

Final Project Report

ECE465

Dominick Przybylo
Friday May 10, 2019
UIN: 271826267

Introduction

32-bit Ripple Carry Adder(RCA32)

A full adder adds binary numbers and accounts for the values that are carried in as well as out. Full adder adds 3 inputs which are A, B, and Cin. The circuit produces 2 outputs which are Sum and Cout. Sum can be found using $\text{Sum} = A \oplus B \oplus \text{Cin}$. Cout can be found using $\text{Cout} = (A \& B) \vee (B \& \text{Cin}) \vee (\text{Cin} \& A)$.

In order to get the 32-bit full adder I implemented a 1-bit full adder 32 times and I got a 32-bit full adder.

32-bit Carry Lookahead Adder(CLA32)

A carry lookahead adder was created to have a faster way of adding two binary numbers. The way it works is two signals(propagate and generate) are created for each bit position, based on whether a carry is propagated through a less significant bit position(one input is at least 1, generated in that bit position (both inputs are 1)or killed in that bit position(both inputs are 0). After P and G are generated then the carries for every bit position are created.

In order to get the 32-bit carry lookahead adder I implemented a 4-bit carry lookahead adder 8 times and that will give me the 32-bit carry lookahead adder.

32-bit Multiplier(MULTIPLIER)

A shift and add multiplier starts off with two inputs a multiplier and a multiplicand. It doesn't matter how many bits but in this we want 32 bits so both inputs would be 16-bits each. The multiplier will then calculate the result using a shift and add method. As a result you should have 32-bit product.


In order to get the 32 bit multiplier, I implemented CLA adder with two inputs which then I used it for the multiplier to get the 32-bit.

Resources

RCA32

Sample 8-bit full adder code

```
module Vadder8(A, B, CIN, S, COUT);  
input [7:0] A, B;  
input CIN;  
output [7:0] S;  
output COUT;  
assign {COUT, S} = A + B + CIN;  
endmodule
```



As we were told not to use `assign {Cout, Sum} = A+B+Cin` in our code I did some research and about the full adder and found $\text{Sum} = A \oplus B \oplus \text{Cin}$ and $\text{Cout} = (A \& B) | (B \& \text{Cin}) | (\text{Cin} \& A)$. From this I have come up with the following code for 1-bit full adder

```
// simple 1bit full adder  
module fulladder1(A, B, Cin, Sum, Cout);  
  
// inputs  
input A;  
input B;  
input Cin;  
  
//outputs  
output Sum;  
output Cout;  
  
//calculating Sum and Cout  
assign Sum = A ^ B ^ Cin;  
assign Cout = (A & B) | (A & Cin) | (B & Cin);  
  
endmodule
```

Now with having a 1-bit full adder I have used that to create a 32-bit full adder.
Sample code of a 16-bit full adder

```
module RCA16(S, Cout, A, B, Cin);
  input [15:0] A, B;
  input Cin;
  output [15:0] S;
  output Cout;

  wire[14:0] c;

  _full_adder FA1(S[0],c[0],A[0],B[0],Cin);
  _full_adder FA2(S[1],c[1],A[1],B[1],c[0]);
  _full_adder FA3(S[2],c[2],A[2],B[2],c[1]);
  _full_adder FA4(S[3],c[3],A[3],B[3],c[2]);
  _full_adder FA5(S[4],c[4],A[4],B[4],c[3]);
  _full_adder FA6(S[5],c[5],A[5],B[5],c[4]);
  _full_adder FA7(S[6],c[6],A[6],B[6],c[5]);
  _full_adder FA8(S[7],c[7],A[7],B[7],c[6]);
  _full_adder FA9(S[8],c[8],A[8],B[8],c[7]);
  _full_adder FA10(S[9],c[9],A[9],B[9],c[8]);
  _full_adder FA11(S[10],c[10],A[10],B[10],c[9]);
  _full_adder FA12(S[11],c[11],A[11],B[11],c[10]);
  _full_adder FA13(S[12],c[12],A[12],B[12],c[11]);
  _full_adder FA14(S[13],c[13],A[13],B[13],c[12]);
  _full_adder FA15(S[14],c[14],A[14],B[14],c[13]);
  _full_adder FA16(S[15],Cout,A[15],B[15],c[14]);

endmodule
```

As this sample code above is for 16-bit I used it as an outline and I used my 1-bit fulladder 32 times and came up with the code below.

RCA 32bit

// implementing 1bit full adder into 32bit fulladder
module fulladder32(A, B, Cin, Sum, Cout);

//inputs
input [31:0] A;
input [31:0] B;
input Cin;

//outputs
output [31:0] Sum;
output Cout;

// wires connecting the full adders
wire [30:0] c;

```
fulladder1 fa0(A[0], B[0]), Cin, Sum[0], c[0];
fulladder1 fa1(A[1], B[1]), c[0], Sum[1], c[1];
fulladder1 fa2(A[2], B[2]), c[1], Sum[2], c[2];
fulladder1 fa3(A[3], B[3]), c[2], Sum[3], c[3];
fulladder1 fa4(A[4], B[4]), c[3], Sum[4], c[4];
fulladder1 fa5(A[5], B[5]), c[4], Sum[5], c[5];
fulladder1 fa6(A[6], B[6]), c[5], Sum[6], c[6];
fulladder1 fa7(A[7], B[7]), c[6], Sum[7], c[7];
fulladder1 fa8(A[8], B[8]), c[7], Sum[8], c[8];
fulladder1 fa9(A[9], B[9]), c[8], Sum[9], c[9];
fulladder1 fa10(A[10], B[10]), c[9], Sum[10], c[10];
fulladder1 fa11(A[11], B[11]), c[10], Sum[11], c[11];
fulladder1 fa12(A[12], B[12]), c[11], Sum[12], c[12];
fulladder1 fa13(A[13], B[13]), c[12], Sum[13], c[13];
fulladder1 fa14(A[14], B[14]), c[13], Sum[14], c[14];
```

```

fulladder1 fa15(A[15], B[15]), c[14], Sum[15], c[15]);
fulladder1 fa16(A[16], B[16]), c[15], Sum[16], c[16]);
fulladder1 fa17(A[17], B[17]), c[16], Sum[17], c[17]);
fulladder1 fa18(A[18], B[18]), c[17], Sum[18], c[18]);
fulladder1 fa19(A[19], B[19]), c[18], Sum[19], c[19]);
fulladder1 fa20(A[20], B[20]), c[19], Sum[20], c[20]);
fulladder1 fa21(A[21], B[21]), c[20], Sum[21], c[21]);
fulladder1 fa22(A[22], B[22]), c[21], Sum[22], c[22]);
fulladder1 fa23(A[23], B[23]), c[22], Sum[23], c[23]);
fulladder1 fa24(A[24], B[24]), c[23], Sum[24], c[24]);
fulladder1 fa25(A[25], B[25]), c[24], Sum[25], c[25]);
fulladder1 fa26(A[26], B[26]), c[25], Sum[26], c[26]);
fulladder1 fa27(A[27], B[27]), c[26], Sum[27], c[27]);
fulladder1 fa28(A[28], B[28]), c[27], Sum[28], c[28]);
fulladder1 fa29(A[29], B[29]), c[28], Sum[29], c[29]);
fulladder1 fa30(A[30], B[30]), c[29], Sum[30], c[30]);
fulladder1 fa31(A[31], B[31]), c[30], Sum[31], Cout);
endmodule

```

CLA32

Sample code of 4-bit carry lookahead adder.

```

module CLA_Adder(a,b,cin,sum,cout);
    input [3:0] a,b;
    input cin;
    output [3:0] sum;
    output cout;
    wire p0,p1,p2,p3,g0,g1,g2,g3,c1,c2,c3,c4;
    assign p0=(a[0]^b[0]),
           p1=(a[1]^b[1]),
           p2=(a[2]^b[2]),
           p3=(a[3]^b[3]);
    assign g0=(a[0]&b[0]),
           g1=(a[1]&b[1]),
           g2=(a[2]&b[2]),
           g3=(a[3]&b[3]);
    assign c0=cin,
           c1=g0|(p0&cin),
           c2=g1|(p1&g0)|(p1&p0&cin),
           c3=g2|(p2&g1)|(p2&p1&g0)|(p1&p1&p0&cin),
           c4=g3|(p3&g2)|(p3&p2&g1)|(p3&p2&p1&g0)|(p3&p2&p1&p0&cin);
    assign sum[0]=p0^c0,
           sum[1]=p1^c1,
           sum[2]=p2^c2,
           sum[3]=p3^c3;
    assign cout=c4;
endmodule

```

From this code above I used it as an outline to recreate a 4-bit carry lookahead adder for what I will be using to implement that for the 32-bit carry lookahead.

This is my 4-bit carry lookahead adder below.

```

// 4-bit carry lookahead adder
module carrylook4(A, B, Cin, Sum, Cout);

```

```

//inputs
input [3:0] A;
input [3:0] B;
input Cin;

//outputs
output [3:0] Sum;
output Cout;

// wires for propagate, generate, and carry
wire [3:0] P, G, C;

// calculating
assign P = A ^ B; //propagate
assign G = A & B; //generate

assign C[0] = Cin;
assign C[1] = G[0] | (P[0] & C[0]);
assign C[2] = G[1] | (P[1] & G[0]) | (P[1] & P[0] & C[0]);
assign C[3] = G[2] | (P[2] & G[1]) | (P[2] & P[1] & G[0]) | (P[2] & P[1] & P[0] & C[0]);
assign Cout = G[3] | (P[3] & G[2]) | (P[3] & P[2] & G[1]) | (P[3] & P[2] & P[1] & P[0] & C[0]);
assign Sum = P ^ C;

endmodule

```

Now that I have my 4-bit cla I came up with the idea of using the 4-bit cla 8 times which will make a 32-bit cla.

Sample 16-bit cla code

```

module carry_look_ahead_16bit(a,b, cin, sum,cout);
input [15:0] a,b;
input cin;
output [15:0] sum;
output cout;
wire c1,c2,c3;

carry_look_ahead_4bit cla1 (.a(a[3:0]), .b(b[3:0]), .cin(cin), .sum(sum[3:0]), .cout(c1));
carry_look_ahead_4bit cla2 (.a(a[7:4]), .b(b[7:4]), .cin(c1), .sum(sum[7:4]), .cout(c2));
carry_look_ahead_4bit cla3(.a(a[11:8]), .b(b[11:8]), .cin(c2), .sum(sum[11:8]), .cout(c3));
carry_look_ahead_4bit cla4(.a(a[15:12]), .b(b[15:12]), .cin(c3), .sum(sum[15:12]), .cout(cout));

endmodule

```

With the above code I used it as an outline to implement the 4-bit cla for 32-bit cla.

This is my 32-bit cla code.

```

// carrylook4 implemented for the carrylook32
module carrylook32(A, B, Cin, Sum, Cout);

```

```

//inputs
input [31:0] A;
input [31:0] B;
input Cin;

//outputs
output [31:0] Sum;
output Cout;

// wires used to connect all the adders
wire C3, C7, C11, C15, C19, C23, C27;

// implementation of carrylook4 into carrylook32
carrylook4 cla1(.A(A[3:0]), .B(B[3:0]), .Cin(Cin), .Sum(Sum[3:0]), .Cout(C3));
carrylook4 cla2(.A(A[7:4]), .B(B[7:4]), .Cin(C3), .Sum(Sum[7:4]), .Cout(C7));
carrylook4 cla3(.A(A[11:8]), .B(B[11:8]), .Cin(C7), .Sum(Sum[11:8]), .Cout(C11));
carrylook4 cla4(.A(A[15:12]), .B(B[15:12]), .Cin(C11), .Sum(Sum[15:12]), .Cout(C15));
carrylook4 cla5(.A(A[19:16]), .B(B[19:16]), .Cin(C15), .Sum(Sum[19:16]), .Cout(C19));
carrylook4 cla6(.A(A[23:20]), .B(B[23:20]), .Cin(C19), .Sum(Sum[23:20]), .Cout(C23));
carrylook4 cla7(.A(A[27:24]), .B(B[27:24]), .Cin(C23), .Sum(Sum[27:24]), .Cout(C27));
carrylook4 cla8(.A(A[31:28]), .B(B[31:28]), .Cin(C27), .Sum(Sum[31:28]), .Cout(Cout));

endmodule

```

Multiplier(cla_adder)

For the multiplier I decided to use cla adder and have that be used for the multiplier.
Below is a same code for the cla.

```

Module cpu_wb_cla_adder(in1,in2,carry_in,sum, carry_out);
parameter DATA_WID = 32;

input [DATA_WID - 1:0] in1;
input [DATA_WID - 1:0] in2;
input carry_in;
output [DATA_WID - 1:0] sum;
output carry_out;

wire [DATA_WID - 1:0] gen;
wire [DATA_WID - 1:0] pro;
wire [DATA_WID:0] carry_tmp;

genvar j, i;
generate

assign carry_tmp[0] = carry_in;

//carry generator
for(j = 0; j < DATA_WID; j = j + 1) begin: carry_generator

```

```

assign gen[j] = in1[j] & in2[j];
assign pro[j] = in1[j] | in2[j];
assign carry_tmp[j+1] = gen[j] | pro[j] & carry_tmp[j];
end
//carry out
assign carry_out = carry_tmp[DATA_WID];
//calculate sum
//assign sum[0] = in1[0] ^ in2 ^ carry_in;
for(i = 0; i < DATA_WID; i = i+1) begin: sum_without_carry
assign sum[i] = in1[i] ^ in2[i] ^ carry_tmp[i];
end
endgenerate
endmodule

```

Using this above code as an outline I came up with the code below.

```

// cla adder
module cla_adder (A, B, Cin, Sum, Cout);
parameter DATA_WID = 32;

//inputs
input [DATA_WID - 1:0] A;
input [DATA_WID - 1:0] B;
input Cin;

//outputs
output [DATA_WID - 1:0] Sum;
output Cout;

wire [DATA_WID - 1:0] G;// generate
wire [DATA_WID - 1:0] P;// propagate
wire [DATA_WID:0] C_temp;// temp value for carry

genvar i, j;
generate
//assume carry_tmp in is zero
assign C_temp[0] = Cin;

//carry generator
for(j = 0; j < DATA_WID; j = j + 1) begin: carry_generator
assign G[j] = A[j] & B[j];
assign P[j] = A[j] | B[j];
assign C_temp[j+1] = G[j] | P[j] & C_temp[j];
end

//carry out
assign Cout = C_temp[DATA_WID];

//calculate sum
for(i = 0; i < DATA_WID; i = i+1) begin: sum_without_carry
assign Sum[i] = A[i] ^ B[i] ^ C_temp[i];
end
endgenerate
endmodule

```


Multiplier(Multiplier)

For the multiplier I will use the cla_adder and have that being multiplied together. Same code below.

```
module cpu_wb_cla_multiplier (multicand, multiplier, product);
parameter MULTICAND_WID = 32;
parameter MULTIPLIER_WID = 32;

input [MULTICAND_WID-1:0] multicand;
input [MULTIPLIER_WID-1:0] multiplier;
output [(MULTICAND_WID + MULTIPLIER_WID - 1):0] product;

wire [MULTICAND_WID - 1:0] multicand_tmp [MULTIPLIER_WID-1:0];
wire [MULTICAND_WID - 1:0] product_tmp [MULTIPLIER_WID-1:0];
wire [MULTIPLIER_WID -1:0] carry_tmp;

generate
//initialize values
for(j = 0; j < MULTIPLIER_WID; j = j + 1) begin: for_loop_j
assign multicand_tmp[j] = multicand & {MULTICAND_WID{multiplier[j]}};
end

assign product_tmp[0] = multicand_tmp[0];
assign carry_tmp[0] = 1'b0;
assign product[0] = product_tmp[0][0];

for(i = 1; i < MULTIPLIER_WID; i = i + 1) begin: for_loop_i
cpu_wb_cla_adder #(.DATA_WID(MULTIPLIER_WID)) add1 (
// Outputs
.sum(product_tmp[i]),
.carry_out(carry_tmp[i]),
// Inputs
.carry_in(1'b0),
.in1(multicand_tmp[i]),
.in2({carry_tmp[i-1],product_tmp[i-1][31-:31]}));
assign product[i] = product_tmp[i][0];
end //end for loop
assign product[(MULTIPLIER_WID+MULTIPLIER_WID-1):MULTIPLIER_WID] =
{carry_tmp[MULTIPLIER_WID-1],product_tmp[MULTIPLIER_WID-1][31-:31]};
endgenerate
endmodule
```

Using this as an outline I have came up with my code below.

```
module cla_multiplier (MULTIPLICAND, MULTIPLIER, PRODUCT);
parameter MULTIPLICAND_WID = 32;
parameter MULTIPLIER_WID = 32;

//inputs
input [MULTIPLICAND_WID-1:0] MULTIPLICAND;
input [MULTIPLIER_WID-1:0] MULTIPLIER;
```

```

//output
output [(MULTIPLICAND_WID + MULTIPLIER_WID - 1):0] PRODUCT;

wire [MULTIPLICAND_WID - 1:0] MULTIPLICAND_temp [MULTIPLIER_WID-1:0];
wire [MULTIPLICAND_WID - 1:0] PRODUCT_temp [MULTIPLIER_WID-1:0];
wire [MULTIPLIER_WID -1:0] C_temp;

genvar i, j;
generate
//initialize values
for(j = 0; j < MULTIPLIER_WID; j = j + 1) begin: for_loop_j
assign MULTIPLICAND_temp[j] = MULTIPLICAND & {MULTIPLICAND_WID{MULTIPLIER[j]}};
end

assign PRODUCT_temp[0] = MULTIPLICAND_temp[0];
assign C_temp[0] = 1'b0;
assign PRODUCT[0] = PRODUCT_temp[0][0];

for(i = 1; i < MULTIPLIER_WID; i = i + 1) begin: for_loop_i
cla_adder #(.DATA_WID(MULTIPLIER_WID)) add1 (
// Outputs
.Sum(PRODUCT_temp[i]),
.Cout(C_temp[i]),
// Inputs
.Cin(1'b0),
.A(MULTIPLICAND_temp[i]),
.B({C_temp[i-1],PRODUCT_temp[i-1][31 -:31]}));
assign PRODUCT[i] = PRODUCT_temp[i][0];
end //end for loop

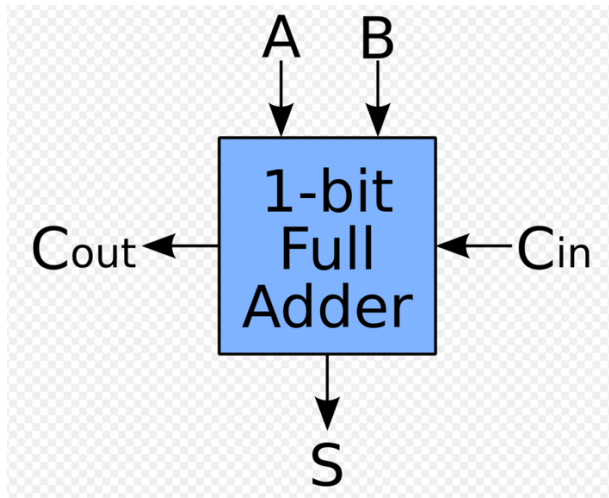
assign          PRODUCT[(MULTIPLIER_WID+MULTIPLIER_WID-1):MULTIPLIER_WID] =
{C_temp[MULTIPLIER_WID-1],PRODUCT_temp[MULTIPLIER_WID-1][31 -:31]};

endgenerate
endmodule

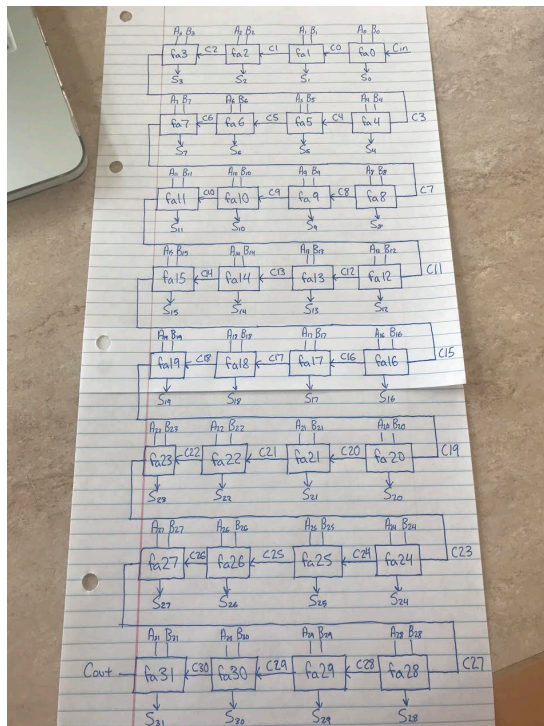
```

Structures for RCA and CLA

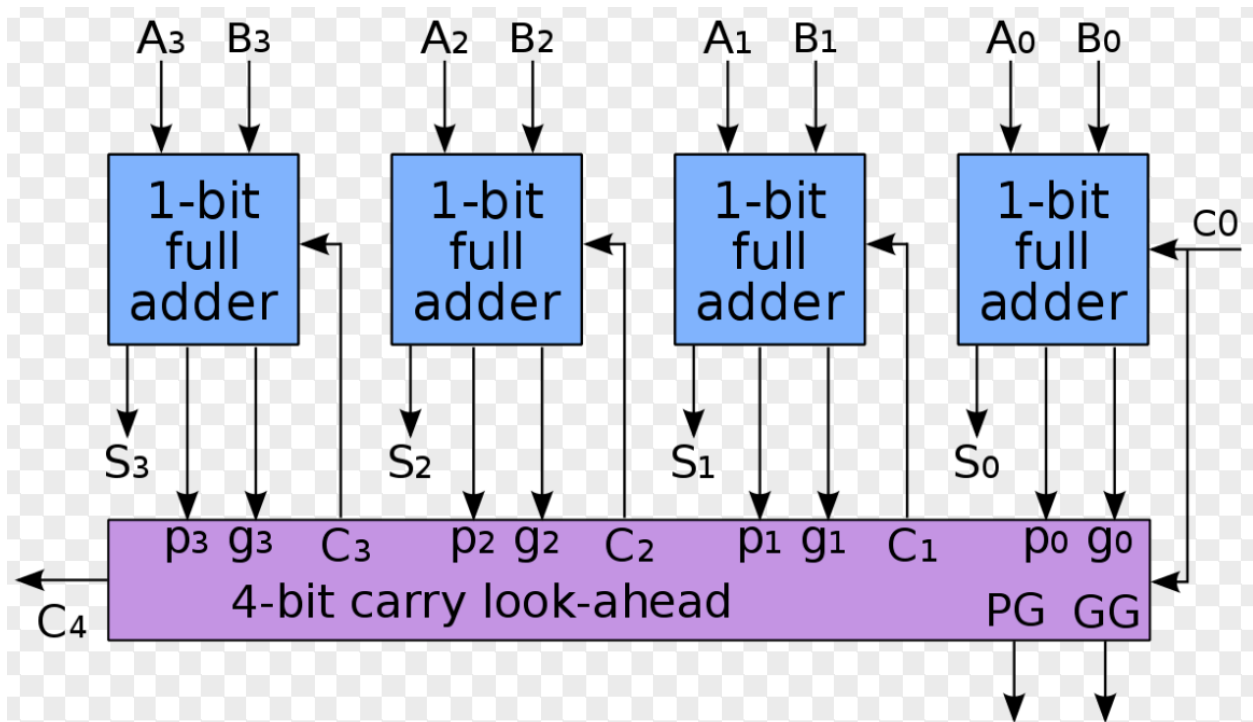
1 bit RCA and 32 bit RCA



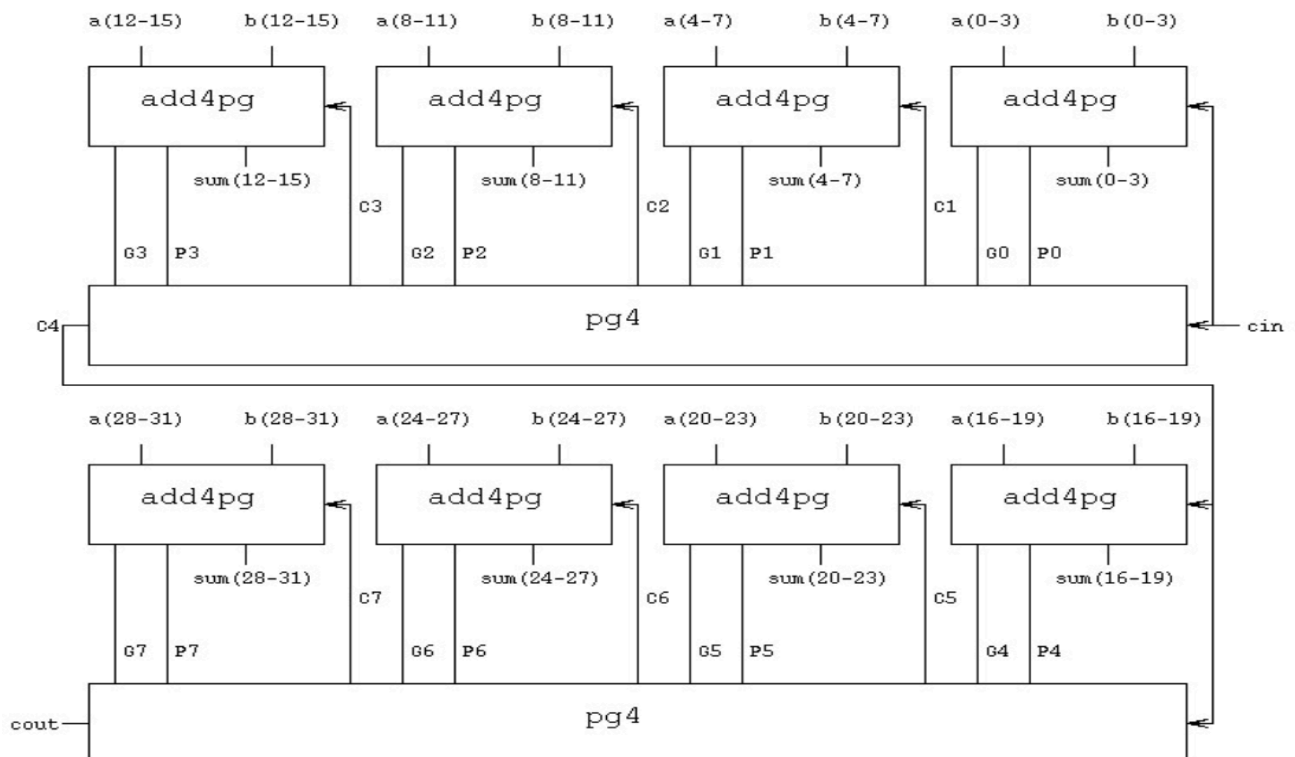
I used 32 1 bit full adders for my 32 bit full adder structure below.



4 bit CLA and 32 bit CLA

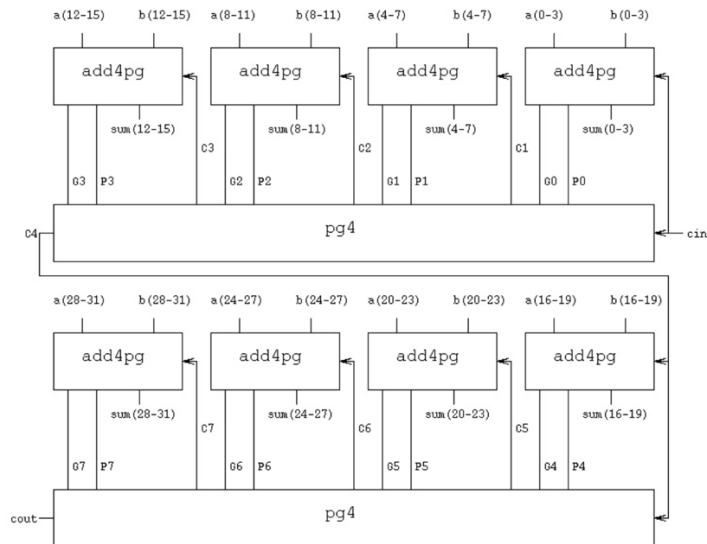


I used 8 4 bit cla to design my 32 bit cla below.

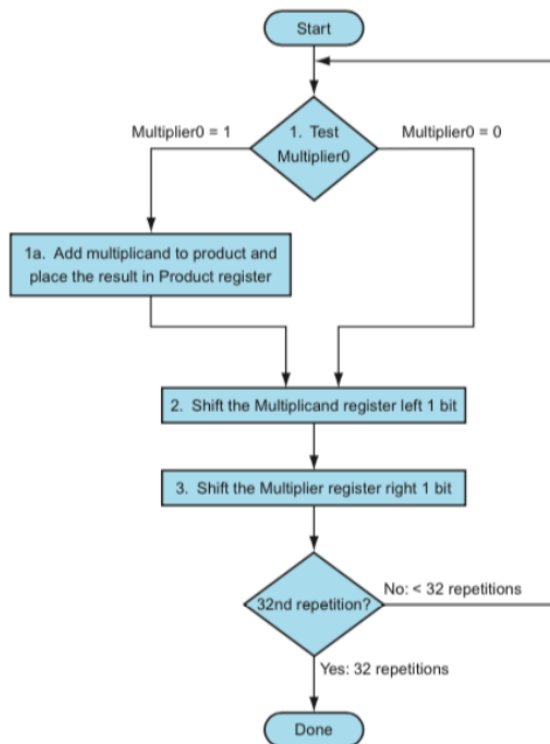


Multiplier

I chose to use a 32 bit carry lookahead adder to use for the multiplier



For the multiplier down below is the structure.

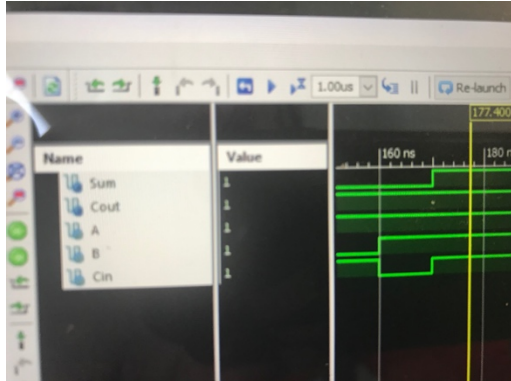


Simulations and Delay

1 bit RCA

According to my testbench down below are my simulations for 1 bit full adder.





Time Analysis

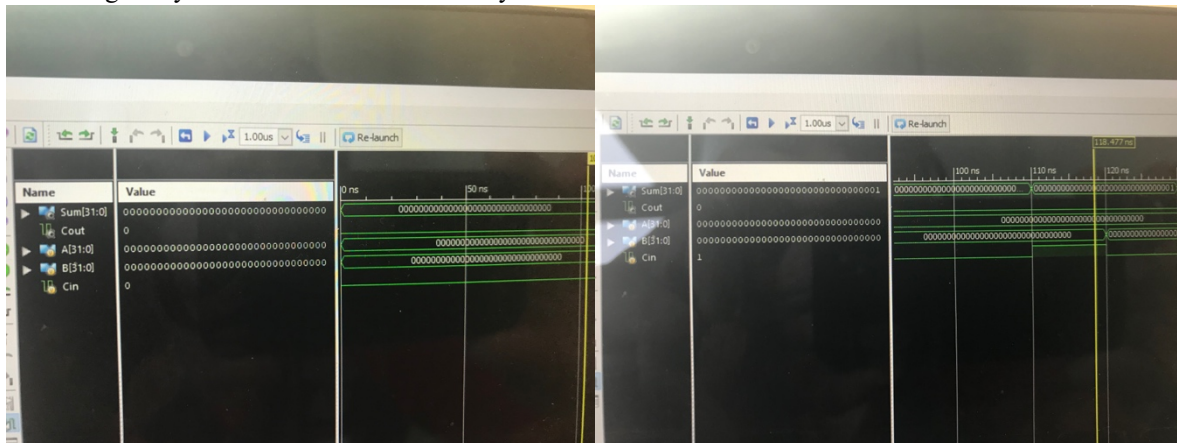
fulladder1

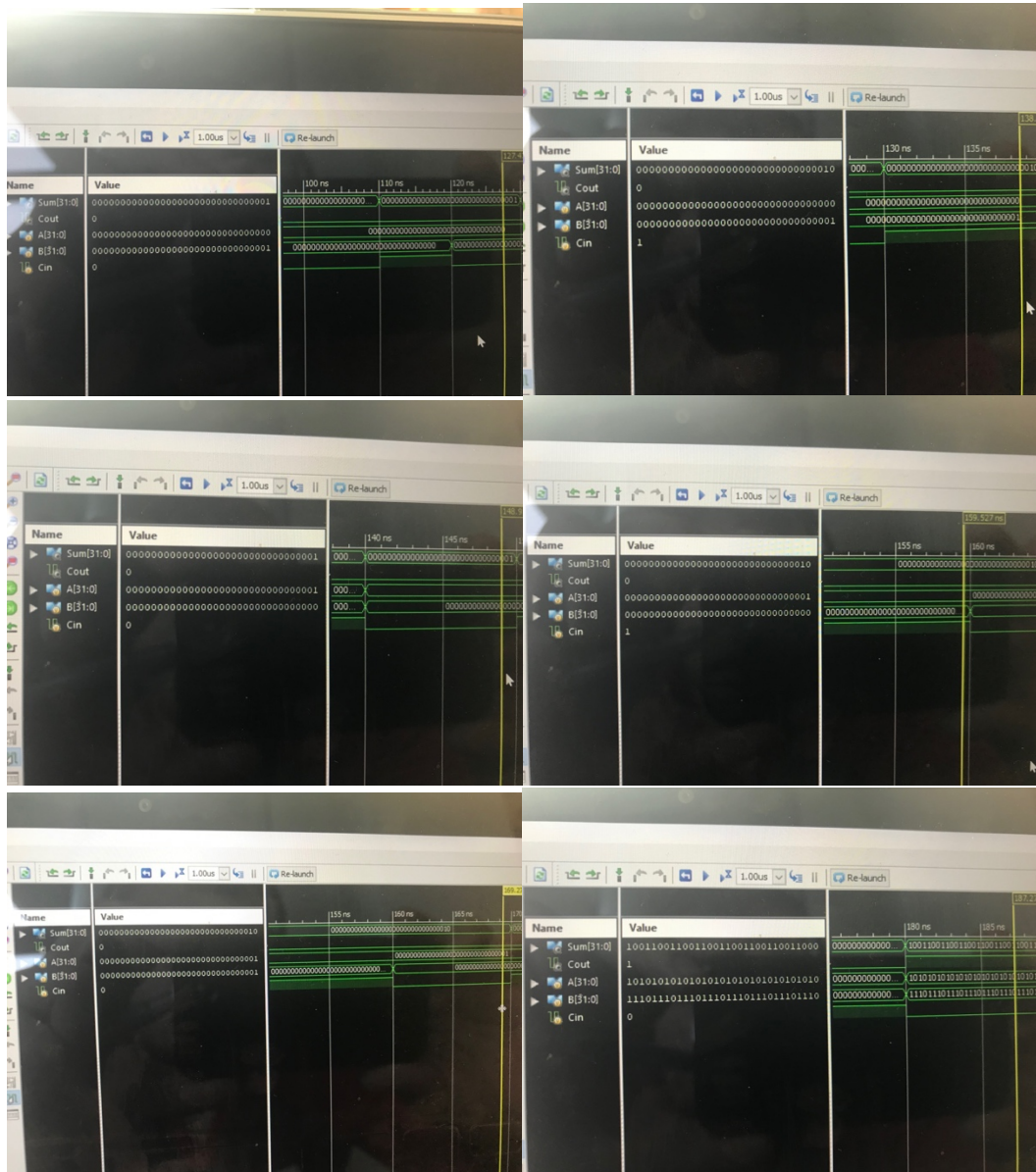
Cin | Cout | 8.302|

Time delay between Cin and Cout is 8.302 ns.

32 bit RCA

According to my testbench down below are my simulations for 32 full adder.





Time Analysis

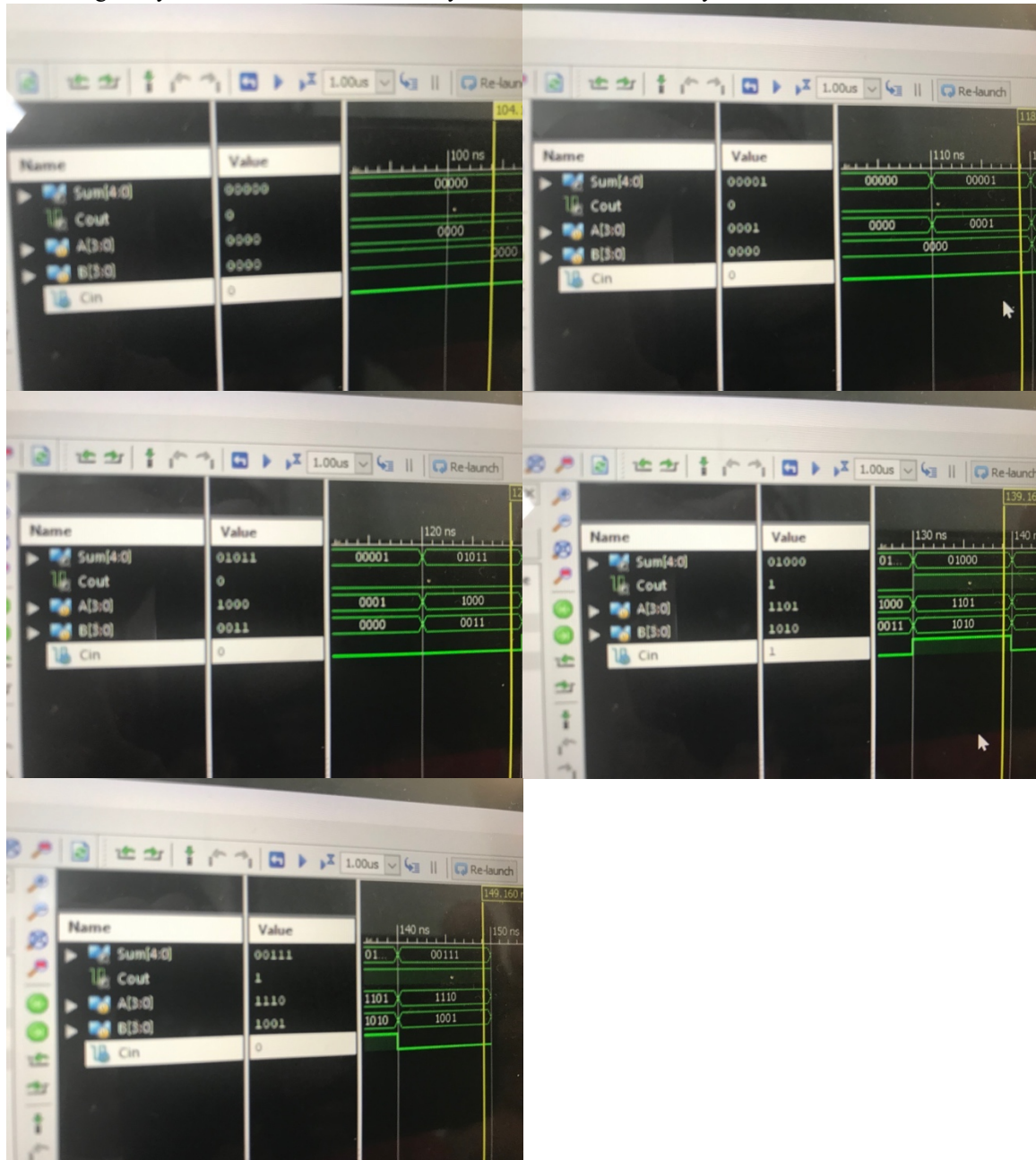
32 bit full adder

Cin	Cout	27.716
-----	------	--------

Time delay between Cin and Cout is 27.716 ns.

4 bit CLA

According to my testbench down below are my simulations for 4 bit carry lookahead adder.



Time Analysis

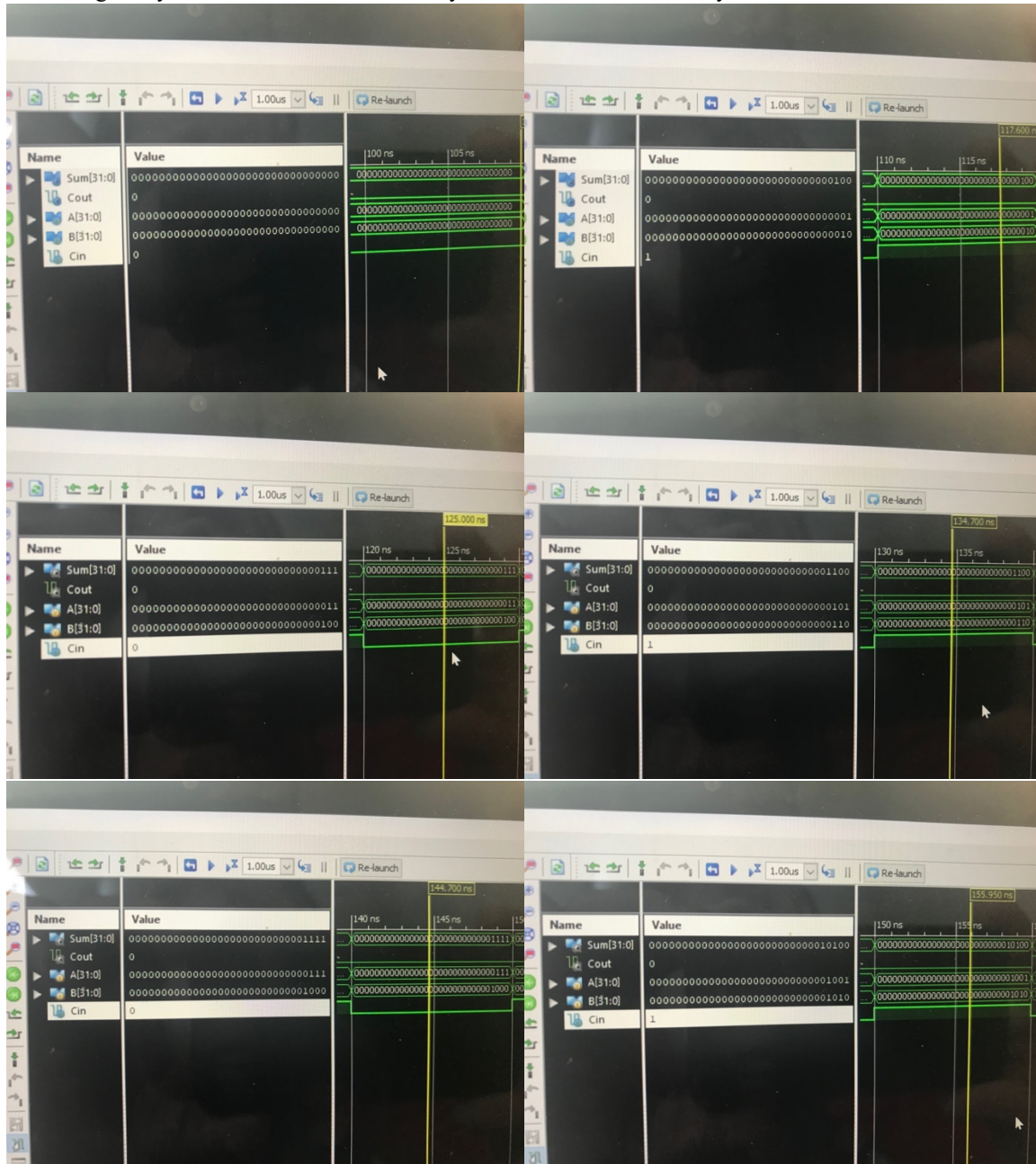
4 bit carry lookahead adder

Cin |Cout | 8.635|

Time delay between Cin and Cout is 8.635 ns.

32 bit CLA

According to my testbench down below are my simulations for 32 bit carry lookahead adder.



Time Analysis

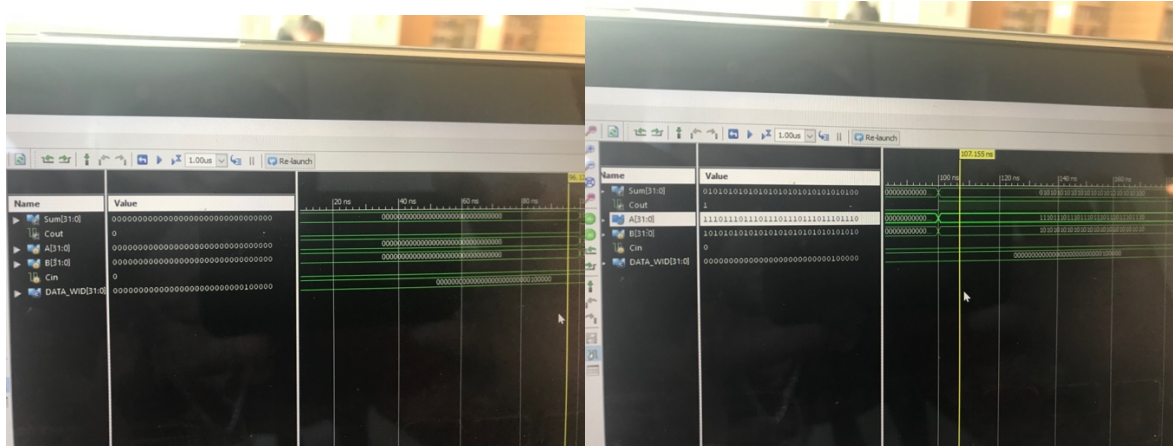
32 bit carry lookahead adder

Cin | Cout | 27.716|

Time delay between Cin and Cout is 27.716 ns.

32 bit CLA

I used a different cla for the multiplier. According to my testbench down below are my simulations of the 32 bit carry lookahead adder.



Time Analysis

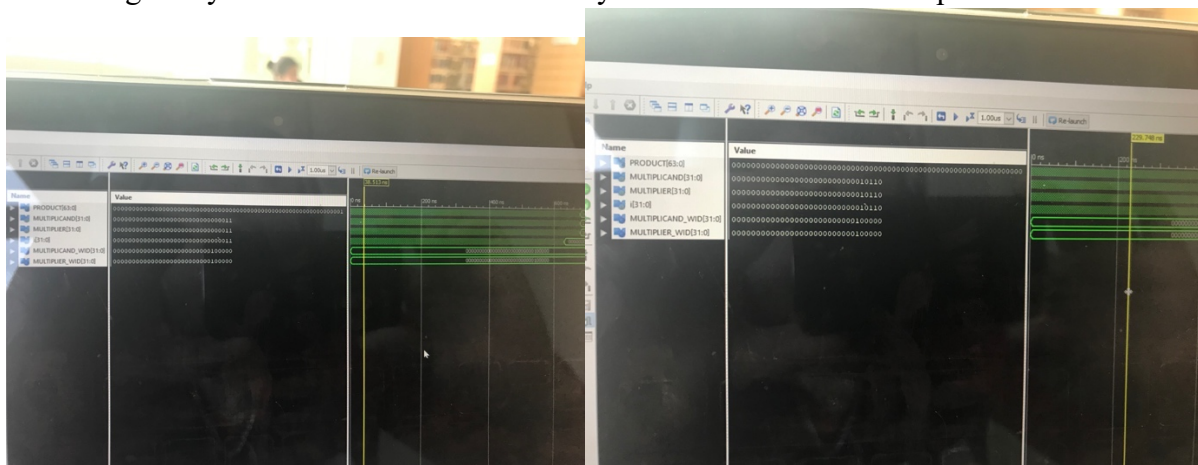
32 bit carry lookahead adder

Cin | Cout | 31.032|

Time delay between Cin and Cout is 31.032 ns.

Multiplier

According to my testbench down below are my simulations for the multiplier.



Time Analysis

Multiplier

MULTIPLICAND<31>	PRODUCT<63>	27.207
MULTIPLIER<31>	PRODUCT<63>	42.233

Time delay between the last Multiplicand and Product is 27.207 ns.
Time delay between the last Multiplier and Product is 42.233 ns.