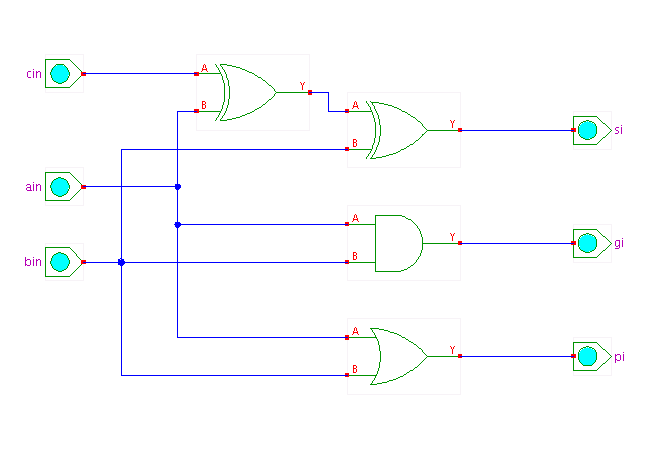
32-bit Carry Look Ahead Adder

Understanding the problem:

* Our goal is to design 32-bit Carry Look Ahead Adder which would improve the amount of time required to generate carry bits.
* In Ripple Carry Adder, a carry bit is calculated alongside the sum bit. So, each bit must wait until the previous carry has been calculated to begin calculating its own result and carry bits.
* The Carry Look Ahead adder calculates one or more carry bits before the sum, which reduces the wait time to calculate the result of the larger value bits.

Devising the plan:

* Initially, we design a 4-bit Carry Look Ahead Adder and then instantiate this design 8 times to extend it for 32 bits.
* We use Ripple Carry Adder to generate sum of 4-bits.We don’t use carry generated by this Ripple Carry Adder.
* We use separate combinational logic known as ‘Carry-Look-Ahead Logic’ to generate carry before the sum.
* For this purpose, we use concept of ‘Group generate’ and ‘Group propagate’.
* For two 1-bit inputs along with input carry bit , ‘generate’ and ‘propagate’ outputs are as follows:



* 4-bit group propagate output is high only when propagate output for each pair of bit is high.
* 4-bit group generate output is low when any of the following conditions is satisfied:

1. 4th pair generate output is low
2. 4th pair propagate output is high and 3rd pair generate output is low.
3. 4th and 3rd pair propagate output is high and 2nd pair generate output is low.
4. 4th, 3rd and 2nd pair propagate output is high and 1st pair generate output is low.

* Same is the case when 4-bit group generate output is high.
* Now, our job is to put these conditions in the form of Verilog code using Behavioral and Structural modeling.

Carrying out the plan:

* Low level module : 4-bit RCA with Carry-Look-Ahead logic
* Top level module : 32-bit Carry Look Ahead Adder
* Verilog code :

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

input a;

input b;

input c\_in;

output sum;

output c\_out;

assign sum=a^b^c\_in;

assign c\_out=(a&b)|(b&c\_in)|(a&c\_in);

endmodule

module adder1(a,b,c\_in,sum,c\_out);

input[3:0] a,b;

input c\_in;

output[3:0]sum;

output c\_out;

wire cA,cB,cC;

FA inst1(a[0],b[0],c\_in,sum[0],cA);

FA inst2(a[1],b[1],cA,sum[1],cB);

FA inst3(a[2],b[2],cB,sum[2],cC);

FA inst4(a[3],b[3],cC,sum[3],c\_out);

endmodule

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

input [3:0] a;

input [3:0] b;

input c\_in;

output [3:0]sum;

output c\_out;

wire g1;

reg p1;

wire y1;

adder1 inst\_1(a,b,c\_in,sum);

always @ (a,b)

begin

if(a[0]==~b[0] && a[1]==~b[1] && a[2]==~b[2] && a[3]==~b[3])

p1=1'b1;

else

p1=1'b0;

end

assign g1=(a[3]&b[3])+((a[3]^b[3])&(a[2]&b[2]))+((a[3]^b[3])&(a[2]^b[2])&(a[1]&b[1]))+((a[3]^b[3])&(a[2]^b[2])&(a[1]^b[1])&(a[0]&b[0]));

and(y1,p1,c\_in);

or(c\_out,g1,y1);

endmodule

module LAC(a,b,c\_in,sum,c\_outf);

input[31:0]a;

input[31:0]b;

input c\_in;

output[31:0]sum;

output c\_outf;

x x1(a[3:0],b[3:0],c\_in,sum[3:0],c\_out);

x x2(a[7:4],b[7:4],c\_out,sum[7:4],c\_out1);

x x3(a[11:8],b[11:8],c\_out1,sum[11:8],c\_out2);

x x4(a[15:12],b[15:12],c\_out2,sum[15:12],c\_out3);

x x5(a[19:16],b[19:16],c\_out3,sum[19:16],c\_out4);

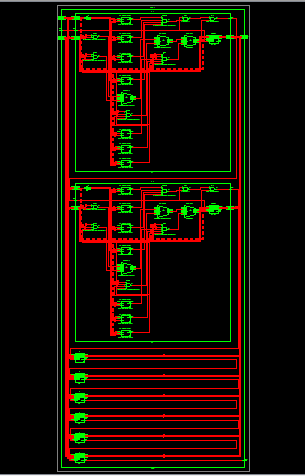
x x6(a[23:20],b[23:20],c\_out4,sum[23:20],c\_out5);

x x7(a[27:24],b[27:24],c\_out5,sum[27:24],c\_out6);

x x8(a[31:28],b[31:28],c\_out6,sum[31:28],c\_outf);

endmodule

RTL Schematic :



Test Bench:

module LAC\_tb;

// Inputs

reg [31:0] a;

reg [31:0] b;

reg c\_in;

// Outputs

wire [31:0] sum;

wire c\_outf;

// Instantiate the Unit Under Test (UUT)

LAC uut (

.a(a),

.b(b),

.c\_in(c\_in),

.sum(sum),

.c\_outf(c\_outf)

);

initial begin

// Initialize Inputs

a = 0;

b = 0;

c\_in = 0;

// Wait 100 ns for global reset to finish

#100;

a=32'b10100011010010011100011010101111;

#100;

b=32'b10101110101011100000111110111010;

#100;

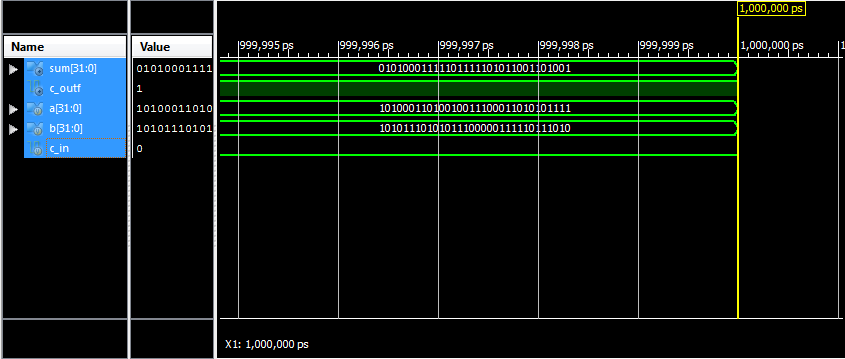
c\_in=1'b0;

// Add stimulus here

end

endmodule

Output:



Looking Back:

* Output carry and sum of two 32-bit inputs is generated correctly.
* Combinational path delay is found to be less than 32-bit RCA. It concludes that Carry Look Ahead Adder is faster than 32-bit RCA.
* But, area required for the design is more.

Synthesis report:

Design Statistics:

# IOs : 98

Cell Usage :

# BELS : 138

# GND : 1

# LUT2 : 32

# LUT3 : 23

# LUT4 : 73

# MUXF5 : 9

# IO Buffers : 98

# IBUF : 65

# OBUF : 33

Device utilization summary:

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

Selected Device : 3s200pq208-5

Number of Slices: 73 out of 1920 3%

Number of 4 input LUTs: 128 out of 3840 3%

Number of IOs: 98

Number of bonded IOBs: 98 out of 141 69%

Timing Summary:

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

Speed Grade: -5

Minimum period: No path found

Minimum input arrival time before clock: No path found

Maximum output required time after clock: No path found

Maximum combinational path delay: 21.183ns (10.954ns logic, 10.229ns route)

Data Path: a<1> to sum<31>

Gate Net

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

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

IBUF:I->O 5 0.715 1.078 a\_1\_IBUF (a\_1\_IBUF)

LUT2:I0->O 2 0.479 1.040 x1/inst\_1/inst2/Mxor\_sum\_xo<0>11 (N38)

LUT4:I0->O 1 0.479 0.704 x1/c\_out\_SW0 (N77)

LUT4:I3->O 4 0.479 0.838 x1/c\_out (c\_out)

LUT4:I2->O 4 0.479 0.838 x2/c\_out (c\_out1)

LUT4:I2->O 4 0.479 0.838 x3/c\_out (c\_out2)

LUT4:I2->O 4 0.479 0.838 x4/c\_out (c\_out3)

LUT4:I2->O 4 0.479 0.838 x5/c\_out (c\_out4)

LUT4:I2->O 4 0.479 0.838 x6/c\_out (c\_out5)

LUT4:I2->O 4 0.479 0.779 x7/c\_out (c\_out6)

MUXF5:S->O 2 0.540 0.915 x8/inst\_1/inst2/c\_out\_f5 (x8/inst\_1/cB)

LUT4:I1->O 1 0.479 0.681 x8/inst\_1/inst4/Mxor\_sum\_xo<0>1 (sum\_31\_OBUF)

OBUF:I->O 4.909 sum\_31\_OBUF (sum<31>)

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

Total **21.183ns** (10.954ns logic, 10.229ns route)

(51.7% logic, 48.3% route)