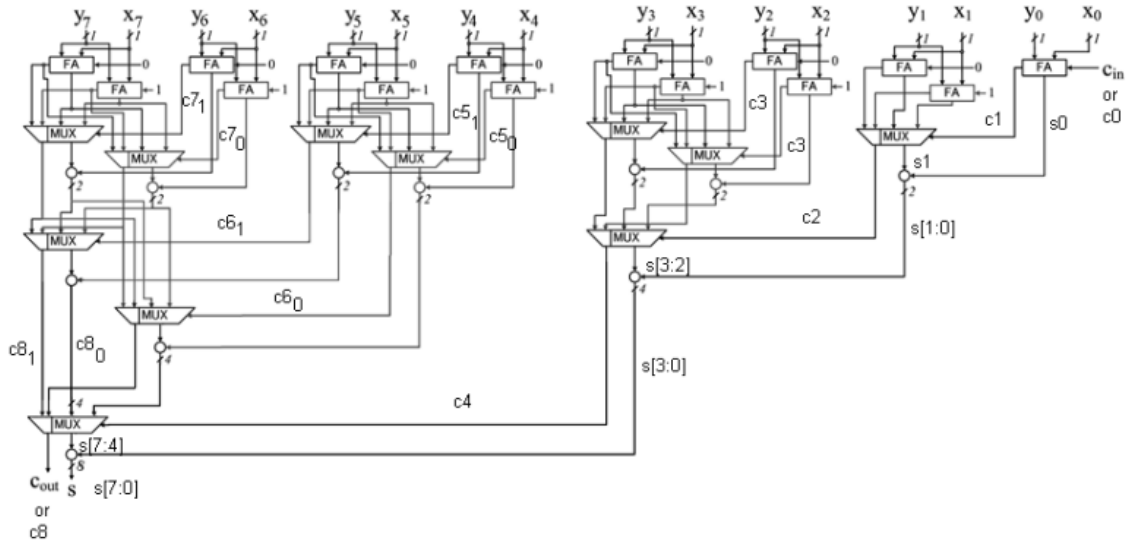
Muhammed Erkmen 040170049

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.

Our design steps:

We implemented 2 types of conditional sum adders. First one is using 8-bit conditional sum adder. Second we designed is 32-bit conditional sum adder.

First stage is calculating the every possible output available each bit pair, we did that by using full adders. In second step, we designed 32-bit conditional adder MUX structure.

We 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.

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

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 we made this part behavioral, not structural.

```
always @(*) begin
if( ((A< 0) && (B<0) && (C[31]==0)) || ((A>0) && (B>0) && (C[31]==1)))
v<=1;
else
v<=0;
```

While designing single module 32-bit conditional sum adder, we calculated every possible outputs from pair bit-0 to pair bit-31 for cin=1 and cin=0. Every stage is the same as the 4x8-bit design, but last mux. In last mux, we give cin as selection bit, so we get the result.

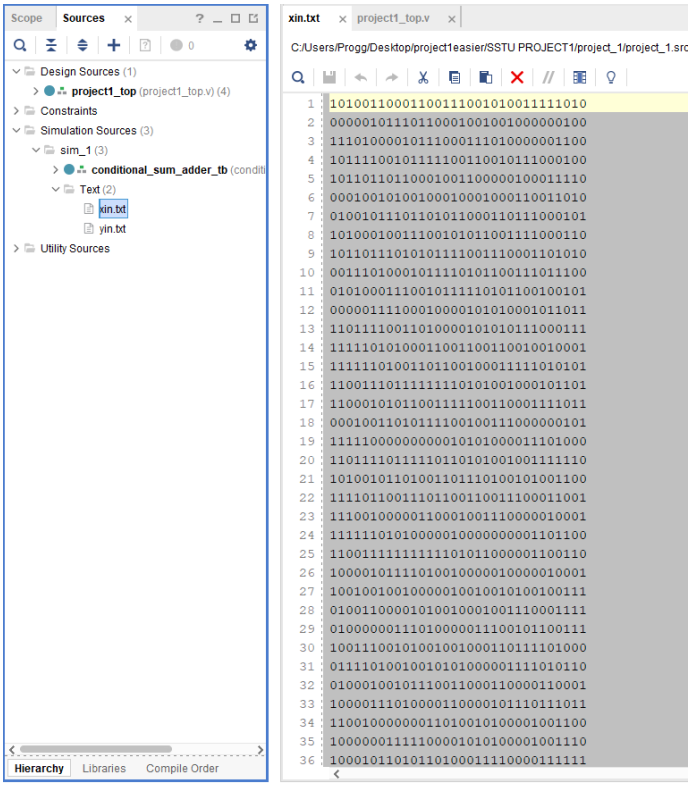
Preparing a testbench:

We designed a testbench with 100 random binary pairs by using $readmemb() function in testbench. We could get random binary numbers from: https://www.browserling.com/tools/random-bin. 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.



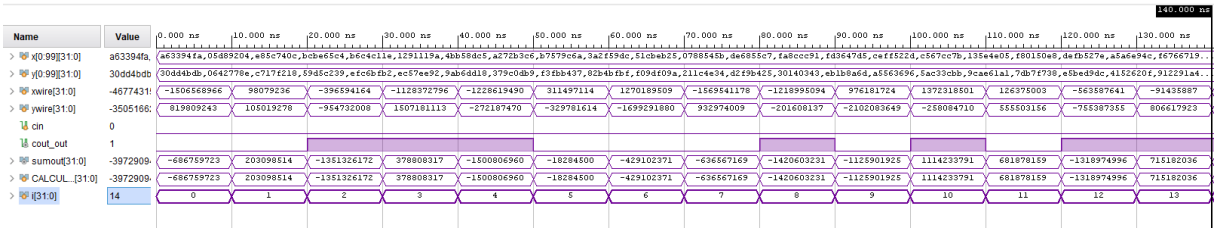
We read these datas and assigned values to [31:0] x [0:99] and [31:0] y [0:99] signed registers.

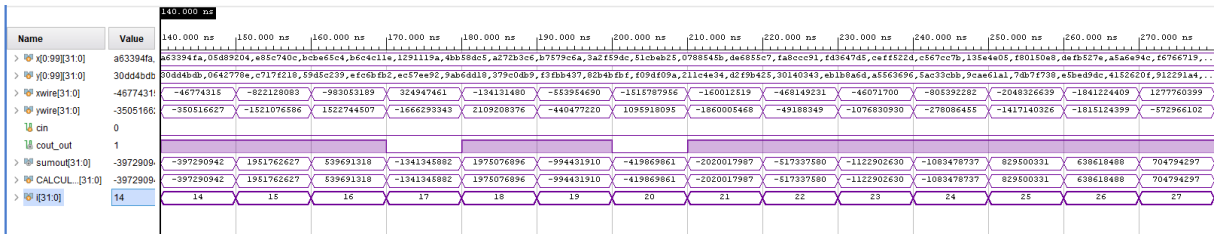
```
initial $readmemb("xin.txt",x);
initial $readmemb("yin.txt",y);
```

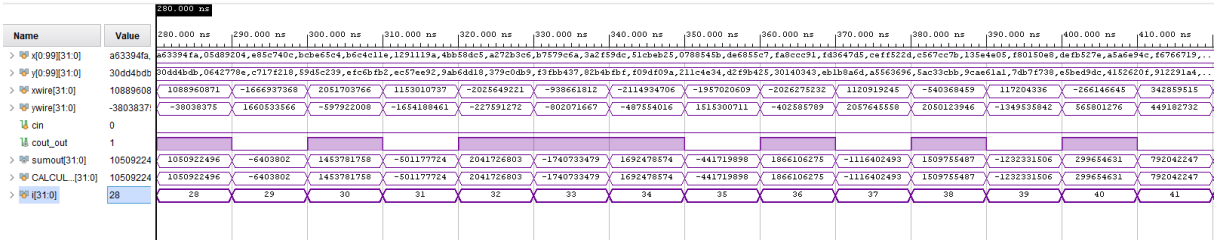
Just initial statements are left. we set a CALCULATOR register which gives directly x+y+cin value in testbench. In every for loop, x and y values goes to my top module and calculator calculates the true result. Then an if block controls that is the output of module (sumout) equal to CALCULATOR value.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

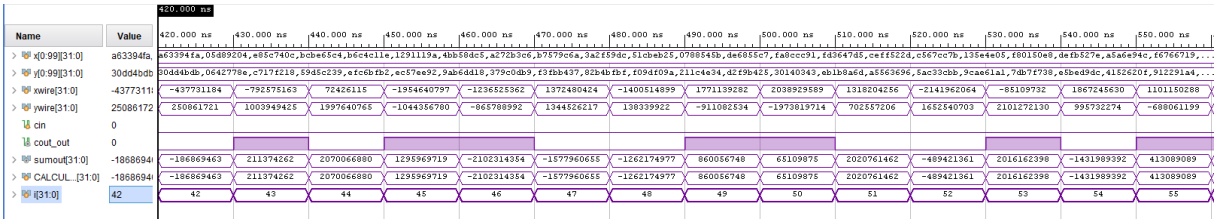
```
for (i=0;i<=99;i=i+1)
begin
 xwire = x[i];
 ywire = y[i];
 CALCULATOR = x[i]+y[i]+cin;
 #5;
 if(CALCULATOR == sumout)
 $display ("x:%d , y: %d, result: %d", xwire,ywire,sumout);
 else
 $display("operation is wrong");
 #5;
end
end
```

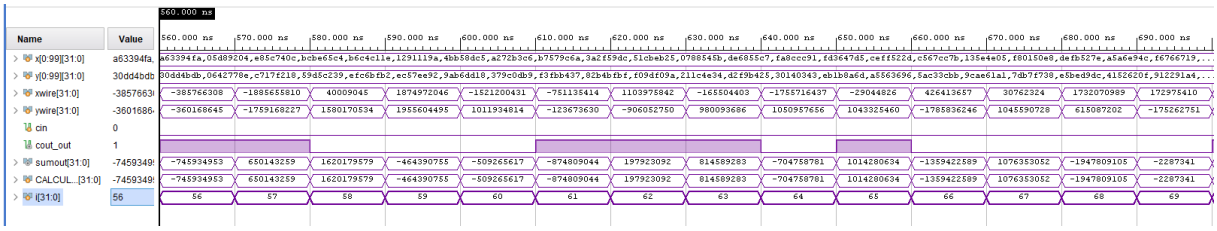
Waveform outputs of both conditional sum adders (4x8-bit and single module 32-bit):

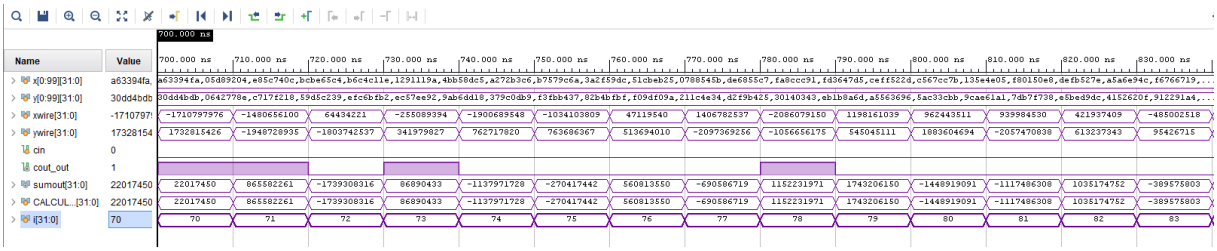


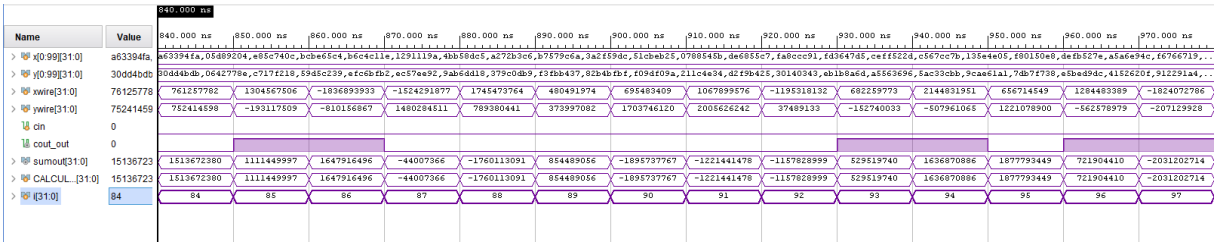


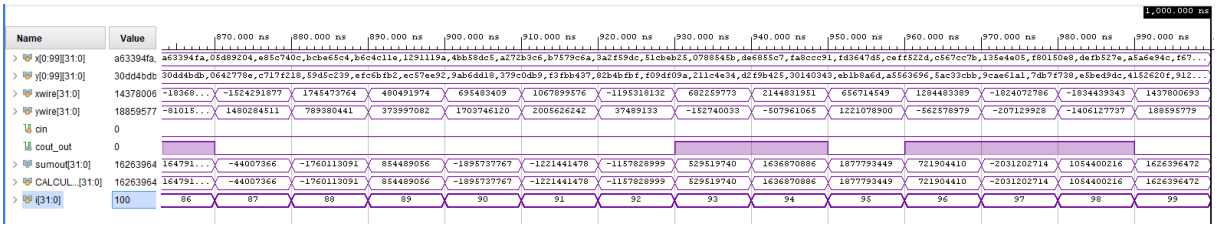

280.000 ns
Name
Value
x[0:99][31:0]
y[0:99][31:0]
xwire[31:0]
ywire[31:0]
cin
cout_out
sumout[31:0]
CALCUL...[31:0]
i[31:0]
28
29
30
31
32
33
34
35
36
37
38
39
40
41


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 to 4x8 bit conditional sum adder:

```
# run 1000ns
x="bin=10100110001100111001010011111010 dec=-1506568966",y="bin=00110000110111010100101111011011 dec=  819809243", result:"bin=11010111000100000111000001101010 dec= -686759723", overflow=0 status=TRUE
x="bin=00000011110110000100100000000000 dec=   98079236",y="bin=00000110010000010011111100001110 dec=  1050129278", result:"bin=00001100000101110000001001110010 dec=  203098514", overflow=0 status=TRUE
x="bin=11101000010111000111010000001100 dec= -396594164",y="bin=11000111000101111111001000011000 dec= -954732008", result:"bin=10101111011010001100110000100100 dec=-1351326172", overflow=0 status=TRUE
x="bin=10111100101111100110010111000100 dec=-1128372796",y="bin=01011001110101011100001000111001 dec= 1507181113", result:"bin=00010110100101000010011111111101 dec=  378808317", overflow=0 status=TRUE
x="bin=10110110110000100110000010001110 dec=-1228619490",y="bin=11101111110001101011111111110100 dec= -272187470", result:"bin=10100111000101100000011010000 dec=-1500806960", overflow=0 status=TRUE
x="bin=00010010100100010001000110011010 dec=  311497114",y="bin=11101100010101111111011010010010 dec= -329781614", result:"bin=11111111011101001000000000101100 dec=  -18284500", overflow=0 status=TRUE
x="bin=01001011101101011000110111000101 dec= 1270189509",y="bin=10011010101101101110110010011000 dec=-1699291880", result:"bin=11100110011011000111010101011011 dec= -429102371", overflow=0 status=TRUE
x="bin=10100010011100101011001111000110 dec=-1569541178",y="bin=00110111100111000000110110111001 dec=  932974009", result:"bin=11011010000001110110000010111111 dec= -636567169", overflow=0 status=TRUE
x="bin=10110111010101111101110001101010 dec=-1218995094",y="bin=11110011111111111011010000010111 dec= -201608137", result:"bin=10101011010100110101000010100001 dec=-1420603231", overflow=0 status=TRUE
x="bin=00111010000101111010110011011000 dec=  976181724",y="bin=10000010101101001011111101111111 dec=-2102083649", result:"bin=10111100111001000001100110011011 dec=-1125901925", overflow=0 status=TRUE
x="bin=01010001110101111101011001111011 dec= 1372318501",y="bin=11110000010111011111000100110111 dec= -258084710", result:"bin=01000010110000101100010010011111 dec= 1114233791", overflow=0 status=TRUE
x="bin=00000111100010000101010001011011 dec=  126375003",y="bin=00100001000111000100111000110100 dec=  555503156", result:"bin=00101000101001001010000101001111 dec=  681878159", overflow=0 status=TRUE
x="bin=11011110011010000101010111000111 dec= -563587641",y="bin=11010010111111001101101000100101 dec= -755387355", result:"bin=10110001011000100000010011101100 dec=-1318974996", overflow=0 status=TRUE
x="bin=11111010100011001100110010010001 dec=  -91435887",y="bin=00110000000101000000001101000011 dec=  806617923", result:"bin=00101010101000000110011111101000 dec=  715182036", overflow=0 status=TRUE
x="bin=11111101001101110010001111010101 dec=  -46774315",y="bin=11101011000111011100010011101101 dec= -350516627", result:"bin=11101000010100011101000100100010 dec= -397290942", overflow=0 status=TRUE
x="bin=11001110111111110101001000101101 dec= -822128083",y="bin=10100101010101100011101010010110 dec=-1521076586", result:"bin=01110100010101110000100011000011 dec= 1951762627", overflow=1 status=TRUE
x="bin=11000101011001111100111000111011 dec= -983053189",y="bin=01011010110000110011110001110011 dec= 1522744507", result:"bin=00100000001010110000010010110110 dec=  539691318", overflow=0 status=TRUE
x="bin=00010011010111100100111000000101 dec=  32494741",y="bin=10011100101011100011000110100001 dec=-1666293343", result:"bin=10110000000011001010111100100110 dec=-1341345882", overflow=0 status=TRUE
x="bin=11111000000000010101000011101000 dec= -134131480",y="bin=01111101101101111111101100111000 dec= 2109208376", result:"bin=01110101011100101001000000100000 dec= 1975076896", overflow=0 status=TRUE
x="bin=11011101111110110101001001111110 dec= -553954690",y="bin=11100101101111101101100111011000 dec= -440477220", result:"bin=11000100101110100010110001011010 dec= -994431910", overflow=0 status=TRUE
x="bin=10100101100111011100100111010101 dec=-1515787956",y="bin=01000001010010001000000000000111 dec= 1095918095", result:"bin=11100110111110010010110111011011 dec= -419869861", overflow=0 status=TRUE
x="bin=11110110011101100110011100011001 dec= -160012519",y="bin=10010001001000101001000110100100 dec=-1860005468", result:"bin=10000111100110001111100010111101 dec=-2020017987", overflow=0 status=TRUE
x="bin=11100100000110001001110000010001 dec= -468149231",y="bin=11111010000100010111000000000111 dec=  -49188349", result:"bin=11100001001010100000011110000100 dec= -517337580", overflow=0 status=TRUE
x="bin=11111101010000010000000001101000 dec=  -46071700",y="bin=10111111110100001101110100101110 dec=-1076830930", result:"bin=10111010000100011101110110010010 dec=-1122902630", overflow=0 status=TRUE
x="bin=11001111111111110101100001100110 dec= -805392282",y="bin=11101111010110010111110001001001 dec= -278086455", result:"bin=10111111010101101101100101011111 dec=-1083478737", overflow=0 status=TRUE
x="bin=10000101111010010000010000010001 dec=-2048326639",y="bin=10101011100010000010011110010010 dec=-1417140326", result:"bin=00110001011100010010110110101011 dec=  829500331", overflow=1 status=TRUE
x="bin=10010010010000001001001010000111 dec=-1841224409",y="bin=10010011100111011011001001010001 dec=-1815124399", result:"bin=00100110000100000100010110111000 dec=  638618488", overflow=1 status=TRUE
x="bin=01001100001010010001001110001111 dec= 1277760399",y="bin=10111101110110010011101001010010 dec= -572966102", result:"bin=00101010000000100100111010111001 dec=  704794297", overflow=0 status=TRUE
x="bin=01000000111010000011100101100111 dec= 1088960871",y="bin=11111110101111011100101001001001 dec=  -38038375", result:"bin=00111110100100011110011100000000 dec= 1050922496", overflow=0 status=TRUE
x="bin=10011100101001001000110111101000 dec=-1666937368",y="bin=01100010111100110111001100111110 dec= 1660533566", result:"bin=11111111100111000100100100100110 dec=  -6403802", overflow=0 status=TRUE
x="bin=01111010010010100100001111010110 dec= 2051703766",y="bin=11011100011100100110111100101000 dec= -597922008", result:"bin=01010110100100110111100101111110 dec= 1453781758", overflow=0 status=TRUE
x="bin=01000100101110011000110000110001 dec= 1153010737",y="bin=10011101011001110001011001010011 dec=-1654188461", result:"bin=11100010001000001010001010000100 dec= -501177724", overflow=0 status=TRUE
x="bin=10000111010000011000101111011011 dec=-2025649221",y="bin=11110010011011100111011110010000 dec= -227591272", result:"bin=01111001101100100010011101010011 dec= 2041726803", overflow=1 status=TRUE
x="bin=11001000000011010010100001001100 dec= -938661812",y="bin=11010000001100010101101110001101 dec= -802071667", result:"bin=10011000001111101000001111011001 dec=-1740733479", overflow=0 status=TRUE
x="bin=10000001111100010000010000010001 dec=-2114934706",y="bin=11100010111100001000000100000000 dec= -487554016", result:"bin=01100001110000010010111011110011 dec= 1692478574", overflow=1 status=TRUE
x="bin=10001011010110100001111000011111 dec=-1957020609",y="bin=01011010010000110100011101100111 dec= 1515300711", result:"bin=11100101101010111110001110100110 dec= -441719898", overflow=0 status=TRUE
x="bin=10000111001100101110111111001000 dec=-2026275232",y="bin=11101000000000010001101111010000 dec= -402585789", result:"bin=01101110001101000100100101100011 dec= 1866106275", overflow=1 status=TRUE
x="bin=01000010110011111101111011001101 dec= 1120919245",y="bin=01111010101001010010110111110110 dec= 2057645558", result:"bin=10111101011101010000110011000011 dec=-1116402493", overflow=1 status=TRUE
x="bin=11011111100101010100001110110101 dec= -540368459",y="bin=01111010001100100110100100010001 dec= 2050123946", result:"bin=01011001111111010000100101011111 dec= 1509755487", overflow=0 status=TRUE
x="bin=00000110111111100011001010111000 dec=  117204336",y="bin=10101111100011111101101111001110 dec=-1349535842", result:"bin=10110110100011000001110100001110 dec=-1232331506", overflow=0 status=TRUE
x="bin=11110000001000010011101100101011 dec= -266146645",y="bin=00100000110111001110111000100111 dec=  565801276", result:"bin=00010001110111100010111011111100 dec=  299654631", overflow=0 status=TRUE
x="bin=00010100011011111001111011111011 dec=  342859515",y="bin=00011010110001011111110000001100 dec=  449182732", result:"bin=00101111001101011001101100000111 dec=  792042247", overflow=0 status=TRUE
x="bin=11100101111010000100000010010000 dec= -437731184",y="bin=00001110111001111011100010011001 dec=  250861721", result:"bin=11110100110111100100110010101001 dec= -186869463", overflow=0 status=TRUE
```

```
x="bin=11100101111010001100000010010000 dec= -437731184",y="bin=00001110111001111011100010011001 dec=  250861721", result:"bin=11110100110111100100110010101001 dec= -186869463", overflow=0 status=TRUE
x="bin=11010000011000010010000101000101 dec= -792575163",y="bin=00111011110101110000110101110001 dec= 1003949425", result:"bin=00001100100110010101000010110110 dec=  211374262", overflow=0 status=TRUE
x="bin=00000100010100010010000100000001 dec=   72426115",y="bin=01110111000010111011011010001111 dec= 1997640765", result:"bin=01111010110000101101101110000000 dec= 2070066880", overflow=0 status=TRUE
x="bin=10001011011111101000110001100011 dec=-1954640797",y="bin=11000001110000000110000101010100 dec=-1044356780", result:"bin=01001101001111110111011011010111 dec= 1295969719", overflow=1 status=TRUE
x="bin=10110110010011000000111011001111 dec=-1236523562",y="bin=11001100011001010001011110000000 dec= -865788992", result:"bin=10000010101000010011101010010110 dec=-2102314354", overflow=0 status=TRUE
x="bin=01010001110011100110001110101000 dec= 1372480424",y="bin=01010000001000111010111010001001 dec= 1344526217", result:"bin=10100001111100100011101110010001 dec=-1577960655", overflow=1 status=TRUE
x="bin=10101100100001011101101101101101 dec=-1400514899",y="bin=00001000001111101111001001010010 dec=  138339922", result:"bin=10110100110001001011110011111111 dec=-1262174977", overflow=0 status=TRUE
x="bin=01010011001000101110000110101010 dec= 1771139282",y="bin=11001001101110001111101111011010 dec= -911082534", result:"bin=00110011010000110110110010101100 dec=  860056748", overflow=0 status=TRUE
x="bin=01111001100011110010001101101111 dec= 2038929589",y="bin=10001010010110011111001010111110 dec=-1973819714", result:"bin=00000011110000100101111111110011 dec=  65109875", overflow=0 status=TRUE
x="bin=01001110100100100011001101100000 dec= 1318204256",y="bin=00101001111000000010110000010110 dec=  702557206", result:"bin=01111000011100100101111101110110 dec= 2020761462", overflow=0 status=TRUE
x="bin=10000000010101000100000001100010 dec=-2141962064",y="bin=01100010111111111000010000111111 dec= 1652540703", result:"bin=11110001010100000000101111001111 dec= -489421361", overflow=0 status=TRUE
x="bin=11111010111010101010100000111000 dec=  -85109732",y="bin=01111101001111110111111001000010 dec= 2101272130", result:"bin=01111000000101100001100100101110 dec= 2016162398", overflow=0 status=TRUE
x="bin=01101111001010111110100001111110 dec= 1867245630",y="bin=00111101101011001101010110010010 dec=  995732274", result:"bin=10101010100101100100100101110000 dec=-1431989392", overflow=1 status=TRUE
x="bin=01000001101000100011100001010000 dec= 1101150288",y="bin=11010101111111010000100111110001 dec= -688061199", result:"bin=00011000100111110011111010000001 dec=  413089089", overflow=0 status=TRUE
x="bin=11101001000000001101011000101100 dec= -385766308",y="bin=11101010100000000000011001101011 dec= -360168645", result:"bin=11010011100000100111011111100101 dec= -745934953", overflow=0 status=TRUE
x="bin=10001111100110110010110011111110 dec=-1885655810",y="bin=10010111000100101011001100111101 dec=-1759168227", result:"bin=00100110110000000011001100011101 dec=  650143259", overflow=1 status=TRUE
x="bin=00000001001100010011111010101001 dec=   40009045",y="bin=01011110000101111011111010100110 dec= 1580170534", result:"bin=01100000010000011111101100111011 dec= 1620179579", overflow=0 status=TRUE
x="bin=01101111110000001110011011000110 dec= 1874972046",y="bin=01110100100100000010000000111111 dec= 1955604495", result:"bin=11100100010100011111010110011101 dec= -464390755", overflow=1 status=TRUE
x="bin=10100101010100010100101110100001 dec=-1521200431",y="bin=00111100010100001110011001011110 dec= 1011934814", result:"bin=11100001101001010011100100101111 dec= -509265617", overflow=0 status=TRUE
x="bin=11010011001110100101010101001010 dec= -751135414",y="bin=11111000101000001110001101101110 dec= -123673630", result:"bin=11001011111010111011111001001100 dec= -874809044", overflow=0 status=TRUE
x="bin=01000001110011010101010110100010 dec= 1103975842",y="bin=11001001111111101011101110110010 dec= -906052750", result:"bin=00001011110011000001000100010100 dec=  197923092", overflow=0 status=TRUE
x="bin=11110110001000100110011010001101 dec= -165504403",y="bin=00111010010110110000010110111110 dec=  980093686", result:"bin=00110000100011011010010110001111 dec=  814589283", overflow=0 status=TRUE
x="bin=10010111010110011110010010101011 dec=-1755716437",y="bin=00111101010010000101011101011000 dec= 1050957656", result:"bin=11010101111111100011110000000011 dec= -704758781", overflow=0 status=TRUE
x="bin=11111110010001001110011111010110 dec=  -29044826",y="bin=00111110001011111111000010000100 dec= 1043325460", result:"bin=01011100011101001011000110111010 dec= 1014280634", overflow=0 status=TRUE
x="bin=00011001011010100001110010111001 dec=  426413657",y="bin=10010101100011100100110100101010 dec=-1785836246", result:"bin=10101110111110001101011110000011 dec=-1359422589", overflow=0 status=TRUE
x="bin=00000001110101010110010101010100 dec=   30762324",y="bin=00111110010100100111001011001000 dec= 1045590728", result:"bin=01000000001001111101100000011100 dec= 1076535052", overflow=0 status=TRUE
x="bin=01100111001111010100111001001101 dec= 1732070989",y="bin=00100100101010010111110001100010 dec=  615087202", result:"bin=10001011111001101100101010101111 dec=-1947809105", overflow=1 status=TRUE
x="bin=00001010010011111010010100110010 dec=  172975410",y="bin=11110101100011011011001111100001 dec= -175262751", result:"bin=11111111111011010001100100010111 dec=  -2287341", overflow=0 status=TRUE
x="bin=10011010000001110100101101101000 dec=-1710797976",y="bin=01100111010010001010101001000010 dec= 1732815426", result:"bin=00000001010011111111101011010100 dec=  22017450", overflow=0 status=TRUE
x="bin=10100111101111101111111011000110 dec=-1480656100",y="bin=10001011111010001100001100001001 dec=-1948728935", result:"bin=00110011100101111011111001010101 dec=  865582261", overflow=1 status=TRUE
x="bin=00000011110101110011000000101101 dec=   64434221",y="bin=10010100011111010001001010110111 dec=-1803742537", result:"bin=10011000010101000010000101100100 dec=-1739308316", overflow=0 status=TRUE
x="bin=11110000110010111010010000111110 dec= -255089394",y="bin=00010100011000100001100101010111 dec=  341979827", result:"bin=00000101001011011101011111000001 dec=  86890433", overflow=0 status=TRUE
x="bin=10001110110110111000111011110100 dec=-1900689548",y="bin=00101101011101100010011001111100 dec=  762717820", result:"bin=10111100001010111101101111100000 dec=-1137971728", overflow=0 status=TRUE
x="bin=11000010010111001101001111111111 dec=-1034103809",y="bin=00101101100000100111010111011111 dec=  763686367", result:"bin=11101111111000011100000111011110 dec= -270417442", overflow=0 status=TRUE
x="bin=00000010110011101111110010110100 dec=   47119540",y="bin=00011110100111100101100100111010 dec=  513694010", result:"bin=00100001011011010101010111101100 dec=  560813550", overflow=0 status=TRUE
x="bin=01010011101100111001110001001001 dec= 1406782537",y="bin=10000010111111001010111110101111 dec=-2097369256", result:"bin=11010110101101011001111011110100 dec= -690586719", overflow=0 status=TRUE
x="bin=10000011101010001111010101010010 dec=-2086079150",y="bin=11000001000001001011010011010001 dec=-1056656175", result:"bin=01000100101011011010101000100011 dec= 1152231971", overflow=1 status=TRUE
x="bin=01000111010101001111010001111111 dec= 1198161039",y="bin=00100000011111001011010001110111 dec=  545045111", result:"bin=01100111111001110011011110000110 dec= 1743206150", overflow=0 status=TRUE
x="bin=00111001010111011011100011110111 dec=  962443511",y="bin=01110000010001011000011011010110 dec= 1883604694", result:"bin=10101001101000110011111111001101 dec=-1448919091", overflow=1 status=TRUE
x="bin=00011000000011100001101001010010 dec=  939984530",y="bin=10000101010111010111111001001010 dec=-2057470838", result:"bin=10111101010110010000000110001100 dec=-1117486308", overflow=0 status=TRUE
x="bin=00011001001001100100000010000001 dec=  421937409",y="bin=00100100100011010100001001011111 dec=  613237343", result:"bin=00111101101100111000001101100000 dec= 1035174752", overflow=0 status=TRUE
x="bin=11100011000101111001011110011010 dec= -485002518",y="bin=00000101101100000000011001011011 dec=   95426715", result:"bin=11101000110001111000101110000101 dec= -389575803", overflow=0 status=TRUE
x="bin=01011010101111111101111100110110 dec=  761257782",y="bin=00101100110110001110111110000110 dec=  752414598", result:"bin=01011010001110001100111010111100 dec= 1513672380", overflow=0 status=TRUE
x="bin=01001101100000000011101101010010 dec= 1304567506",y="bin=11110100011110101000010110111011 dec= -193117509", result:"bin=01000010001111101011000011000101 dec= 1111449997", overflow=0 status=TRUE
x="bin=10010010100000011001110010000110 dec=-1836893933",y="bin=11001111101101011111110010111101 dec= -810156867", result:"bin=01100010001111001001101011110000 dec= 1647916496", overflow=1 status=TRUE
x="bin=10100101001001010010011011101011 dec=-1524291877",y="bin=01011000001110101011001010111111 dec= 1480284511", result:"bin=11111101011000001000000001110100 dec=  -44007366", overflow=0 status=TRUE
```

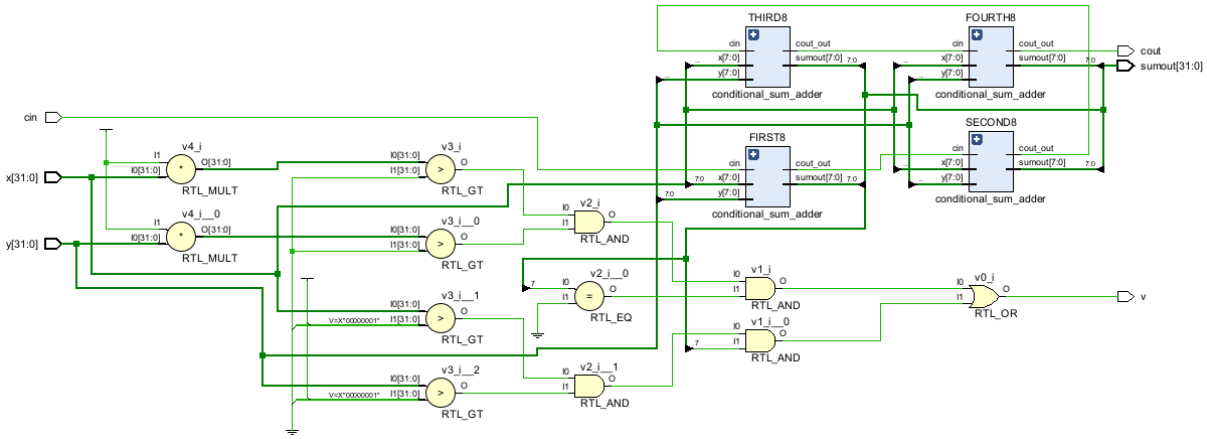
```
x="bin=01101000000001100111010000111000 dec= 1745473764",y="bin=00101111000011001111111010101001 dec=  789380441", result:"bin=10010111000101101100111000111101 dec=-1760173091", overflow=1 status=TRUE
x="bin=00011100101000111011100111000110 dec=  480491974",y="bin=00010110010010101011111000011010 dec=  373997082", result:"bin=00110010111011100111011111100000 dec=  854489056", overflow=0 status=TRUE
x="bin=00101001011100010000011110000001 dec=  695483409",y="bin=01100101100011010001101001001000 dec= 1703746120", result:"bin=10001111000000010110001101011001 dec=-1895737767", overflow=1 status=TRUE
x="bin=00111111110100110110101010111000 dec= 1067899576",y="bin=01110111100010110110110100000010 dec= 2005626242", result:"bin=10110111001100100100000011101000 dec=-1221441478", overflow=1 status=TRUE
x="bin=10111000110000001100010000011000 dec=-1195318132",y="bin=00000010001111000000010011101101 dec=   37489133", result:"bin=10111010111111001111011111001101 dec=-1157828999", overflow=0 status=TRUE
x="bin=00101000101010100111010100111101 dec=  682259773",y="bin=11110110111001010111110011111 dec= -152740033", result:"bin=00011111100011111010100111111000 dec=  529519740", overflow=0 status=TRUE
x="bin=01111111110101111000100111001111 dec= 2144831951",y="bin=11100001101110010000100010111 dec= -507961065", result:"bin=01100001100000010101011100110000 dec= 1636870886", overflow=0 status=TRUE
x="bin=00100111001001001010110011010001 dec=  656714549",y="bin=01001000110100000010111101110100 dec= 1221078900", result:"bin=01101111111101100110101010010001 dec= 1877793449", overflow=0 status=TRUE
x="bin=01001100100011111010100100111101 dec= 1284483389",y="bin=11011110011011011011111011011001 dec= -562578979", result:"bin=00101011000001110001100010011010 dec=  721904410", overflow=0 status=TRUE
x="bin=10010011010001101110111011010110 dec=-1824072786",y="bin=11110011010011101110010101110000 dec= -207129928", result:"bin=10000110111011100100111001100110 dec=-2031202714", overflow=0 status=TRUE
x="bin=10010010110100010100101010001001 dec=-1834439343",y="bin=10101100001100000011000110000111 dec=-1406127737", result:"bin=00111110110110000101110111011000 dec= 1054400216", overflow=1 status=TRUE
x="bin=01010101101100110001100011110101 dec= 1437800693",y="bin=00001011001111010111111001000011 dec=  188595779", result:"bin=01100000111000011010111001110000 dec= 1626396472", overflow=0 status=TRUE
```

We writed these outputs to a txt file too. As it can seem, our outflow mechanism works correct.

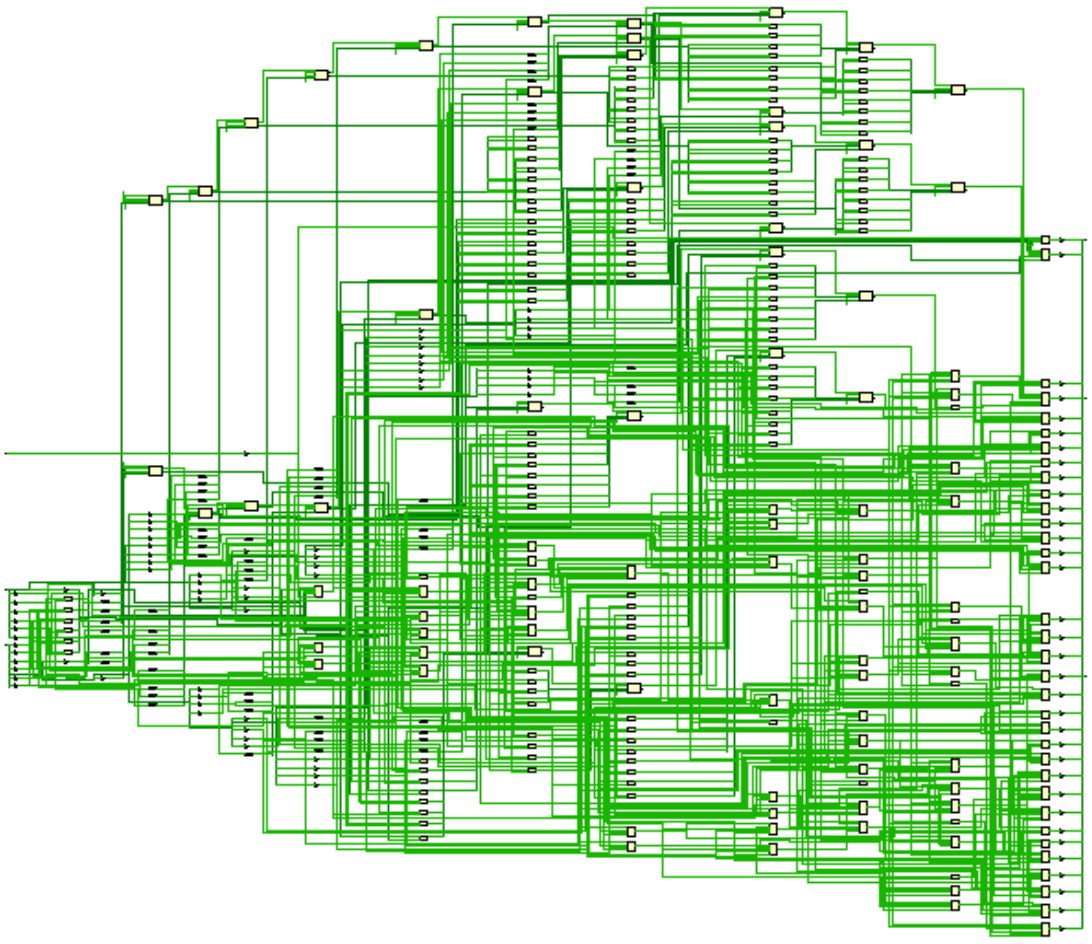
4x8-bit conditional sum adder Implementation Reports:

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

RTL Schematic:



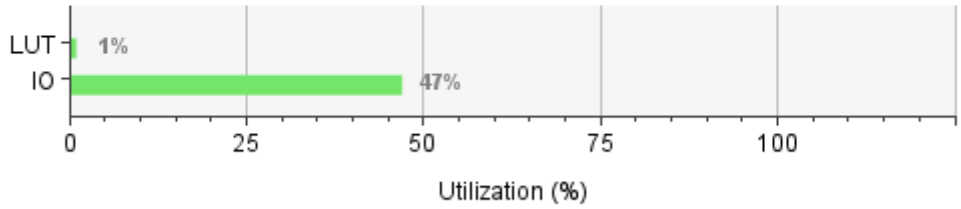
Technology Schematic of 4x8 bit conditional sum adder:



Utilization Report of 4x8 bit conditional sum adder:

Summary

Resource	Utilization	Available	Utilization %
LUT	190	32600	0.58
IO	99	210	47.14



190 LUT used in this design

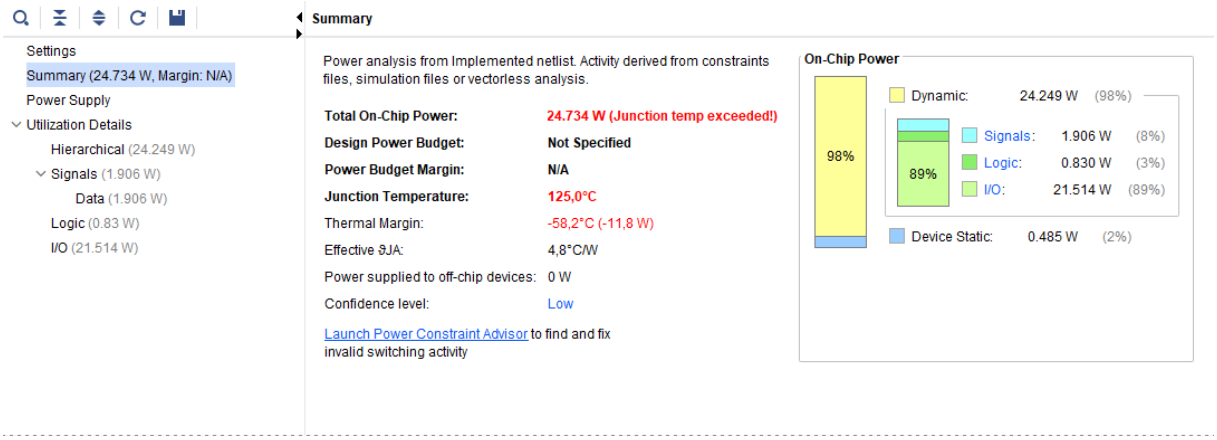
Timing Report of 4x8 bit conditional sum adder:

Combinational Delays

From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
x[5]	sumout[28	16.874	SLOW	5.992	FAST
x[5]	sumout[25	16.854	SLOW	6.040	FAST
x[1]	sumout[28	16.825	SLOW	6.204	FAST
x[5]	sumout[30	16.811	SLOW	5.984	FAST
x[1]	sumout[25	16.805	SLOW	6.252	FAST
x[1]	sumout[30	16.763	SLOW	6.196	FAST
x[5]	sumout[27	16.737	SLOW	5.954	FAST
x[5]	sumout[29	16.704	SLOW	6.006	FAST
x[0]	sumout[28	16.696	SLOW	6.101	FAST
x[1]	sumout[27	16.688	SLOW	6.166	FAST
x[0]	sumout[25	16.676	SLOW	6.150	FAST
x[1]	sumout[29	16.655	SLOW	6.218	FAST
x[5]	sumout[26	16.645	SLOW	5.895	FAST
x[0]	sumout[30	16.634	SLOW	6.093	FAST
x[1]	sumout[26	16.596	SLOW	6.107	FAST
x[0]	sumout[27	16.559	SLOW	6.063	FAST
x[5]	sumout[31	16.547	SLOW	5.940	FAST
x[0]	sumout[29	16.526	SLOW	6.115	FAST

So maximum delay is 16.874 nanosecond. That means our period must be bigger than 16.874 nS. Maximum clock is 1/((16.873)*(10e-9)) almost **59.26 MHz.**

Power Report of 4x8 bit conditional sum adder:



So average power on-chip is 24.734 Watt, because we used (* DONT_TOUCH *) and that makes the algorithm do the power analysis with vectorless estimation power analysis and result is the average power.

Source Codes of 4x8-bit conditional sum adder:

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
```

4x8-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
```

4x8-bit Conditional Sum Adder testbench:

```
`timescale 1ns / 1ps


module conditional_sum_adder_tb;

reg signed [31:0] A [0:99];
reg signed [31:0] B [0:99];
reg cin;
wire cout_out;
wire signed [31:0] sumout;
reg signed [31:0] CALCULATOR;
reg signed [31:0] xwire;
reg signed [31:0] ywire;
wire v;
project1_top
TOP(.x(xwire),.y(ywire),.cin(cin),.cout(cout_out),.v(v),.sumout(sumout));

integer n;
initial $readmemb("xin.txt",A);
initial $readmemb("yin.txt",B);
initial n = $fopen("outfile.txt","w");
```

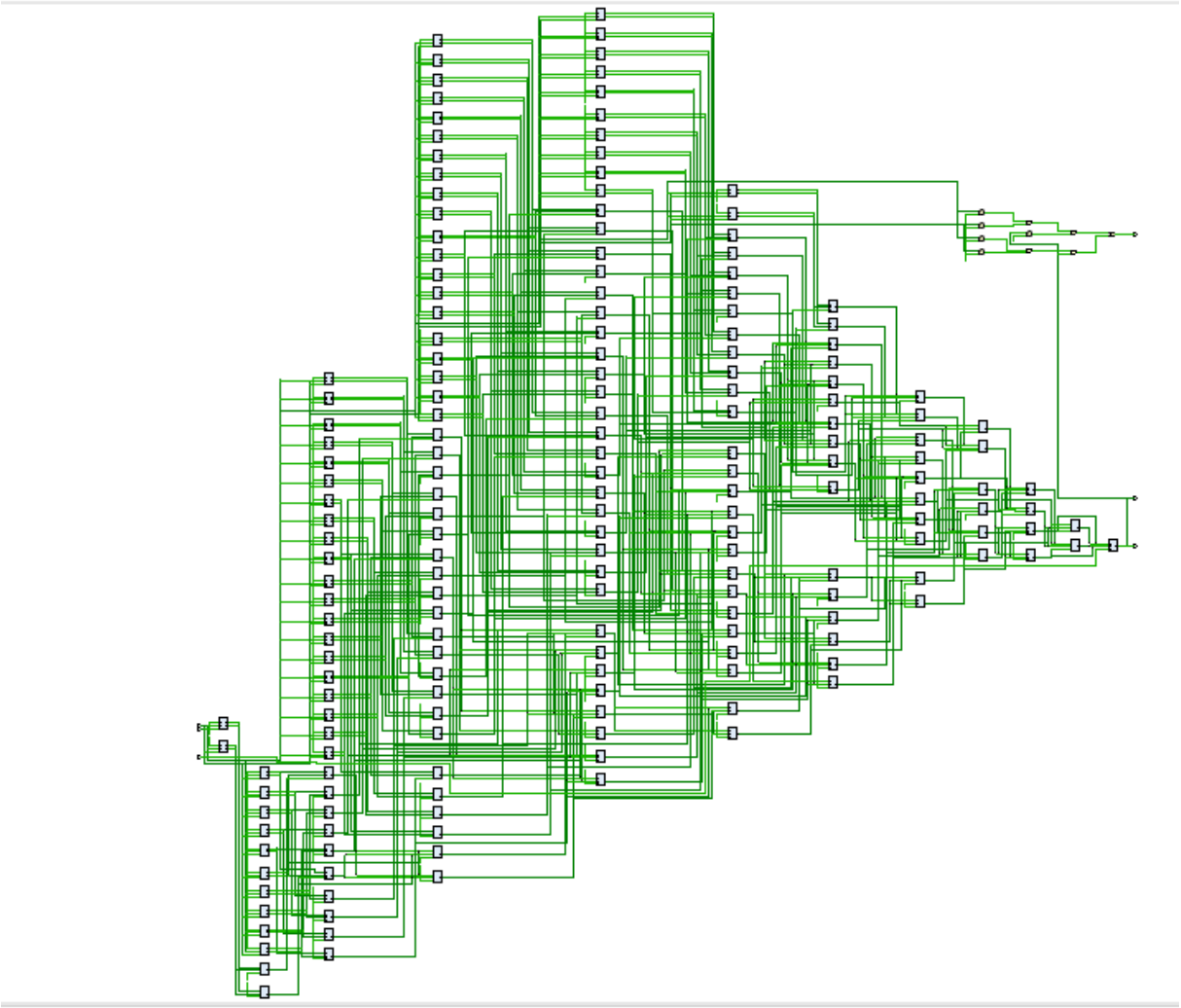
```
integer i;

initial begin
cin = 0;
for (i=0;i<=99;i=i+1)
begin
 xwire = A[i];
 ywire = B[i];
 CALCULATOR = A[i]+B[i]+cin;
 #5;
 if(CALCULATOR == sumout)
 begin
 $fdisplay (n,"x='bin=%b dec=%d',y='bin=%b dec=%d', result:'bin=%b
dec=%d', overflow=%d status=TRUE",
xwire,xwire,ywire,ywire,sumout,sumout,v);
 $display ("x='bin=%b dec=%d',y='bin=%b dec=%d', result:'bin=%b dec=%d',
overflow=%d status=TRUE", xwire,xwire,ywire,ywire,sumout,sumout,v);
 end
 else
 begin
 $fdisplay(n,"operation is wrong");
 $display("operation is wrong");
 end
 #5;
end
$finish;
end
endmodule
```

Testbenches are not excatly same with the 32-bit conditional adder because of the module names and variable names.

Implementation of single module 32-bit conditional sum adder reports:

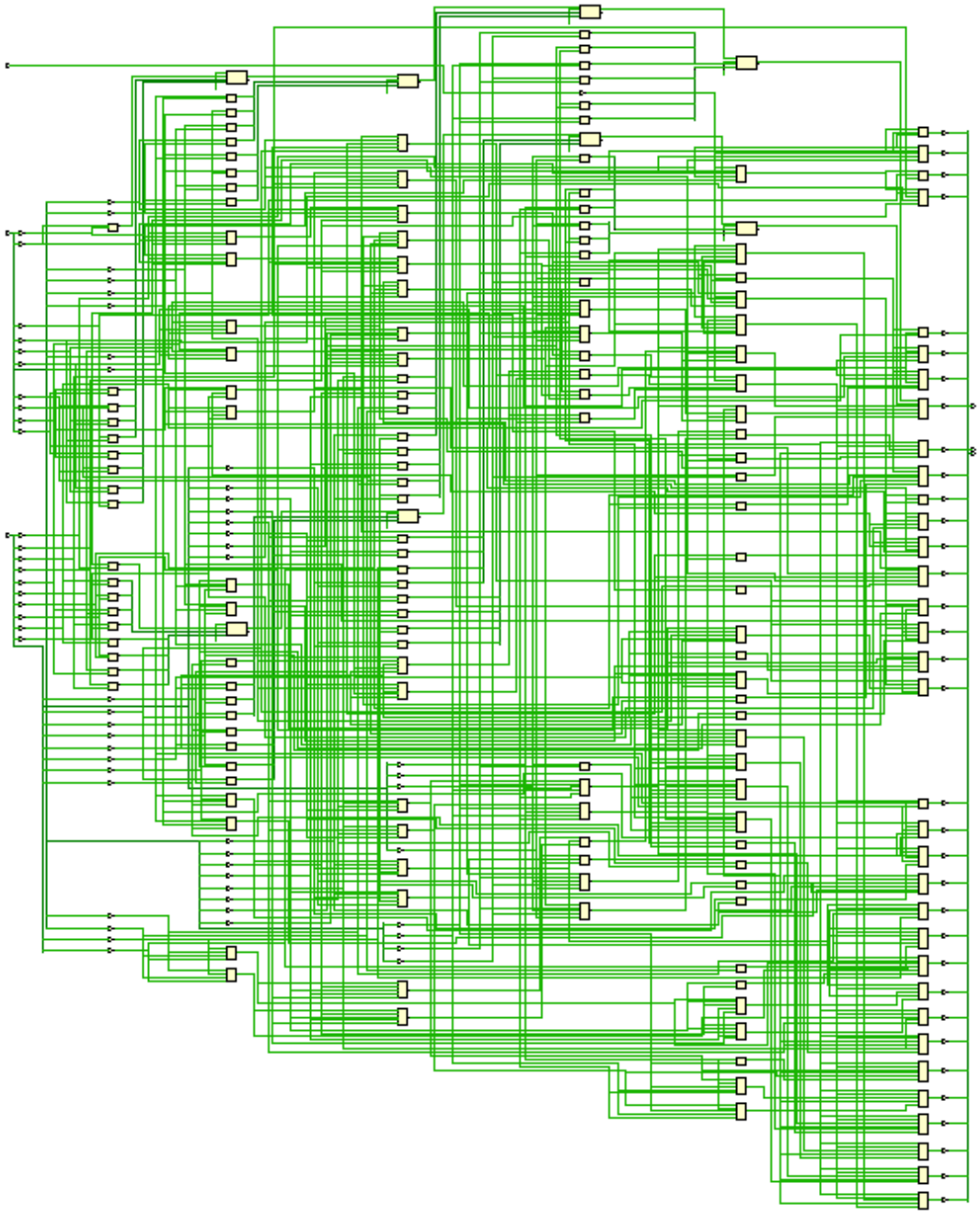
So we put top of our module (* DONT_TOUCH = "TRUE" *) statement then we started implementation

RTL Schematic of signle module 32-bit conditional sum adder:



.

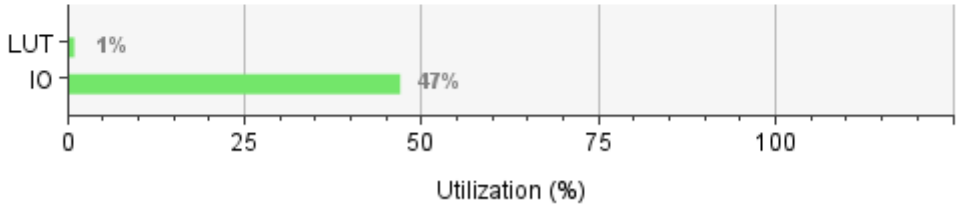
Technology Schematic of single module 32-bit conditional sum adder:



Utilization Report of single module 32-bit conditional sum adder:

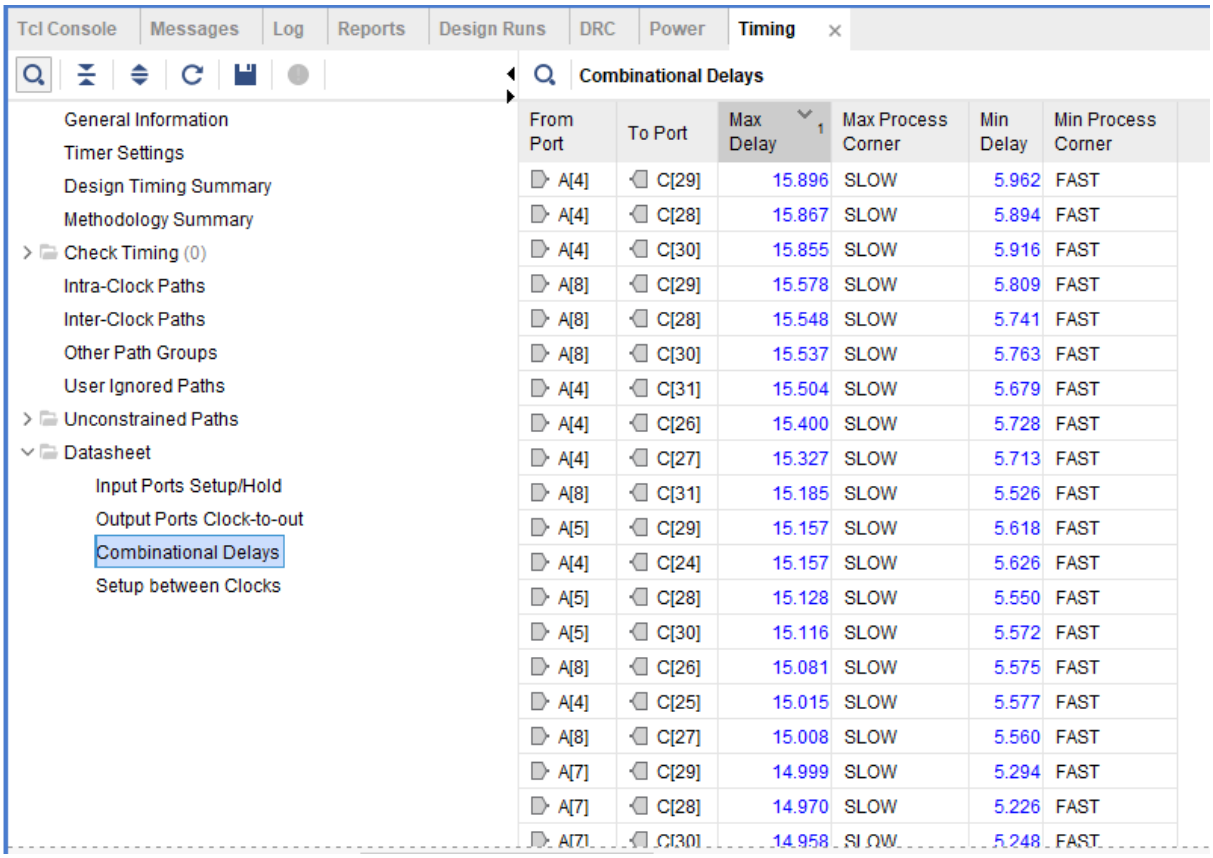
Summary

Resource	Utilization	Available	Utilization %
LUT	107	32600	0.33
IO	99	210	47.14



107 LUT used in this design.

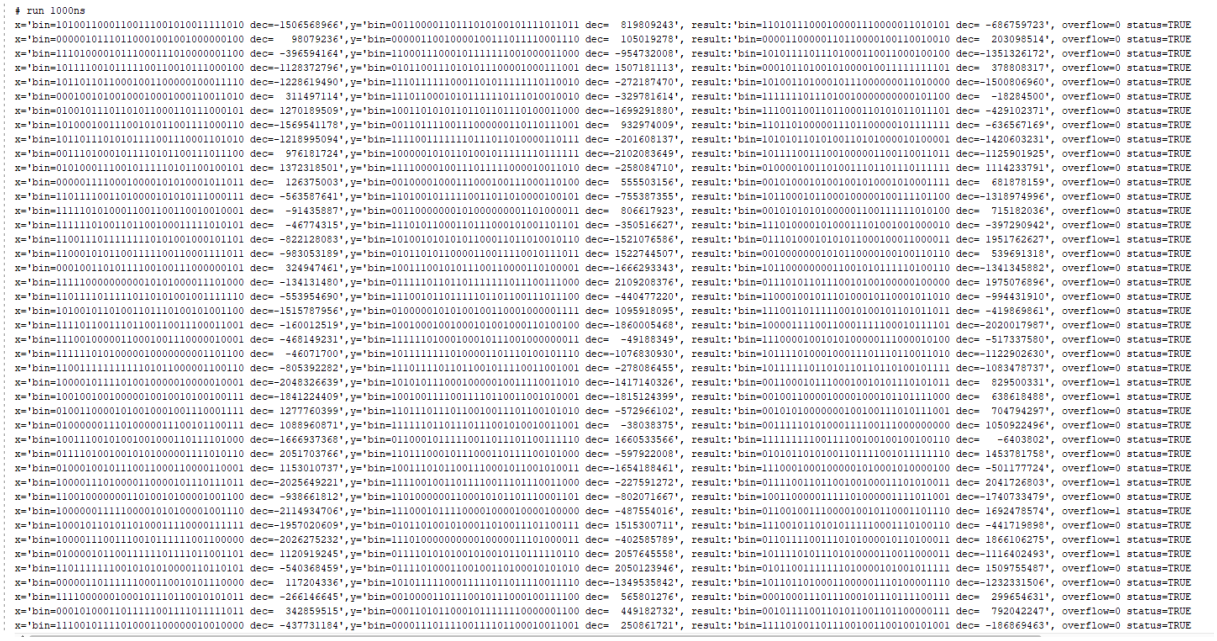
Timing Report of sinlge module 32-bit conditional sum adder:



From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
A[4]	C[29]	15.896	SLOW	5.962	FAST
A[4]	C[28]	15.867	SLOW	5.894	FAST
A[4]	C[30]	15.855	SLOW	5.916	FAST
A[8]	C[29]	15.578	SLOW	5.809	FAST
A[8]	C[28]	15.548	SLOW	5.741	FAST
A[8]	C[30]	15.537	SLOW	5.763	FAST
A[4]	C[31]	15.504	SLOW	5.679	FAST
A[4]	C[26]	15.400	SLOW	5.728	FAST
A[4]	C[27]	15.327	SLOW	5.713	FAST
A[8]	C[31]	15.185	SLOW	5.526	FAST
A[5]	C[29]	15.157	SLOW	5.618	FAST
A[4]	C[24]	15.157	SLOW	5.626	FAST
A[5]	C[28]	15.128	SLOW	5.550	FAST
A[5]	C[30]	15.116	SLOW	5.572	FAST
A[8]	C[26]	15.081	SLOW	5.575	FAST
A[4]	C[25]	15.015	SLOW	5.577	FAST
A[8]	C[27]	15.008	SLOW	5.560	FAST
A[7]	C[29]	14.999	SLOW	5.294	FAST
A[7]	C[28]	14.970	SLOW	5.226	FAST
A[7]	C[30]	14.958	SLOW	5.248	FAST

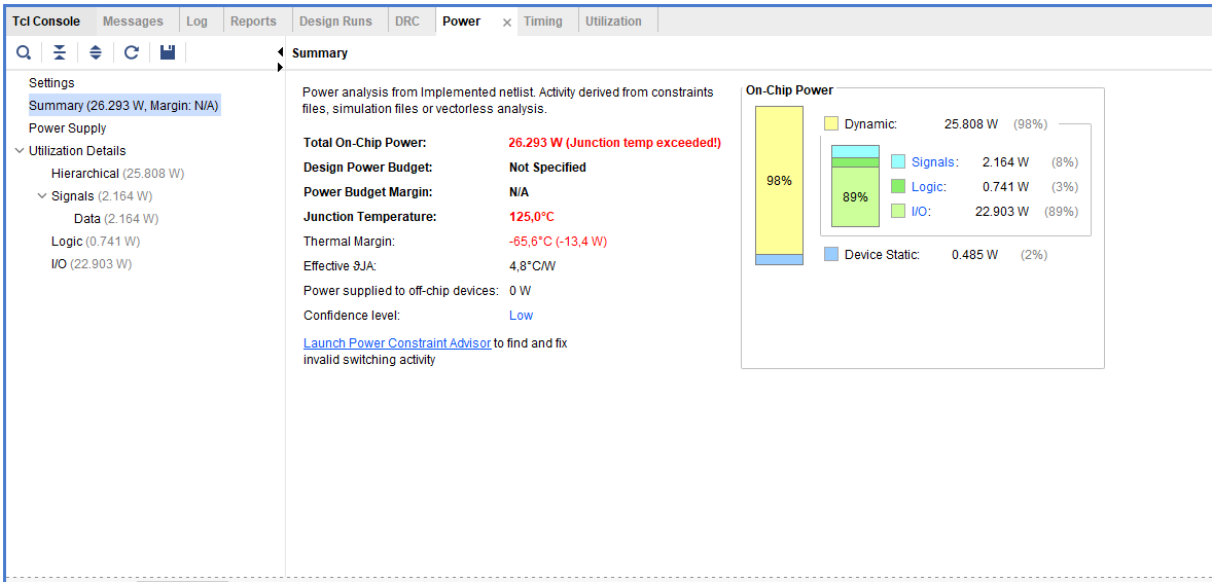
Our maximum delay is 15.896 ns. So, our maximum clock frequency is 1/((15.896)*(10e-9)) = **62.90MHz.**

TCL Console output when simulation applied to single module 32-bit conditional sum adder:



Their both console output ,output waves and outfile is same and all true. Output file can be found in /sim/sim_1/behav/xsim/outfile.txt in project file.

Power Report of single module 32-bit conditional sum adder:



Power analysis shows that power on-chip is 26.293 Watts and according to we wrote (*DONT_TOUCH*), power analysis works in vectorless estimation mode and that result is average.

Verilog Code of single module 32-bit conditional sum adder

```
`timescale 1ns / 1ps
(* DONT_TOUCH = "TRUE" *)
module cond_sum_adder(
    input [31:0]A,
    input [31:0]B,
    input cin,
    output v,
    output carry,
    output [31:0]C
    );

    wire S00010; //Naming [ 'S'um or 'C'arryout][ Number( 00, 01, 02,..,
31)][ Block width( 01, 02, 04, 08, 16, 32)][ Carry In( 0, 1)]
    wire S01010;
    wire S02010;
    wire S03010;
    wire S04010;
    wire S05010;
    wire S06010;
    wire S07010;
    wire S08010;
    wire S09010;
    wire S10010;
    wire S11010;
    wire S12010;
    wire S13010;
    wire S14010;
    wire S15010;
    wire S16010;
    wire S17010;
    wire S18010;
    wire S19010;
    wire S20010;
    wire S21010;
    wire S22010;
    wire S23010;
    wire S24010;
    wire S25010;
    wire S26010;
    wire S27010;
    wire S28010;
    wire S29010;
    wire S30010;
    wire S31010;

    wire C00010; // carry
    wire C01010;
    wire C02010;
    wire C03010;
    wire C04010;
    wire C05010;
    wire C06010;
    wire C07010;
    wire C08010;
    wire C09010;
    wire C10010;
```

```
    wire C11010;
    wire C12010;
    wire C13010;
    wire C14010;
    wire C15010;
    wire C16010;
    wire C17010;
    wire C18010;
    wire C19010;
    wire C20010;
    wire C21010;
    wire C22010;
    wire C23010;
    wire C24010;
    wire C25010;
    wire C26010;
    wire C27010;
    wire C28010;
    wire C29010;
    wire C30010;
    wire C31010;


    wire S00011; //block width 1 and carry in 1
    wire S01011;
    wire S02011;
    wire S03011;
    wire S04011;
    wire S05011;
    wire S06011;
    wire S07011;
    wire S08011;
    wire S09011;
    wire S10011;
    wire S11011;
    wire S12011;
    wire S13011;
    wire S14011;
    wire S15011;
    wire S16011;
    wire S17011;
    wire S18011;
    wire S19011;
    wire S20011;
    wire S21011;
    wire S22011;
    wire S23011;
    wire S24011;
    wire S25011;
    wire S26011;
    wire S27011;
    wire S28011;
    wire S29011;
    wire S30011;
    wire S31011;

    wire C00011;// carry
    wire C01011;
    wire C02011;
    wire C03011;
    wire C04011;
```

```
    wire C05011;
    wire C06011;
    wire C07011;
    wire C08011;
    wire C09011;
    wire C10011;
    wire C11011;
    wire C12011;
    wire C13011;
    wire C14011;
    wire C15011;
    wire C16011;
    wire C17011;
    wire C18011;
    wire C19011;
    wire C20011;
    wire C21011;
    wire C22011;
    wire C23011;
    wire C24011;
    wire C25011;
    wire C26011;
    wire C27011;
    wire C28011;
    wire C29011;
    wire C30011;
    wire C31011;

    wire [1:0] S00020; //block width 2 and carry in 0
    wire [1:0] S01020;
    wire [1:0] S02020;
    wire [1:0] S03020;
    wire [1:0] S04020;
    wire [1:0] S05020;
    wire [1:0] S06020;
    wire [1:0] S07020;
    wire [1:0] S08020;
    wire [1:0] S09020;
    wire [1:0] S10020;
    wire [1:0] S11020;
    wire [1:0] S12020;
    wire [1:0] S13020;
    wire [1:0] S14020;
    wire [1:0] S15020;

    wire C00020; // carry
    wire C01020;
    wire C02020;
    wire C03020;
    wire C04020;
    wire C05020;
    wire C06020;
    wire C07020;
    wire C08020;
    wire C09020;
    wire C10020;
    wire C11020;
    wire C12020;
    wire C13020;
    wire C14020;
    wire C15020;
```

```
    wire [1:0] S00021; //block width 2 and carry in 1
    wire [1:0] S01021;
    wire [1:0] S02021;
    wire [1:0] S03021;
    wire [1:0] S04021;
    wire [1:0] S05021;
    wire [1:0] S06021;
    wire [1:0] S07021;
    wire [1:0] S08021;
    wire [1:0] S09021;
    wire [1:0] S10021;
    wire [1:0] S11021;
    wire [1:0] S12021;
    wire [1:0] S13021;
    wire [1:0] S14021;
    wire [1:0] S15021;

    wire C00021;// carry
    wire C01021;
    wire C02021;
    wire C03021;
    wire C04021;
    wire C05021;
    wire C06021;
    wire C07021;
    wire C08021;
    wire C09021;
    wire C10021;
    wire C11021;
    wire C12021;
    wire C13021;
    wire C14021;
    wire C15021;

    wire [3:0] S00040; //block width 4 and carry in 0
    wire [3:0] S01040;
    wire [3:0] S02040;
    wire [3:0] S03040;
    wire [3:0] S04040;
    wire [3:0] S05040;
    wire [3:0] S06040;
    wire [3:0] S07040;

    wire C00040; // carry
    wire C01040;
    wire C02040;
    wire C03040;
    wire C04040;
    wire C05040;
    wire C06040;
    wire C07040;

    wire [3:0] S00041; //block width 4 and carry in 1
    wire [3:0] S01041;
    wire [3:0] S02041;
    wire [3:0] S03041;
    wire [3:0] S04041;
    wire [3:0] S05041;
    wire [3:0] S06041;
    wire [3:0] S07041;
```

```
    wire C00041;// carry
    wire C01041;
    wire C02041;
    wire C03041;
    wire C04041;
    wire C05041;
    wire C06041;
    wire C07041;

    wire [7:0] S00080; //block width 8 and carry in 0
    wire [7:0] S01080;
    wire [7:0] S02080;
    wire [7:0] S03080;

    wire C00080; // carry
    wire C01080;
    wire C02080;
    wire C03080;

    wire [7:0] S00081; //block width 8 and carry in 1
    wire [7:0] S01081;
    wire [7:0] S02081;
    wire [7:0] S03081;

    wire C00081;// carry
    wire C01081;
    wire C02081;
    wire C03081;

    wire [15:0] S00160; //block width 16 and carry in 0
    wire [15:0] S01160;

    wire C00160; // carry
    wire C01160;

    wire [15:0] S00161; //block width 16 and carry in 1
    wire [15:0] S01161;

    wire C00161;// carry
    wire C01161;

    wire [31:0] S00320; //block width 32 and carry in 0

    wire C00320; // carry

    wire [31:0] S00321; //block width 32 and carry in 1

    wire C00321; // carry

    wire tempC01;
    wire tempC02;
    wire tempC03;
    wire tempC04;
    wire tempC05;
    wire tempC06;
    wire tempC07;
    wire tempC08;
    wire tempC09;
    wire tempC10;
    wire tempC11;
```

```
    wire tempC12;
    wire tempC13;
    wire tempC14;
    wire tempC15;
    wire tempC16;
    wire tempC17;
    wire tempC18;
    wire tempC19;
    wire tempC20;
    wire tempC21;
    wire tempC22;
    wire tempC23;
    wire tempC24;
    wire tempC25;
    wire tempC26;
    wire tempC27;
    wire tempC28;
    wire tempC29;
    wire tempC30;
    wire tempC31;
    wire tempC32;
    wire tempC33;
    wire tempC34;
    wire tempC35;
    wire tempC36;
    wire tempC37;
    wire tempC38;
    wire tempC39;
    wire tempC40;
    wire tempC41;
    wire tempC42;
    wire tempC43;
    wire tempC44;
    wire tempC45;
    wire tempC46;
    wire tempC47;
    wire tempC48;
    wire tempC49;
    wire tempC50;
    wire tempC51;
    wire tempC52;
    wire tempC53;
    wire tempC54;
    wire tempC55;
    wire tempC56;
    wire tempC57;
    wire tempC58;
    wire tempC59;
    wire tempC60;
    wire tempC61;
    wire tempC62;

    //full adder for cin 0
    //                  sum      cout    inp1  inp2   cin
    fullAdder inst01(S00010, C00010, A[0], B[0], 1'b0);
    fullAdder inst02(S01010, C01010, A[1], B[1], 1'b0);
    fullAdder inst03(S02010, C02010, A[2], B[2], 1'b0);
    fullAdder inst04(S03010, C03010, A[3], B[3], 1'b0);
    fullAdder inst05(S04010, C04010, A[4], B[4], 1'b0);
    fullAdder inst06(S05010, C05010, A[5], B[5], 1'b0);
    fullAdder inst07(S06010, C06010, A[6], B[6], 1'b0);
```

```
    fullAdder inst08(S07010, C07010, A[7], B[7], 1'b0);
    fullAdder inst09(S08010, C08010, A[8], B[8], 1'b0);
    fullAdder inst10(S09010, C09010, A[9], B[9], 1'b0);
    fullAdder inst11(S10010, C10010, A[10], B[10], 1'b0);
    fullAdder inst12(S11010, C11010, A[11], B[11], 1'b0);
    fullAdder inst13(S12010, C12010, A[12], B[12], 1'b0);
    fullAdder inst14(S13010, C13010, A[13], B[13], 1'b0);
    fullAdder inst15(S14010, C14010, A[14], B[14], 1'b0);
    fullAdder inst16(S15010, C15010, A[15], B[15], 1'b0);
    fullAdder inst17(S16010, C16010, A[16], B[16], 1'b0);
    fullAdder inst18(S17010, C17010, A[17], B[17], 1'b0);
    fullAdder inst19(S18010, C18010, A[18], B[18], 1'b0);
    fullAdder inst20(S19010, C19010, A[19], B[19], 1'b0);
    fullAdder inst21(S20010, C20010, A[20], B[20], 1'b0);
    fullAdder inst22(S21010, C21010, A[21], B[21], 1'b0);
    fullAdder inst23(S22010, C22010, A[22], B[22], 1'b0);
    fullAdder inst24(S23010, C23010, A[23], B[23], 1'b0);
    fullAdder inst25(S24010, C24010, A[24], B[24], 1'b0);
    fullAdder inst26(S25010, C25010, A[25], B[25], 1'b0);
    fullAdder inst27(S26010, C26010, A[26], B[26], 1'b0);
    fullAdder inst28(S27010, C27010, A[27], B[27], 1'b0);
    fullAdder inst29(S28010, C28010, A[28], B[28], 1'b0);
    fullAdder inst30(S29010, C29010, A[29], B[29], 1'b0);
    fullAdder inst31(S30010, C30010, A[30], B[30], 1'b0);
    fullAdder inst32(S31010, C31010, A[31], B[31], 1'b0);

    //full adder for cin 1
    //                 sum      cout  inp1  inp2  cin
    fullAdder inst33(S00011, C00011, A[0], B[0], 1'b1);
    fullAdder inst34(S01011, C01011, A[1], B[1], 1'b1);
    fullAdder inst35(S02011, C02011, A[2], B[2], 1'b1);
    fullAdder inst36(S03011, C03011, A[3], B[3], 1'b1);
    fullAdder inst37(S04011, C04011, A[4], B[4], 1'b1);
    fullAdder inst38(S05011, C05011, A[5], B[5], 1'b1);
    fullAdder inst39(S06011, C06011, A[6], B[6], 1'b1);
    fullAdder inst40(S07011, C07011, A[7], B[7], 1'b1);
    fullAdder inst41(S08011, C08011, A[8], B[8], 1'b1);
    fullAdder inst42(S09011, C09011, A[9], B[9], 1'b1);
    fullAdder inst43(S10011, C10011, A[10], B[10], 1'b1);
    fullAdder inst44(S11011, C11011, A[11], B[11], 1'b1);
    fullAdder inst45(S12011, C12011, A[12], B[12], 1'b1);
    fullAdder inst46(S13011, C13011, A[13], B[13], 1'b1);
    fullAdder inst47(S14011, C14011, A[14], B[14], 1'b1);
    fullAdder inst48(S15011, C15011, A[15], B[15], 1'b1);
    fullAdder inst49(S16011, C16011, A[16], B[16], 1'b1);
    fullAdder inst50(S17011, C17011, A[17], B[17], 1'b1);
    fullAdder inst51(S18011, C18011, A[18], B[18], 1'b1);
    fullAdder inst52(S19011, C19011, A[19], B[19], 1'b1);
    fullAdder inst53(S20011, C20011, A[20], B[20], 1'b1);
    fullAdder inst54(S21011, C21011, A[21], B[21], 1'b1);
    fullAdder inst55(S22011, C22011, A[22], B[22], 1'b1);
    fullAdder inst56(S23011, C23011, A[23], B[23], 1'b1);
    fullAdder inst57(S24011, C24011, A[24], B[24], 1'b1);
    fullAdder inst58(S25011, C25011, A[25], B[25], 1'b1);
    fullAdder inst59(S26011, C26011, A[26], B[26], 1'b1);
    fullAdder inst60(S27011, C27011, A[27], B[27], 1'b1);
    fullAdder inst61(S28011, C28011, A[28], B[28], 1'b1);
    fullAdder inst62(S29011, C29011, A[29], B[29], 1'b1);
    fullAdder inst63(S30011, C30011, A[30], B[30], 1'b1);
    fullAdder inst64(S31011, C31011, A[31], B[31], 1'b1);
```

```
    //mux for block width 1 carry in 0
    multiplexer01 inst65( {tempC01,S00020[0]}, {C00010,S00010},
{C00011,S00011}, 1'b0);
    multiplexer01 inst66( {C00020,S00020[1]}, {C01010,S01010},
{C01011,S01011}, tempC01);
    multiplexer01 inst67( {tempC02,S01020[0]}, {C02010,S02010},
{C02011,S02011}, 1'b0);
    multiplexer01 inst68( {C01020,S01020[1]}, {C03010,S03010},
{C03011,S03011}, tempC02);
    multiplexer01 inst69( {tempC03,S02020[0]}, {C04010,S04010},
{C04011,S04011}, 1'b0);
    multiplexer01 inst70( {C02020,S02020[1]}, {C05010,S05010},
{C05011,S05011}, tempC03);
    multiplexer01 inst71( {tempC04,S03020[0]}, {C06010,S06010},
{C06011,S06011}, 1'b0);
    multiplexer01 inst72( {C03020,S03020[1]}, {C07010,S07010},
{C07011,S07011}, tempC04);
    multiplexer01 inst73( {tempC05,S04020[0]}, {C08010,S08010},
{C08011,S08011}, 1'b0);
    multiplexer01 inst74( {C04020,S04020[1]}, {C09010,S09010},
{C09011,S09011}, tempC05);
    multiplexer01 inst75( {tempC06,S05020[0]}, {C10010,S10010},
{C10011,S10011}, 1'b0);
    multiplexer01 inst76( {C05020,S05020[1]}, {C11010,S11010},
{C11011,S11011}, tempC06);
    multiplexer01 inst77( {tempC07,S06020[0]}, {C12010,S12010},
{C12011,S12011}, 1'b0);
    multiplexer01 inst78( {C06020,S06020[1]}, {C13010,S13010},
{C13011,S13011}, tempC07);
    multiplexer01 inst79( {tempC08,S07020[0]}, {C14010,S14010},
{C14011,S14011}, 1'b0);
    multiplexer01 inst80( {C07020,S07020[1]}, {C15010,S15010},
{C15011,S15011}, tempC08);
    multiplexer01 inst81( {tempC09,S08020[0]}, {C16010,S16010},
{C16011,S16011}, 1'b0);
    multiplexer01 inst82( {C08020,S08020[1]}, {C17010,S17010},
{C17011,S17011}, tempC09);
    multiplexer01 inst83( {tempC10,S09020[0]}, {C18010,S18010},
{C18011,S18011}, 1'b0);
    multiplexer01 inst84( {C09020,S09020[1]}, {C19010,S19010},
{C19011,S19011}, tempC10);
    multiplexer01 inst85( {tempC11,S10020[0]}, {C20010,S20010},
{C20011,S20011}, 1'b0);
    multiplexer01 inst86( {C10020,S10020[1]}, {C21010,S21010},
{C21011,S21011}, tempC11);
    multiplexer01 inst87( {tempC12,S11020[0]}, {C22010,S22010},
{C22011,S22011}, 1'b0);
    multiplexer01 inst88( {C11020,S11020[1]}, {C23010,S23010},
{C23011,S23011}, tempC12);
    multiplexer01 inst89( {tempC13,S12020[0]}, {C24010,S24010},
{C24011,S24011}, 1'b0);
    multiplexer01 inst90( {C12020,S12020[1]}, {C25010,S25010},
{C25011,S25011}, tempC13);
    multiplexer01 inst91( {tempC14,S13020[0]}, {C26010,S26010},
{C26011,S26011}, 1'b0);
    multiplexer01 inst92( {C13020,S13020[1]}, {C27010,S27010},
{C27011,S27011}, tempC14);
    multiplexer01 inst93( {tempC15,S14020[0]}, {C28010,S28010},
{C28011,S28011}, 1'b0);
```

```
    multiplexer01 inst94( {C14020,S14020[1]}, {C29010,S29010},
{C29011,S29011}, tempC15);
    multiplexer01 inst95( {tempC16,S15020[0]}, {C30010,S30010},
{C30011,S30011}, 1'b0);
    multiplexer01 inst96( {C15020,S15020[1]}, {C31010,S31010},
{C31011,S31011}, tempC16);

    //mux for block width 1 carry in 1
    multiplexer01 inst97( {tempC17,S00021[0]}, {C00010,S00010},
{C00011,S00011}, 1'b1);
    multiplexer01 inst98( {C00021,S00021[1]}, {C01010,S01010},
{C01011,S01011}, tempC17);
    multiplexer01 inst99( {tempC18,S01021[0]}, {C02010,S02010},
{C02011,S02011}, 1'b1);
    multiplexer01 inst100( {C01021,S01021[1]}, {C03010,S03010},
{C03011,S03011}, tempC18);
    multiplexer01 inst101( {tempC19,S02021[0]}, {C04010,S04010},
{C04011,S04011}, 1'b1);
    multiplexer01 inst102( {C02021,S02021[1]}, {C05010,S05010},
{C05011,S05011}, tempC19);
    multiplexer01 inst103( {tempC20,S03021[0]}, {C06010,S06010},
{C06011,S06011}, 1'b1);
    multiplexer01 inst104( {C03021,S03021[1]}, {C07010,S07010},
{C07011,S07011}, tempC20);
    multiplexer01 inst105( {tempC21,S04021[0]}, {C08010,S08010},
{C08011,S08011}, 1'b1);
    multiplexer01 inst106( {C04021,S04021[1]}, {C09010,S09010},
{C09011,S09011}, tempC21);
    multiplexer01 inst107( {tempC22,S05021[0]}, {C10010,S10010},
{C10011,S10011}, 1'b1);
    multiplexer01 inst108( {C05021,S05021[1]}, {C11010,S11010},
{C11011,S11011}, tempC22);
    multiplexer01 inst109( {tempC23,S06021[0]}, {C12010,S12010},
{C12011,S12011}, 1'b1);
    multiplexer01 inst110( {C06021,S06021[1]}, {C13010,S13010},
{C13011,S13011}, tempC23);
    multiplexer01 inst111( {tempC24,S07021[0]}, {C14010,S14010},
{C14011,S14011}, 1'b1);
    multiplexer01 inst112( {C07021,S07021[1]}, {C15010,S15010},
{C15011,S15011}, tempC24);
    multiplexer01 inst113( {tempC25,S08021[0]}, {C16010,S16010},
{C16011,S16011}, 1'b1);
    multiplexer01 inst114( {C08021,S08021[1]}, {C17010,S17010},
{C17011,S17011}, tempC25);
    multiplexer01 inst115( {tempC26,S09021[0]}, {C18010,S18010},
{C18011,S18011}, 1'b1);
    multiplexer01 inst116( {C09021,S09021[1]}, {C19010,S19010},
{C19011,S19011}, tempC26);
    multiplexer01 inst117( {tempC27,S10021[0]}, {C20010,S20010},
{C20011,S20011}, 1'b1);
    multiplexer01 inst118( {C10021,S10021[1]}, {C21010,S21010},
{C21011,S21011}, tempC27);
    multiplexer01 inst119( {tempC28,S11021[0]}, {C22010,S22010},
{C22011,S22011}, 1'b1);
    multiplexer01 inst120( {C11021,S11021[1]}, {C23010,S23010},
{C23011,S23011}, tempC28);
    multiplexer01 inst121( {tempC29,S12021[0]}, {C24010,S24010},
{C24011,S24011}, 1'b1);
    multiplexer01 inst122( {C12021,S12021[1]}, {C25010,S25010},
{C25011,S25011}, tempC29);
```

```
    multiplexer01 inst123( {tempC30,S13021[0]}, {C26010,S26010},
{C26011,S26011}, 1'b1);
    multiplexer01 inst124( {C13021,S13021[1]}, {C27010,S27010},
{C27011,S27011}, tempC30);
    multiplexer01 inst125( {tempC31,S14021[0]}, {C28010,S28010},
{C28011,S28011}, 1'b1);
    multiplexer01 inst126( {C14021,S14021[1]}, {C29010,S29010},
{C29011,S29011}, tempC31);
    multiplexer01 inst127( {tempC32,S15021[0]}, {C30010,S30010},
{C30011,S30011}, 1'b1);
    multiplexer01 inst128( {C15021,S15021[1]}, {C31010,S31010},
{C31011,S31011}, tempC32);


    //mux for block width 2 carry in 0
    multiplexer02 inst129( {tempC33,S00040[1:0]}, {C00020,S00020[1:0]},
{C00021,S00021[1:0]}, 1'b0);
    multiplexer02 inst130( {C00040,S00040[3:2]}, {C01020,S01020[1:0]},
{C01021,S01021[1:0]}, tempC33);
    multiplexer02 inst131( {tempC34,S01040[1:0]}, {C02020,S02020[1:0]},
{C02021,S02021[1:0]}, 1'b0);
    multiplexer02 inst132( {C01040,S01040[3:2]}, {C03020,S03020[1:0]},
{C03021,S03021[1:0]}, tempC34);
    multiplexer02 inst133( {tempC35,S02040[1:0]}, {C04020,S04020[1:0]},
{C04021,S04021[1:0]}, 1'b0);
    multiplexer02 inst134( {C02040,S02040[3:2]}, {C05020,S05020[1:0]},
{C05021,S05021[1:0]}, tempC35);
    multiplexer02 inst135( {tempC36,S03040[1:0]}, {C06020,S06020[1:0]},
{C06021,S06021[1:0]}, 1'b0);
    multiplexer02 inst136( {C03040,S03040[3:2]}, {C07020,S07020[1:0]},
{C07021,S07021[1:0]}, tempC36);
    multiplexer02 inst137( {tempC37,S04040[1:0]}, {C08020,S08020[1:0]},
{C08021,S08021[1:0]}, 1'b0);
    multiplexer02 inst138( {C04040,S04040[3:2]}, {C09020,S09020[1:0]},
{C09021,S09021[1:0]}, tempC37);
    multiplexer02 inst139( {tempC38,S05040[1:0]}, {C10020,S10020[1:0]},
{C10021,S10021[1:0]}, 1'b0);
    multiplexer02 inst140( {C05040,S05040[3:2]}, {C11020,S11020[1:0]},
{C11021,S11021[1:0]}, tempC38);
    multiplexer02 inst141( {tempC39,S06040[1:0]}, {C12020,S12020[1:0]},
{C12021,S12021[1:0]}, 1'b0);
    multiplexer02 inst142( {C06040,S06040[3:2]}, {C13020,S13020[1:0]},
{C13021,S13021[1:0]}, tempC39);
    multiplexer02 inst143( {tempC40,S07040[1:0]}, {C14020,S14020[1:0]},
{C14021,S14021[1:0]}, 1'b0);
    multiplexer02 inst144( {C07040,S07040[3:2]}, {C15020,S15020[1:0]},
{C15021,S15021[1:0]}, tempC40);

    //mux for block width 2 carry in 1
    multiplexer02 inst145( {tempC41,S00041[1:0]}, {C00020,S00020[1:0]},
{C00021,S00021[1:0]}, 1'b1);
    multiplexer02 inst146( {C00041,S00041[3:2]}, {C01020,S01020[1:0]},
{C01021,S01021[1:0]}, tempC41);
    multiplexer02 inst147( {tempC42,S01041[1:0]}, {C02020,S02020[1:0]},
{C02021,S02021[1:0]}, 1'b1);
    multiplexer02 inst148( {C01041,S01041[3:2]}, {C03020,S03020[1:0]},
{C03021,S03021[1:0]}, tempC42);
    multiplexer02 inst149( {tempC43,S02041[1:0]}, {C04020,S04020[1:0]},
{C04021,S04021[1:0]}, 1'b1);
    multiplexer02 inst150( {C02041,S02041[3:2]}, {C05020,S05020[1:0]},
{C05021,S05021[1:0]}, tempC43);
```

```
    multiplexer02 inst151( {tempC44,S03041[1:0]}, {C06020,S06020[1:0]},
{C06021,S06021[1:0]}, 1'b1);
    multiplexer02 inst152( {C03041,S03041[3:2]}, {C07020,S07020[1:0]},
{C07021,S07021[1:0]}, tempC44);
    multiplexer02 inst153( {tempC45,S04041[1:0]}, {C08020,S08020[1:0]},
{C08021,S08021[1:0]}, 1'b1);
    multiplexer02 inst154( {C04041,S04041[3:2]}, {C09020,S09020[1:0]},
{C09021,S09021[1:0]}, tempC45);
    multiplexer02 inst155( {tempC46,S05041[1:0]}, {C10020,S10020[1:0]},
{C10021,S10021[1:0]}, 1'b1);
    multiplexer02 inst156( {C05041,S05041[3:2]}, {C11020,S11020[1:0]},
{C11021,S11021[1:0]}, tempC46);
    multiplexer02 inst157( {tempC47,S06041[1:0]}, {C12020,S12020[1:0]},
{C12021,S12021[1:0]}, 1'b1);
    multiplexer02 inst158( {C06041,S06041[3:2]}, {C13020,S13020[1:0]},
{C13021,S13021[1:0]}, tempC47);
    multiplexer02 inst159( {tempC48,S07041[1:0]}, {C14020,S14020[1:0]},
{C14021,S14021[1:0]}, 1'b1);
    multiplexer02 inst160( {C07041,S07041[3:2]}, {C15020,S15020[1:0]},
{C15021,S15021[1:0]}, tempC48);


    //mux for block width 4 carry in 0
    multiplexer04 inst161( {tempC49,S00080[3:0]}, {C00040,S00040[3:0]},
{C00041,S00041[3:0]}, 1'b0);
    multiplexer04 inst162( {C00080,S00080[7:4]}, {C01040,S01040[3:0]},
{C01041,S01041[3:0]}, tempC49);
    multiplexer04 inst163( {tempC50,S01080[3:0]}, {C02040,S02040[3:0]},
{C02041,S02041[3:0]}, 1'b0);
    multiplexer04 inst164( {C01080,S01080[7:4]}, {C03040,S03040[3:0]},
{C03041,S03041[3:0]}, tempC50);
    multiplexer04 inst165( {tempC51,S02080[3:0]}, {C04040,S04040[3:0]},
{C04041,S04041[3:0]}, 1'b0);
    multiplexer04 inst166( {C02080,S02080[7:4]}, {C05040,S05040[3:0]},
{C05041,S05041[3:0]}, tempC51);
    multiplexer04 inst167( {tempC52,S03080[3:0]}, {C06040,S06040[3:0]},
{C06041,S06041[3:0]}, 1'b0);
    multiplexer04 inst168( {C03080,S03080[7:4]}, {C07040,S07040[3:0]},
{C07041,S07041[3:0]}, tempC52);

    //mux for block width 4 carry in 1
    multiplexer04 inst169( {tempC53,S00081[3:0]}, {C00040,S00040[3:0]},
{C00041,S00041[3:0]}, 1'b1);
    multiplexer04 inst170( {C00081,S00081[7:4]}, {C01040,S01040[3:0]},
{C01041,S01041[3:0]}, tempC53);
    multiplexer04 inst171( {tempC54,S01081[3:0]}, {C02040,S02040[3:0]},
{C02041,S02041[3:0]}, 1'b1);
    multiplexer04 inst172( {C01081,S01081[7:4]}, {C03040,S03040[3:0]},
{C03041,S03041[3:0]}, tempC54);
    multiplexer04 inst173( {tempC55,S02081[3:0]}, {C04040,S04040[3:0]},
{C04041,S04041[3:0]}, 1'b1);
    multiplexer04 inst174( {C02081,S02081[7:4]}, {C05040,S05040[3:0]},
{C05041,S05041[3:0]}, tempC55);
    multiplexer04 inst175( {tempC56,S03081[3:0]}, {C06040,S06040[3:0]},
{C06041,S06041[3:0]}, 1'b1);
    multiplexer04 inst176( {C03081,S03081[7:4]}, {C07040,S07040[3:0]},
{C07041,S07041[3:0]}, tempC56);


    //mux for block width 8 carry in 0
```

```
    multiplexer08 inst177( {tempC57,S00160[7:0]}, {C00080,S00080[7:0]},
{C00081,S00081[7:0]}, 1'b0);
    multiplexer08 inst178( {C00160,S00160[15:8]}, {C01080,S01080[7:0]},
{C01081,S01081[7:0]}, tempC57);
    multiplexer08 inst179( {tempC58,S01160[7:0]}, {C02080,S02080[7:0]},
{C02081,S02081[7:0]}, 1'b0);
    multiplexer08 inst180( {C01160,S01160[15:8]}, {C03080,S03080[7:0]},
{C03081,S03081[7:0]}, tempC58);

    //mux for block width 8 carry in 1
    multiplexer08 inst181( {tempC59,S00161[7:0]}, {C00080,S00080[7:0]},
{C00081,S00081[7:0]}, 1'b1);
    multiplexer08 inst182( {C00161,S00161[15:8]}, {C01080,S01080[7:0]},
{C01081,S01081[7:0]}, tempC59);
    multiplexer08 inst183( {tempC60,S01161[7:0]}, {C02080,S02080[7:0]},
{C02081,S02081[7:0]}, 1'b1);
    multiplexer08 inst184( {C01161,S01161[15:8]}, {C03080,S03080[7:0]},
{C03081,S03081[7:0]}, tempC60);

    //mux for block width 16 carry in 0
    multiplexer16 inst185( {tempC61,S00320[15:0]}, {C00160,S00160[15:0]},
{C00161,S00161[15:0]}, 1'b0);
    multiplexer16 inst186( {C00320,S00320[31:16]}, {C01160,S01160[15:0]},
{C01161,S01161[15:0]}, tempC61);

    //mux for block width 16 carry in 1
    multiplexer16 inst187( {tempC62,S00321[15:0]}, {C00160,S00160[15:0]},
{C00161,S00161[15:0]}, 1'b1);
    multiplexer16 inst188( {C00321,S00321[31:16]}, {C01160,S01160[15:0]},
{C01161,S01161[15:0]}, tempC62);

    //mux for block width 32 carry in Cin
    multiplexer32 inst189( {carry,C[31:0]}, {C00320,S00320[31:0]},
{C00321,S00321[31:0]}, cin);

    assign v = carry ^ (C00320 | C00321);
endmodule

module fullAdder(sum, cout, op1, op2, cin);
    input cin, op1, op2;
    output cout, sum;

    assign sum = cin^(op1^op2);
    assign cout = ((op1^op2)&cin) | (op1&op2);

endmodule

module multiplexer01(f, inp1, inp2, sel);//to choose from block width 1
    input sel;
    input [1:0] inp1;
    input [1:0] inp2;
    output wire [1:0] f;

    assign f = sel ? inp2 : inp1;
endmodule

module multiplexer02(f, inp1, inp2, sel);//to choose from block width 2
    input sel;
    input [2:0] inp1;
    input [2:0] inp2;
    output wire [2:0] f;
```

```
    assign f = sel ? inp2 : inp1;
endmodule

module multiplexer04(f, inp1, inp2, sel);//to choose from block width 4
    input sel;
    input [4:0] inp1;
    input [4:0] inp2;
    output wire [4:0] f;

    assign f = sel ? inp2 : inp1;
endmodule

module multiplexer08(f, inp1, inp2, sel);//to choose from block width 8
    input sel;
    input [8:0] inp1;
    input [8:0] inp2;
    output wire [8:0] f;

    assign f = sel ? inp2 : inp1;
endmodule

module multiplexer16(f, inp1, inp2, sel);//to choose from block width 16
    input sel;
    input [16:0] inp1;
    input [16:0] inp2;
    output wire [16:0] f;

    assign f = sel ? inp2 : inp1;
endmodule

module multiplexer32(f, inp1, inp2, sel);//to choose from block width 32
    input sel;
    input [32:0] inp1;
    input [32:0] inp2;
    output wire [32:0] f;

    assign f = sel ? inp2 : inp1;
endmodule
```

Testbench code of single module 32-bit conditional sum adder.

```
`timescale 1ns / 1ps

module conditional_sum_adder_tb;

reg signed [31:0] A [0:99];
reg signed [31:0] B [0:99];
reg cin;
wire cout_out;
wire signed [31:0] sumout;
reg signed [31:0] CALCULATOR;
reg signed [31:0] xwire;
reg signed [31:0] ywire;
wire v;
cond_sum_adder
TOP(.A(xwire),.B(ywire),.cin(cin),.carry(cout_out),.v(v),.C(sumout));

integer n;
initial $readmemb("xin.txt",A);
```

```
initial $readmemb("yin.txt",B);
initial n = $fopen("outfile.txt","w");


integer i;

initial begin
cin = 0;
for (i=0;i<=99;i=i+1)
begin
 xwire = A[i];
 ywire = B[i];
 CALCULATOR = A[i]+B[i]+cin;
 #5;
 if(CALCULATOR == sumout)
 begin
 $fdisplay (n,"x='bin=%b dec=%d',y='bin=%b dec=%d', result:'bin=%b dec=%d',
overflow=%d status=TRUE", xwire,xwire,ywire,ywire,sumout,sumout,v);
 $display ("x='bin=%b dec=%d',y='bin=%b dec=%d', result:'bin=%b dec=%d',
overflow=%d status=TRUE", xwire,xwire,ywire,ywire,sumout,sumout,v);
 end
 else
 begin
 $fdisplay(n,"operation is wrong");
 $display("operation is wrong");
 end
 #5;
end
$finish;
end

endmodule
```

COMPARISON

	4x8-bit 32-bit cond. sumadder	Single module 32-bit cond. Sum adder
Utilization	199 LUTs	107 LUTs
Max Clock Freq	**59.26MHz**	62.90 MHz
Average Power	24.734 Watt	26.293 Watt

In 4x8-bit implementation, there are almost %100 percent more lut cell used than single module 32-bit conditional sum adder.

Their maximum clock frequencies are close to each othe but single module 32-bit cont sum adder is better.

In power usage, 4x8 bit 32-bit cond. Sum adder uses less power than single module32-bit cond. sum adder.

Work package of this report:

Research	Muhammed Erkmen, Enes Şentürk
Understanding	Muhammed Erkmen, Enes Şentürk
Design	Muhammed Erkmen, Enes Şentürk
Code	Muhammed Erkmen, Enes Şentürk
Test	Muhammed Erkmen, Enes Şentürk
Report	Muhammed Erkmen, Enes Şentürk