جامعة بيرزيت

BIRZEIT UNIVERSITY

Faculty of Engineering & Technology

Electrical & Computer Engineering Department

Advanced Digital System Design ENCS3310

Project Report

Prepared by:

Mohammad Abu Shams 1200549

Instructor: Dr. Abdallatif Abuissa

Section: 2

Date: 22-8-2022

Table of Contents

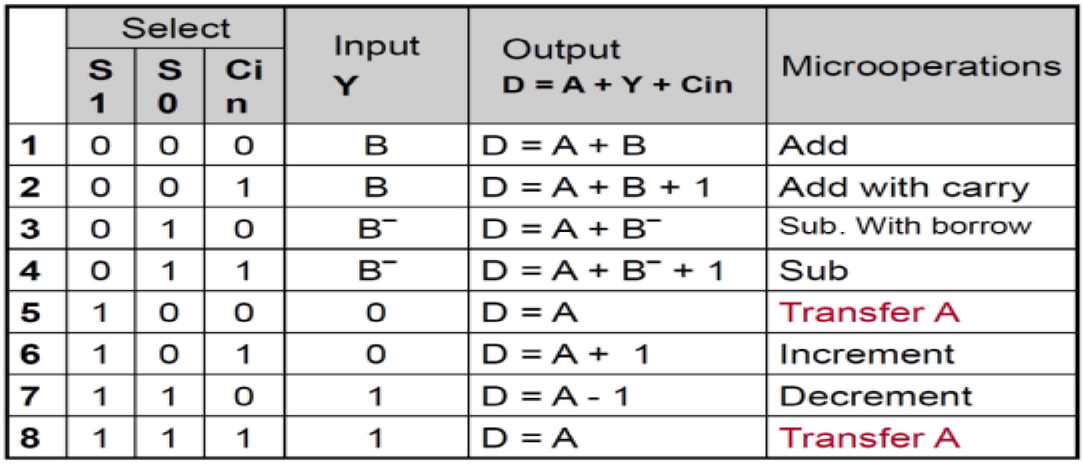
List of Figures

List of Tables

Brief introduction and background

In this project, we will design and implement an Arithmetic unit. The inputs and the output of the Arithmetic Unit are fed through/to flip-flops (registers). The Arithmetic unit works as shown in the following table:

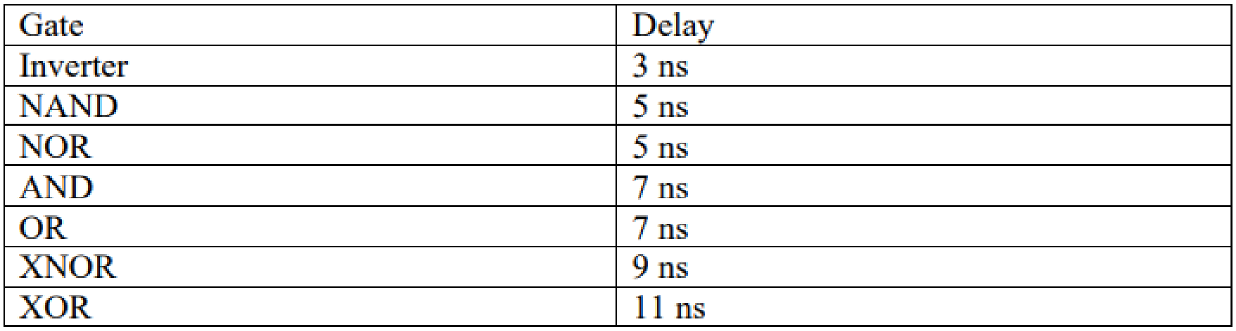
Table 1: Arithmetic unit table



	Select			Input Y	Output D = A + Y + Cin	Microoperations
	S 1	S 0	Ci n			
1	0	0	0	B	D = A + B	Add
2	0	0	1	B	D = A + B + 1	Add with carry
3	0	1	0	B^-	D = A + B^-	Sub. With borrow
4	0	1	1	B^-	D = A + B^- + 1	Sub
5	1	0	0	0	D = A	Transfer A
6	1	0	1	0	D = A + 1	Increment
7	1	1	0	1	D = A - 1	Decrement
8	1	1	1	1	D = A	Transfer A

First, in stage 1, we built 1-bit full adder from basic gates, then we built 4-bit adder by calling 1-bit full adder 4 times, and we built mux 4x1 also from basic gates, after that we called them 4 times and built the system structurally with basic gates, and we do a verifications to the system that will discover if I had an error, and the delay of gates is:

Table 2: Delay of gates



Gate	Delay
Inverter	3 ns
NAND	5 ns
NOR	5 ns
AND	7 ns
OR	7 ns
XNOR	9 ns
XOR	11 ns

Second, in stage 2**,** we replace the ripple adder by look-ahead adder, firstly we built look-ahead adder using loops, and the multiplexers that we built before, we invoke it 4 times and built the system structurally by basic gates, and also we do a verification.

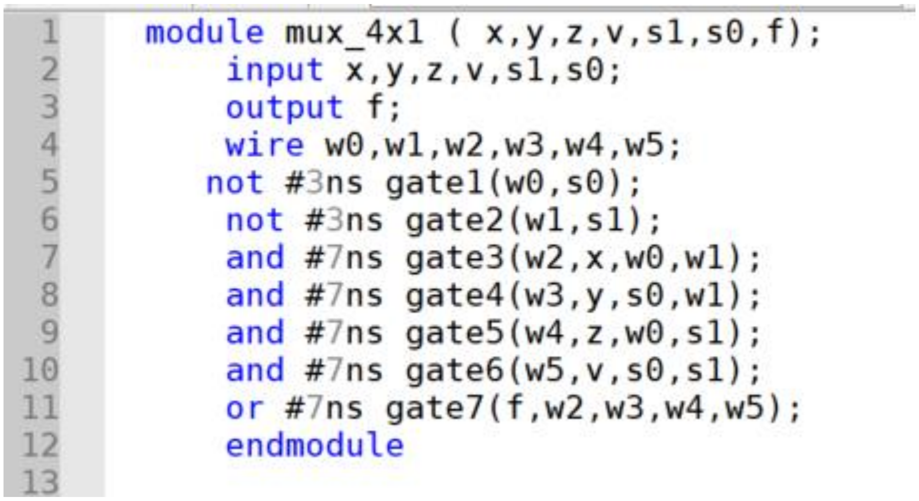
We use active-HDL student edition to write my codes and simulates them and tracking the output.

Design philosophy

Stage1

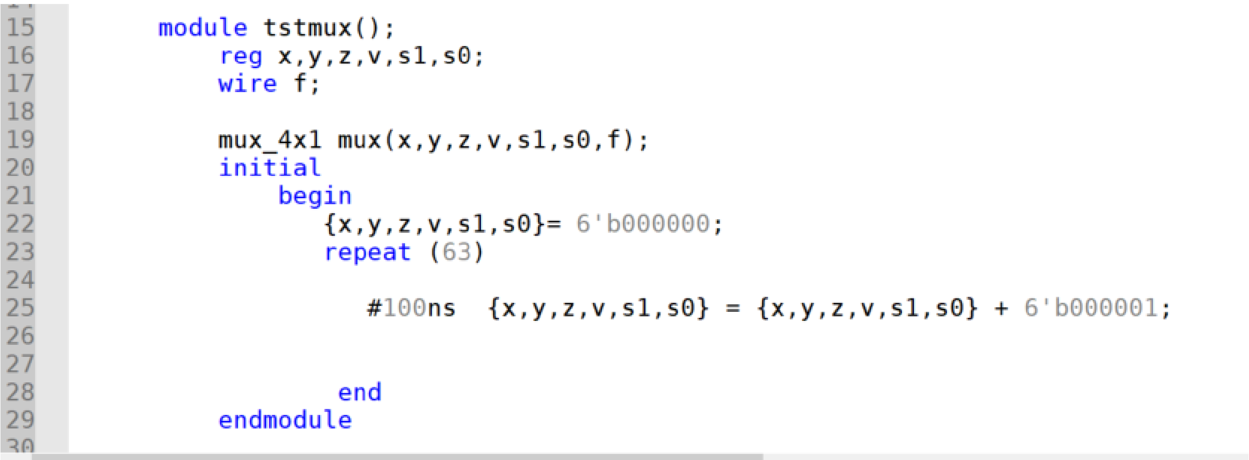
MUX4X1

First, in stage 1, I built mux4x1 using basic gates with 4 inputs (x, y, z, v) and 2 selection lines (s1, s0) and one output (f). The mux contains 4 and gates and one or gate and two invertor gates. I built it as shown below:



```
module mux_4x1 ( x,y,z,v,s1,s0,f);
    input x,y,z,v,s1,s0;
    output f;
    wire w0,w1,w2,w3,w4,w5;
   not #3ns gate1(w0,s0);
    not #3ns gate2(w1,s1);
    and #7ns gate3(w2,x,w0,w1);
    and #7ns gate4(w3,y,s0,w1);
    and #7ns gate5(w4,z,w0,s1);
    and #7ns gate6(w5,v,s0,s1);
    or #7ns gate7(f,w2,w3,w4,w5);
    endmodule
```

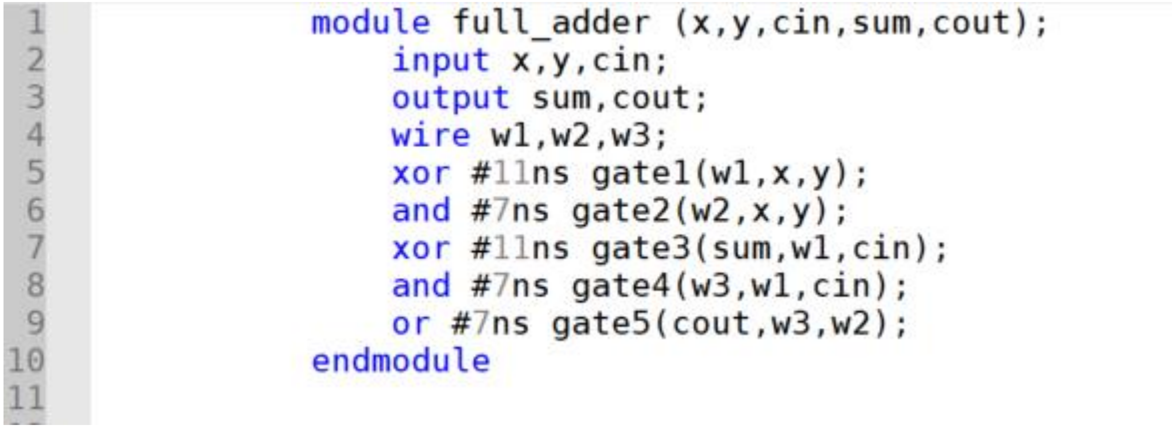
After this I built a test bench for this mux to become sure that it works perfectly. All inputs (x, y, z, v) became a registers and the output (f) became a wire. Then I called the mux4x1 module and I give all registers a value of zero then it increased by 1 every 100 ns and it's repeat 63 times(2^6 - 1) to include all cases. Number 6 equals numbers of inputs. I built it as shown below:



```
module tstmux();
    reg x,y,z,v,s1,s0;
    wire f;

    mux_4x1 mux(x,y,z,v,s1,s0,f);
    initial
        begin
           {x,y,z,v,s1,s0}= 6'b000000;
           repeat (63)

              #100ns  {x,y,z,v,s1,s0} = {x,y,z,v,s1,s0} + 6'b000001;


            end
    endmodule
```

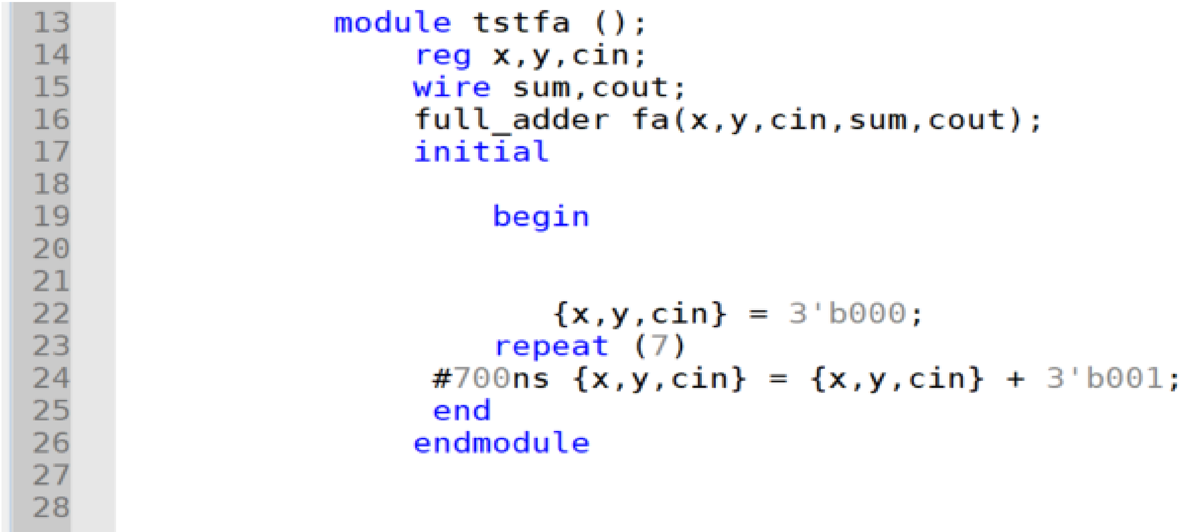
1-bit Full Adder

Then, I built 1-bit full adder using basic gates with 3 inputs (x, y, carry in) and 2 output(sum and carry out). The full adder contains 2 and gates and one or gate and two xor gates. I built it as shown below:



```
module full_adder (x,y,cin,sum,cout);
    input x,y,cin;
    output sum,cout;
    wire w1,w2,w3;
    xor #11ns gate1(w1,x,y);
    and #7ns gate2(w2,x,y);
    xor #11ns gate3(sum,w1,cin);
    and #7ns gate4(w3,w1,cin);
    or #7ns gate5(cout,w3,w2);
endmodule
```

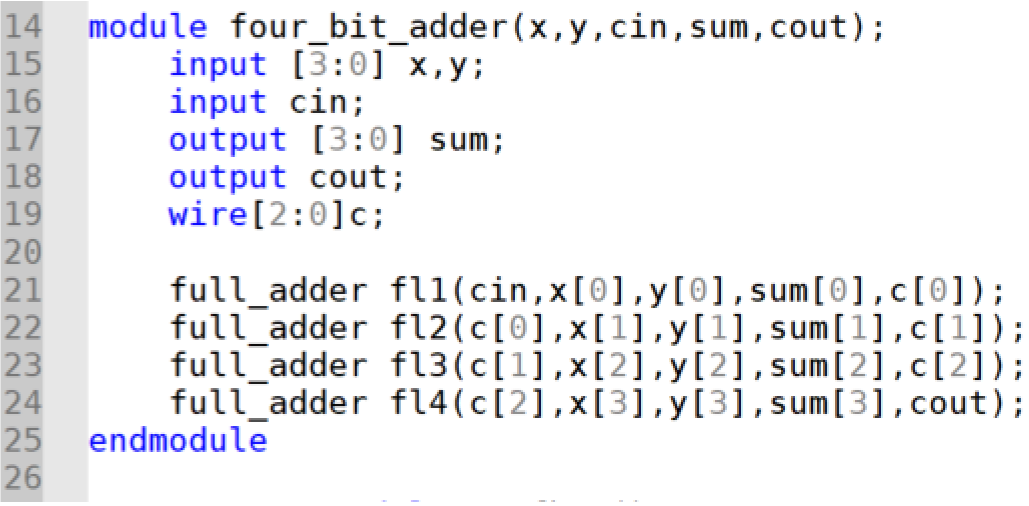
After this I built a test bench for this full adder to become sure that it works perfectly. All inputs (x, y, carry in) became a registers and the outputs (sum and carry out) became a wire. Then I called the full adder module and I give all registers a value of zero then it increased by 1 every 700 ns and it’s repeat 7 times(2^3 -1) to include all cases. 3 inputs. I built it as shown below:



```
module tstfa ();
    reg x,y,cin;
    wire sum,cout;
    full_adder fa(x,y,cin,sum,cout);
    initial

        begin


           {x,y,cin} = 3'b000;
        repeat (7)
     #700ns {x,y,cin} = {x,y,cin} + 3'b001;
     end
    endmodule
```

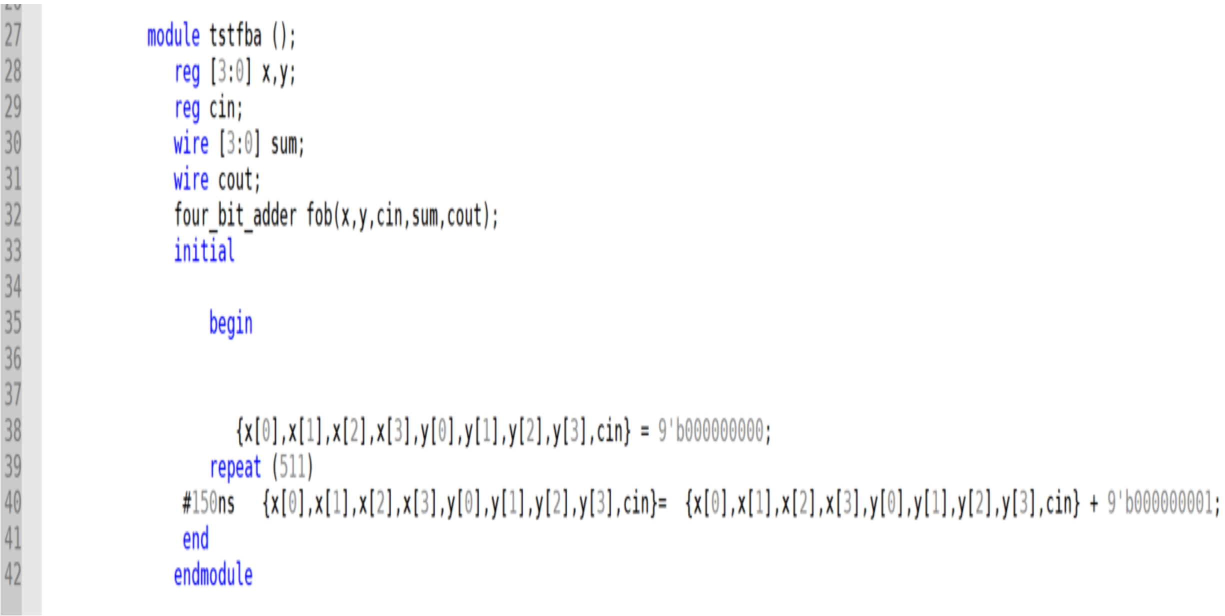
Ripple Adder

I built ripple adder by calling full adder module 4 times, and the input (x, y) is in array and output sum also in array of four index [3:0]. I built it as shown below:



```
module four_bit_adder(x,y,cin,sum,cout);
    input [3:0] x,y;
    input cin;
    output [3:0] sum;
    output cout;
    wire[2:0]c;

    full_adder fl1(cin,x[0],y[0],sum[0],c[0]);
    full_adder fl2(c[0],x[1],y[1],sum[1],c[1]);
    full_adder fl3(c[1],x[2],y[2],sum[2],c[2]);
    full_adder fl4(c[2],x[3],y[3],sum[3],cout);
endmodule
```

After this I built a test bench for this ripple adder to become sure that it works perfectly. I give all registers a value of zero then it increased by 1 every 150 ns and it’s repeat 511 times(2^9 -1) to include all cases. 9 inputs. I built it as shown below:



```
module tstfba ();
   reg [3:0] x,y;
   reg cin;
   wire [3:0] sum;
   wire cout;
   four_bit_adder fob(x,y,cin,sum,cout);
   initial

      begin


        {x[0],x[1],x[2],x[3],y[0],y[1],y[2],y[3],cin} = 9'b000000000;
      repeat (511)
    #150ns   {x[0],x[1],x[2],x[3],y[0],y[1],y[2],y[3],cin}=  {x[0],x[1],x[2],x[3],y[0],y[1],y[2],y[3],cin} + 9'b000000001;
    end
   endmodule
```

Arithmetic Unit 1

I built the system1 by calling mux4x1 four times and calling the ripple adder, and there is a four invertor gates to give us B\.

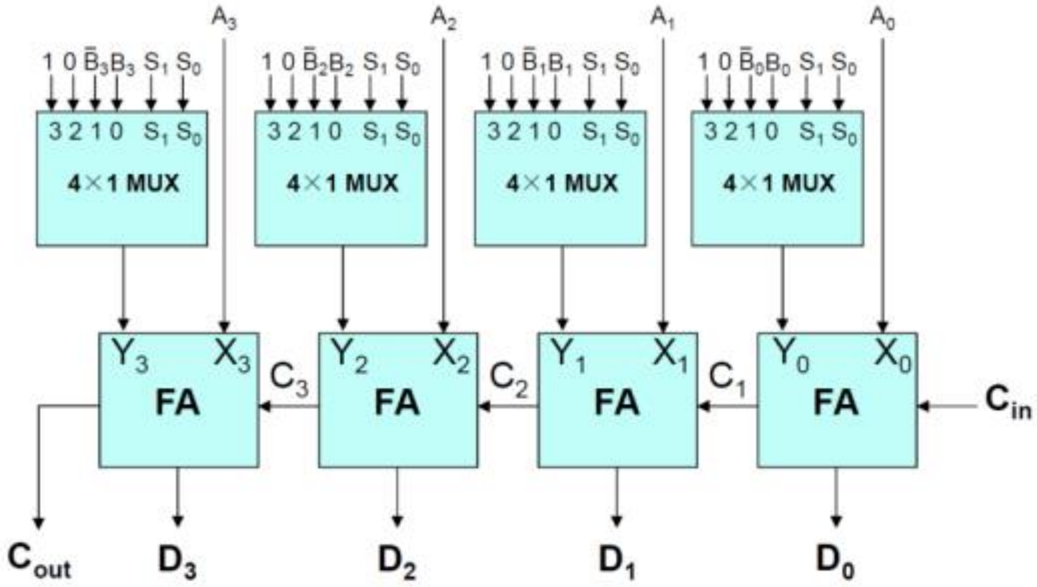
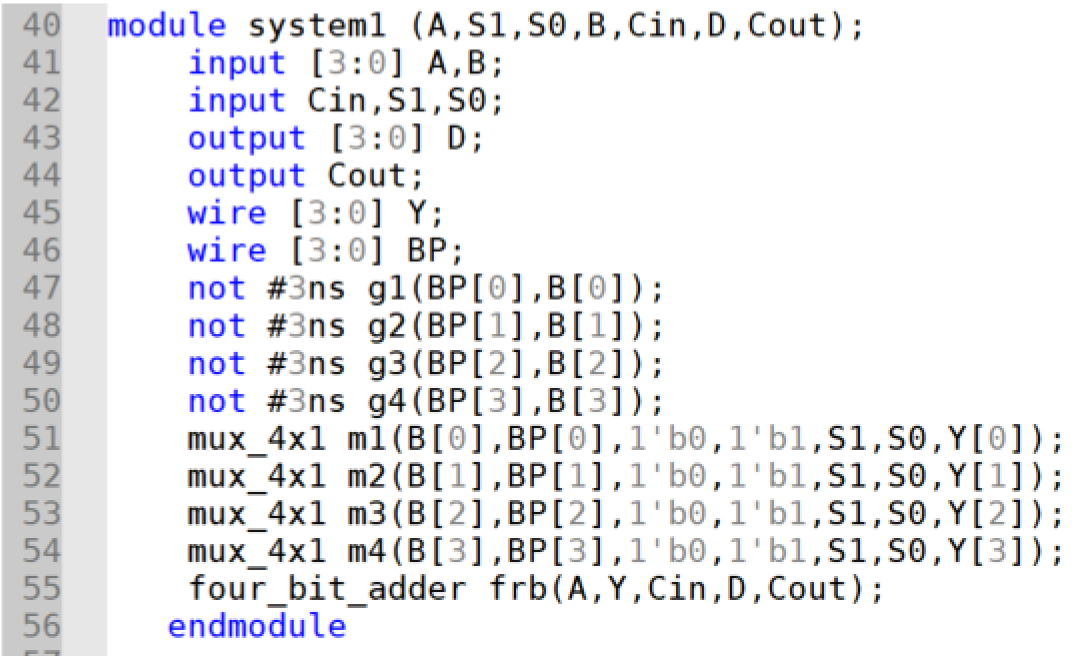


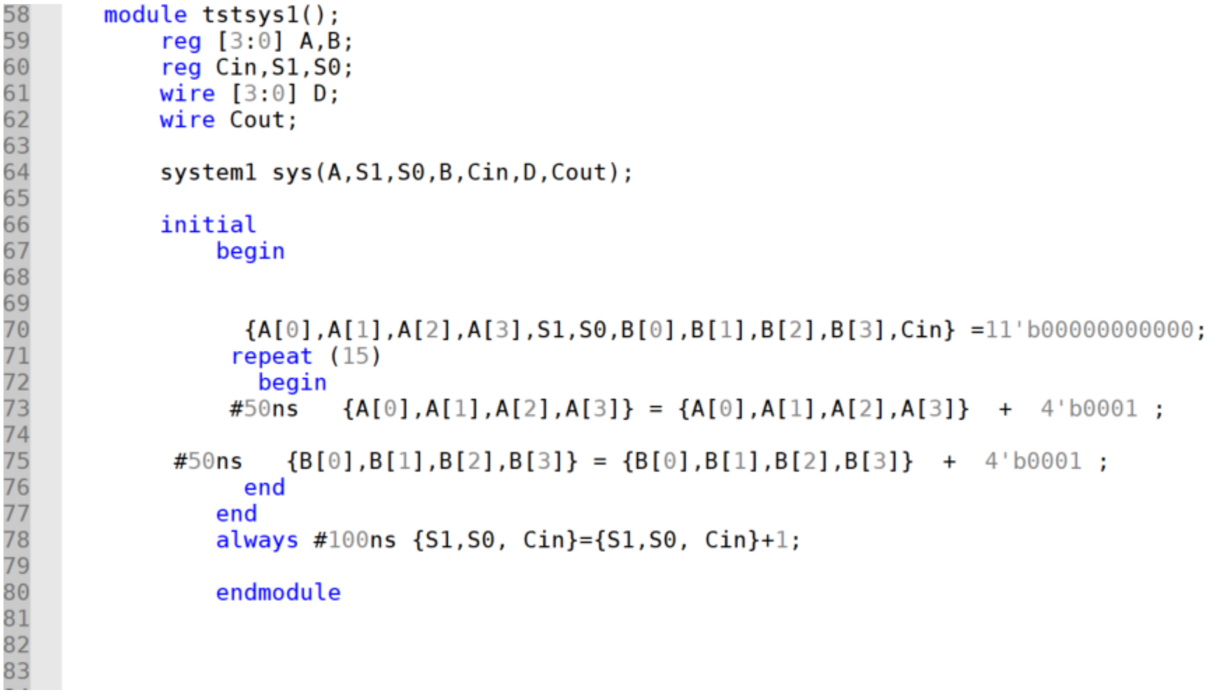
Figure 1: 4-bit Arithmetic Unit

I built the system below:



```
module system1 (A,S1,S0,B,Cin,D,Cout);
    input [3:0] A,B;
    input Cin,S1,S0;
    output [3:0] D;
    output Cout;
    wire [3:0] Y;
    wire [3:0] BP;
    not #3ns g1(BP[0],B[0]);
    not #3ns g2(BP[1],B[1]);
    not #3ns g3(BP[2],B[2]);
    not #3ns g4(BP[3],B[3]);
    mux_4x1 m1(B[0],BP[0],1'b0,1'b1,S1,S0,Y[0]);
    mux_4x1 m2(B[1],BP[1],1'b0,1'b1,S1,S0,Y[1]);
    mux_4x1 m3(B[2],BP[2],1'b0,1'b1,S1,S0,Y[2]);
    mux_4x1 m4(B[3],BP[3],1'b0,1'b1,S1,S0,Y[3]);
    four_bit_adder frb(A,Y,Cin,D,Cout);
   endmodule
```

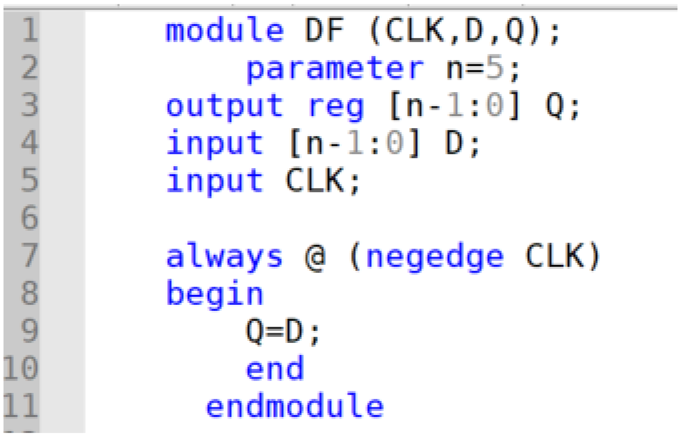
After this I built a test bench for this system to become sure that it works perfectly. I give all registers a value of zero then A and B increasing every 50ns.It’s repeated 15 times (2^4 -1). And S1,S0,Cin increasing every 100ns.



```
module tstsys1();
    reg [3:0] A,B;
    reg Cin,S1,S0;
    wire [3:0] D;
    wire Cout;

    system1 sys(A,S1,S0,B,Cin,D,Cout);

    initial
        begin


          {A[0],A[1],A[2],A[3],S1,S0,B[0],B[1],B[2],B[3],Cin} =11'b00000000000;
         repeat (15)
           begin
         #50ns   {A[0],A[1],A[2],A[3]} = {A[0],A[1],A[2],A[3]}  +  4'b0001 ;

     #50ns   {B[0],B[1],B[2],B[3]} = {B[0],B[1],B[2],B[3]}  +  4'b0001 ;
          end
        end
        always #100ns {S1,S0, Cin}={S1,S0, Cin}+1;

        endmodule
```

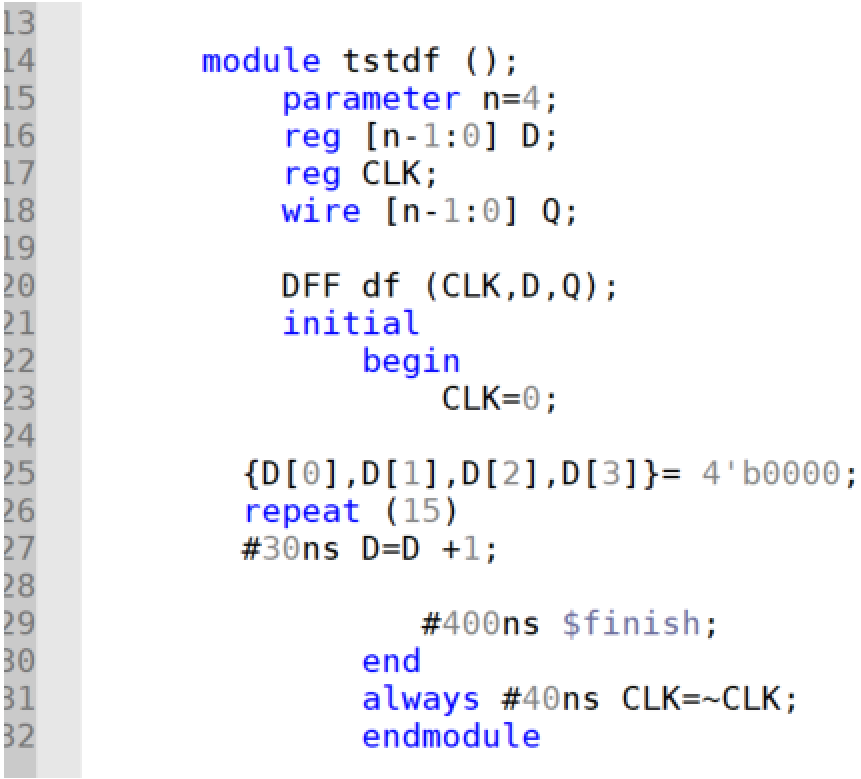
D-flip flop

I built the D flip flop. With input clock and D and output Q. And when the negative edge happened then Q=D. I built it below:



```
module DF (CLK,D,Q);
    parameter n=5;
output reg [n-1:0] Q;
input [n-1:0] D;
input CLK;

always @ (negedge CLK)
begin
    Q=D;
    end
  endmodule
```

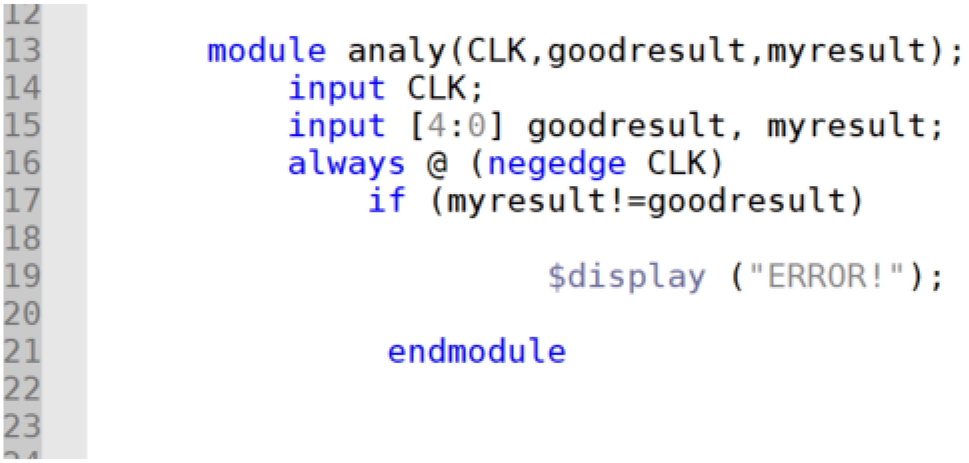
After this I built a test bench for this module to become sure that it works perfectly. I give all registers a value of zero then D increasing every 30ns. It’s repeated 15 times (2^4 -1). And CLK increasing every 40ns.



```

  module tstdf ();
      parameter n=4;
      reg [n-1:0] D;
      reg CLK;
      wire [n-1:0] Q;

      DFF df (CLK,D,Q);
      initial
          begin
              CLK=0;

{D[0],D[1],D[2],D[3]}= 4'b0000;
repeat (15)
#30ns D=D +1;

             #400ns $finish;
          end
          always #40ns CLK=~CLK;
          endmodule
```

Analyzer

I built the Analyzer. With input clock and good result and my result. And when the negative edge happened if my result don’t equal good result then display error . I built it below:



```
module analy(CLK,goodresult,myresult);
    input CLK;
    input [4:0] goodresult, myresult;
    always @ (negedge CLK)
        if (myresult!=goodresult)

                 $display ("ERROR!");

         endmodule
```

Stage2

Carry look ahead adder

In stage 2, I replace the ripple adder with a carry look ahead adder.

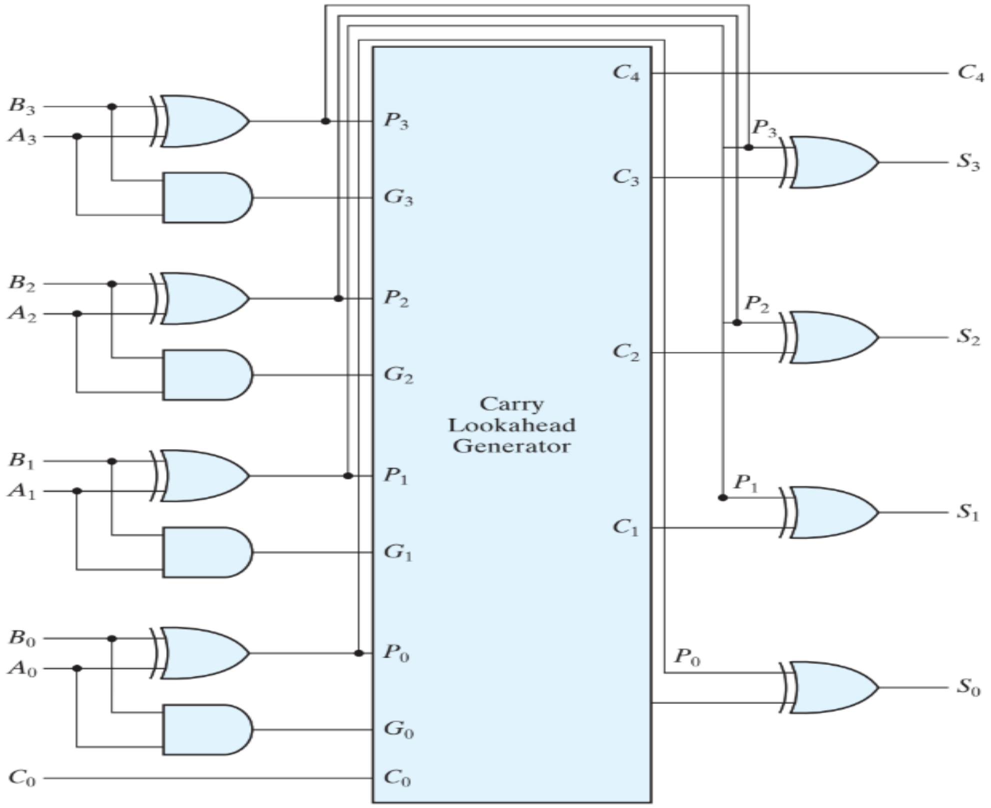


Figure 2: Carry Lookahead Generator [1]

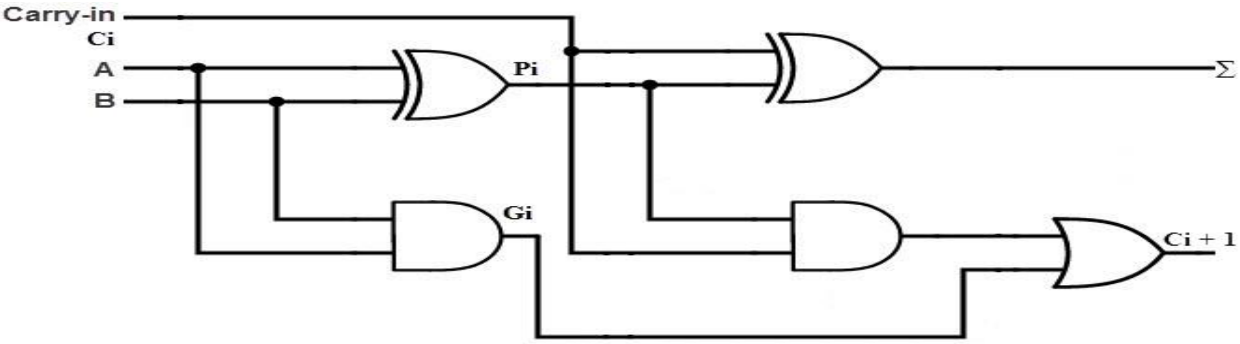
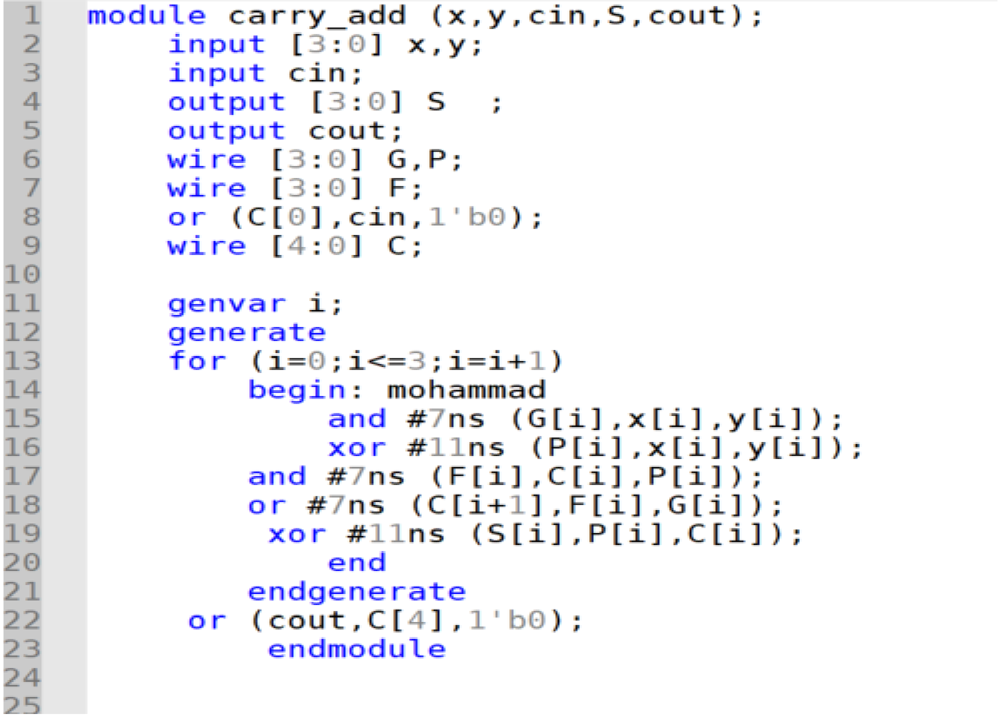


Figure 3: Carry lookahead [2]

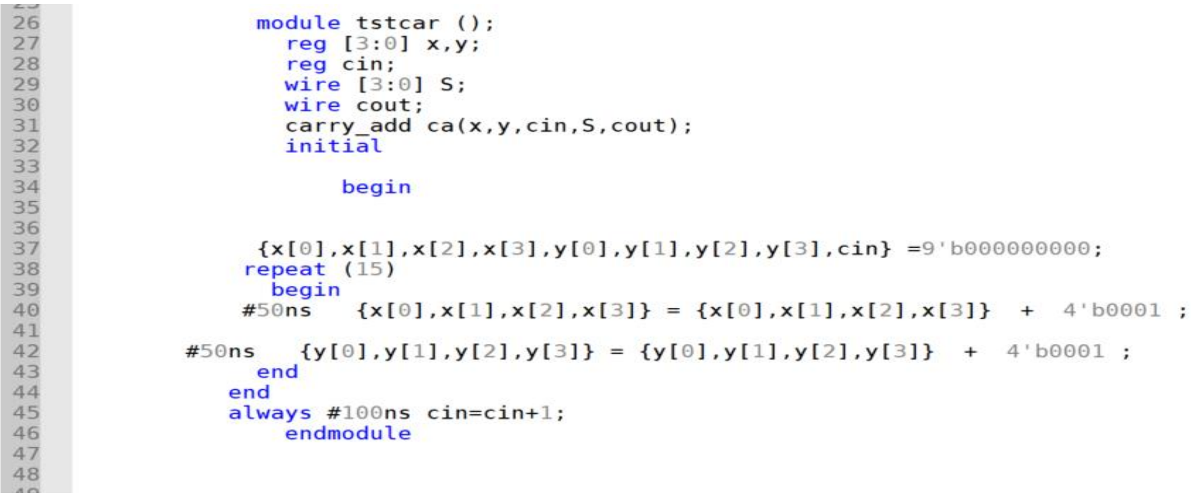
We find the sum and carryout from this equation as shown above.

I built the system of carry look ahead adder using for loop as shown.



```
module carry_add (x,y,cin,S,cout);
    input [3:0] x,y;
    input cin;
    output [3:0] S  ;
    output cout;
    wire [3:0] G,P;
    wire [3:0] F;
    or (C[0],cin,1'b0);
    wire [4:0] C;

    genvar i;
    generate
    for (i=0;i<=3;i=i+1)
        begin: mohammad
            and #7ns (G[i],x[i],y[i]);
            xor #11ns (P[i],x[i],y[i]);
        and #7ns (F[i],C[i],P[i]);
        or #7ns (C[i+1],F[i],G[i]);
         xor #11ns (S[i],P[i],C[i]);
            end
        endgenerate
     or (cout,C[4],1'b0);
         endmodule
```

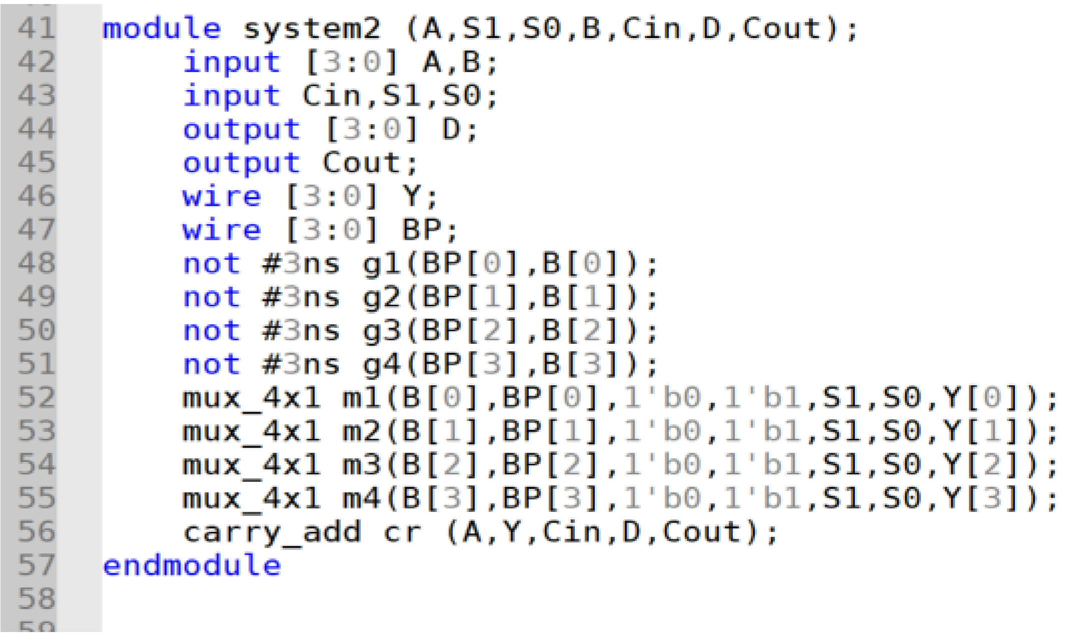
After this I built a test bench for this carry ahead adder to become sure that it works perfectly. I give all registers a value of zero then x and y increasing every 50ns.It's repeated 15 times (2^4 - 1). And cin increasing every 100ns.



```
              module tstcar ();
                reg [3:0] x,y;
                reg cin;
                wire [3:0] S;
                wire cout;
                carry_add ca(x,y,cin,S,cout);
                initial

                    begin


              {x[0],x[1],x[2],x[3],y[0],y[1],y[2],y[3],cin} =9'b000000000;
             repeat (15)
               begin
             #50ns   {x[0],x[1],x[2],x[3]} = {x[0],x[1],x[2],x[3]}  +  4'b0001 ;

         #50ns   {y[0],y[1],y[2],y[3]} = {y[0],y[1],y[2],y[3]}  +  4'b0001 ;
              end
            end
            always #100ns cin=cin+1;
                endmodule
```

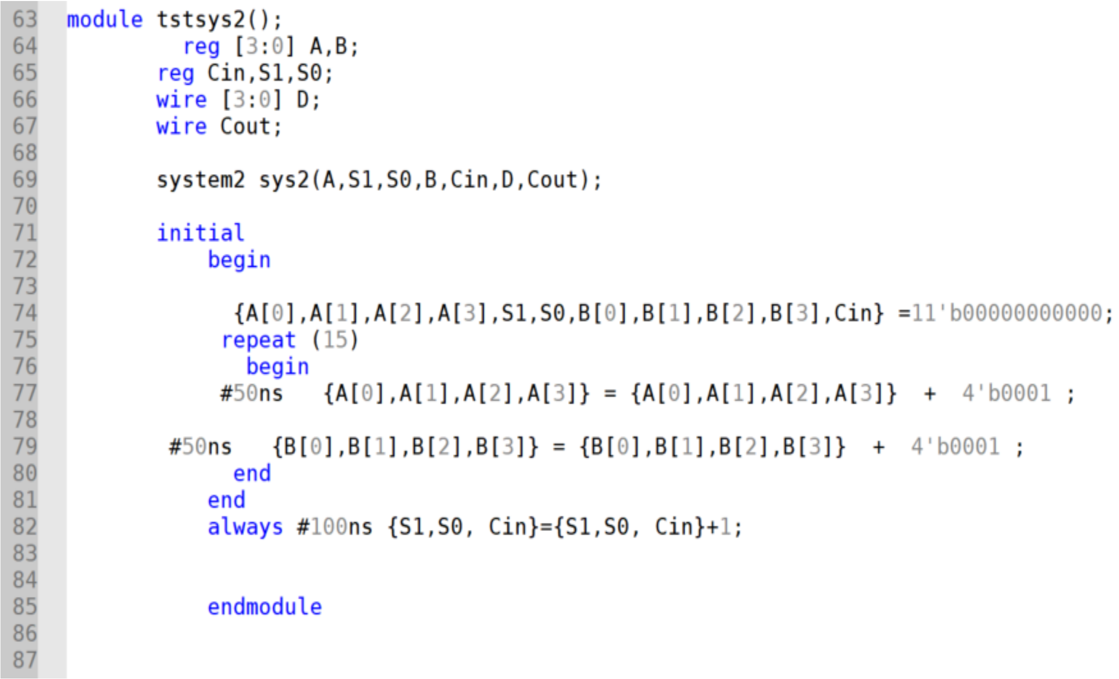
Arithmetic Unit 2

I built the system1 by calling mux4x1 four times and calling the carry lookahead adder, and there is a four invertor gates to give us B\. I built it as below:



```
module system2 (A,S1,S0,B,Cin,D,Cout);
    input [3:0] A,B;
    input Cin,S1,S0;
    output [3:0] D;
    output Cout;
    wire [3:0] Y;
    wire [3:0] BP;
    not #3ns g1(BP[0],B[0]);
    not #3ns g2(BP[1],B[1]);
    not #3ns g3(BP[2],B[2]);
    not #3ns g4(BP[3],B[3]);
    mux_4x1 m1(B[0],BP[0],1'b0,1'b1,S1,S0,Y[0]);
    mux_4x1 m2(B[1],BP[1],1'b0,1'b1,S1,S0,Y[1]);
    mux_4x1 m3(B[2],BP[2],1'b0,1'b1,S1,S0,Y[2]);
    mux_4x1 m4(B[3],BP[3],1'b0,1'b1,S1,S0,Y[3]);
    carry_add cr (A,Y,Cin,D,Cout);
endmodule
```

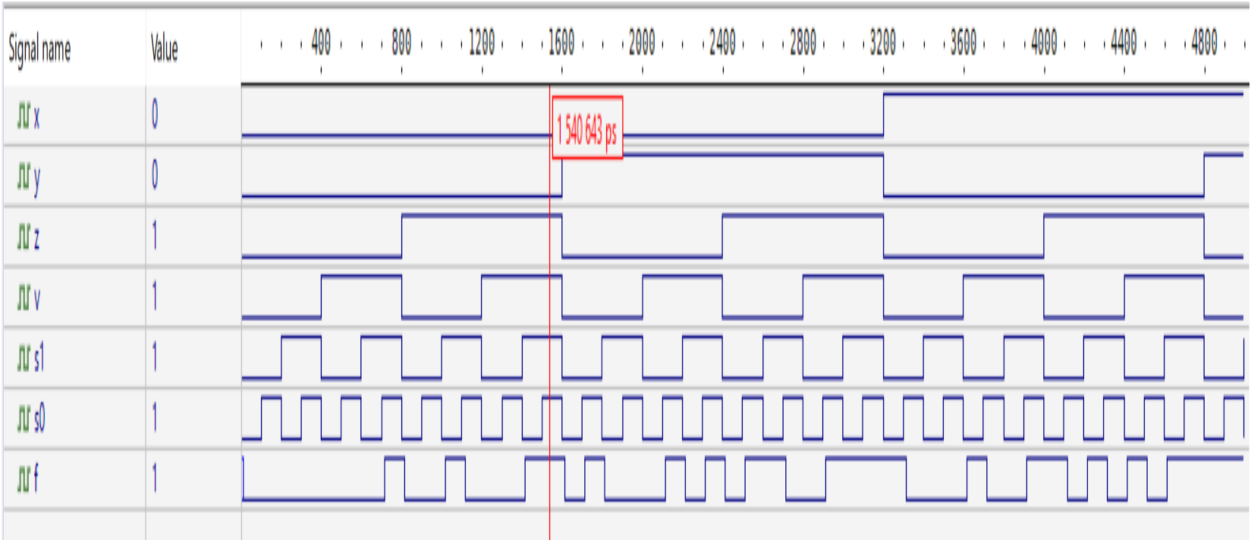
After this I built a test bench for this system to become sure that it works perfectly. I give all registers a value of zero then A and B increasing every 50ns.It's repeated 15 times (2^4 -1). And S1,S0,Cin increasing every 100ns.



```
module tstsys2();
         reg [3:0] A,B;
       reg Cin,S1,S0;
       wire [3:0] D;
       wire Cout;

       system2 sys2(A,S1,S0,B,Cin,D,Cout);

       initial
           begin

             {A[0],A[1],A[2],A[3],S1,S0,B[0],B[1],B[2],B[3],Cin} =11'b00000000000;
            repeat (15)
              begin
            #50ns   {A[0],A[1],A[2],A[3]} = {A[0],A[1],A[2],A[3]}  +  4'b0001 ;

        #50ns   {B[0],B[1],B[2],B[3]} = {B[0],B[1],B[2],B[3]}  +  4'b0001 ;
             end
           end
           always #100ns {S1,S0, Cin}={S1,S0, Cin}+1;


           endmodule
```

Results

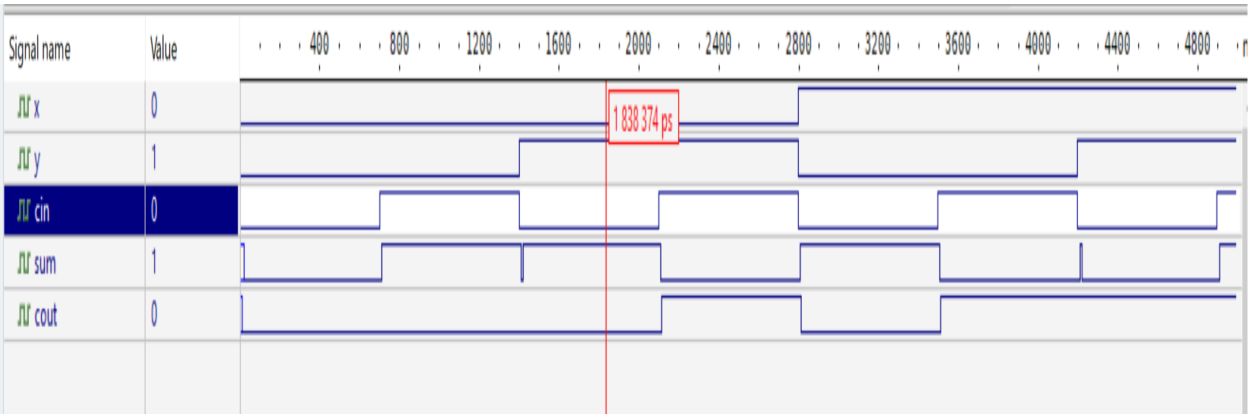
MUX4X1



Signal name	Value
x	0
y	0
z	1
v	1
s1	1
s0	1
f	1

When the selection lines (S1, S0) equal (0,0) the output f equal the value in the input x, Such as in 838ps. And when the selection lines (S1, S0) equal (0,1) the output f equal the value in the input y, Such as in 532ps. And when the selection lines (S1, S0) equal (1,0) the output f equal the value in the input z, Such as in 1888ps. And when the selection lines (S1, S0) equal (1,1) the output f equal the value in the input z, Such as above.

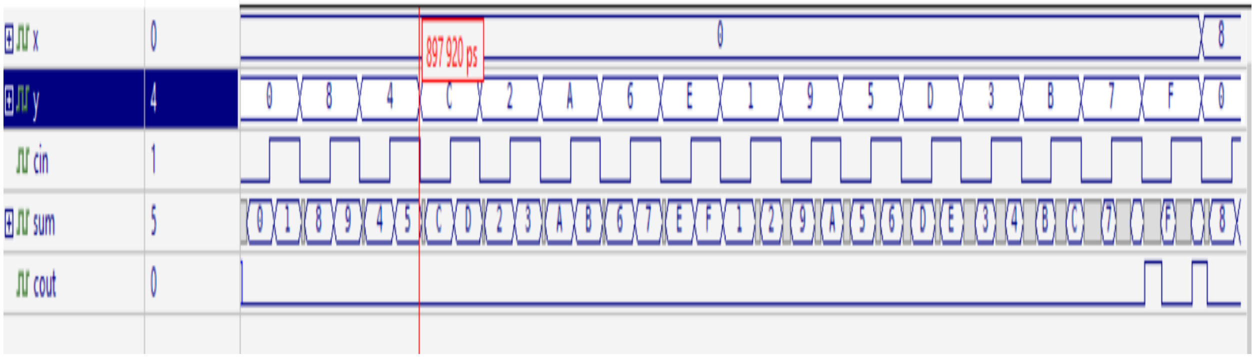
1-bit Full Adder



Signal name	Value
x	0
y	1
cin	0
sum	1
cout	0

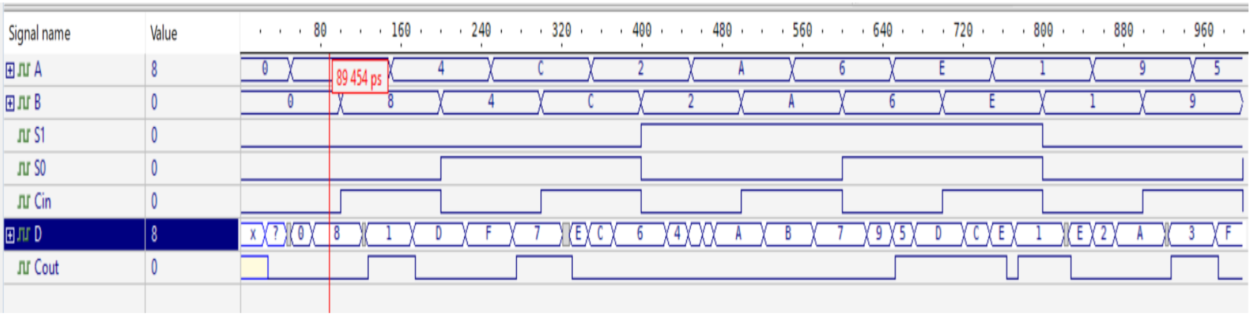
When x, y, cin equal {010} the sum equal 1 and carryout equal 0. Also for example, when x, y, cin equal {110} then sum equal 0 and carry out equal 1, and so on.

Ripple Adder



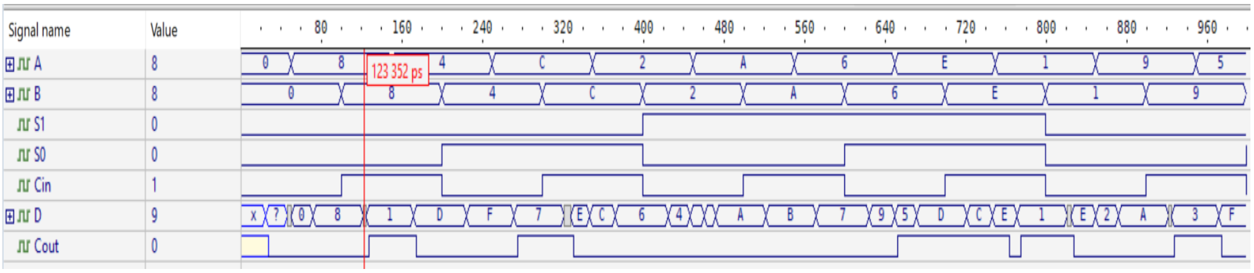
It works As a full adder, but now there is 4bit.(4+1+0=5).

Arithmetic Unit 1



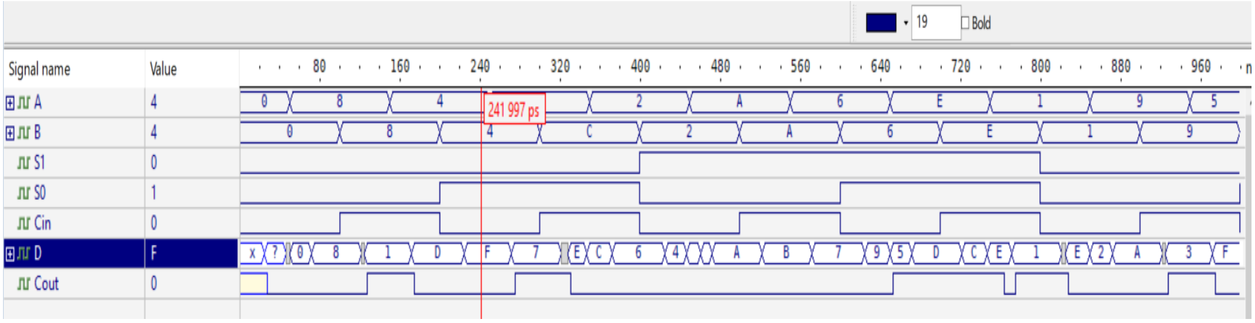
Signal name	Value
A	8
B	0
S1	0
S0	0
Cin	0
D	8
Cout	0

Case 1: when S1,S0,Cin = 000 , then D= A+B (8=8+0). And Cout =0.



Signal name	Value
A	8
B	8
S1	0
S0	0
Cin	1
D	9
Cout	0

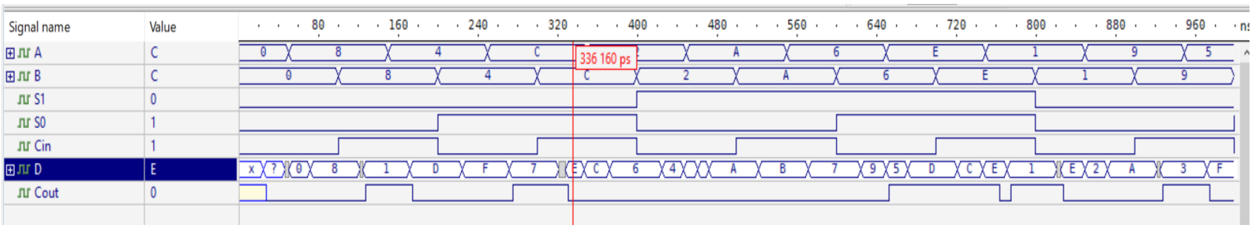
Case 2: when S1,S0,Cin = 001 , then D= A+B+1 (9=8+0+1). And Cout =0.



19 □Bold

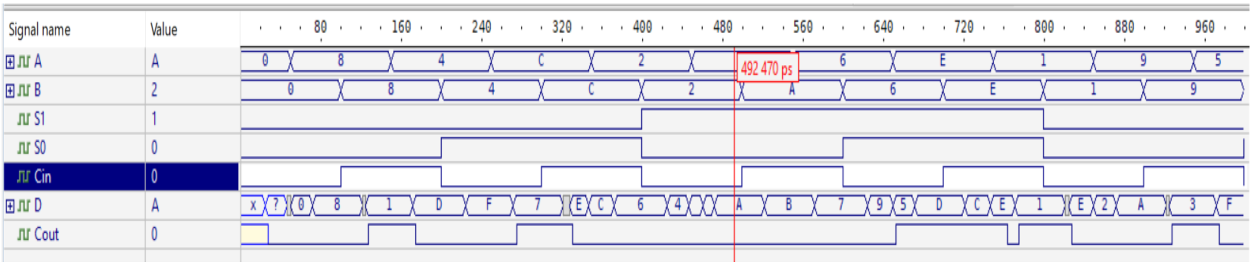
Signal name	Value
A	4
B	4
S1	0
S0	1
Cin	0
D	F
Cout	0

Case 3: when S1,S0,Cin = 010 , then D= A+Bnot (F=4+B+1). And Cout =0.



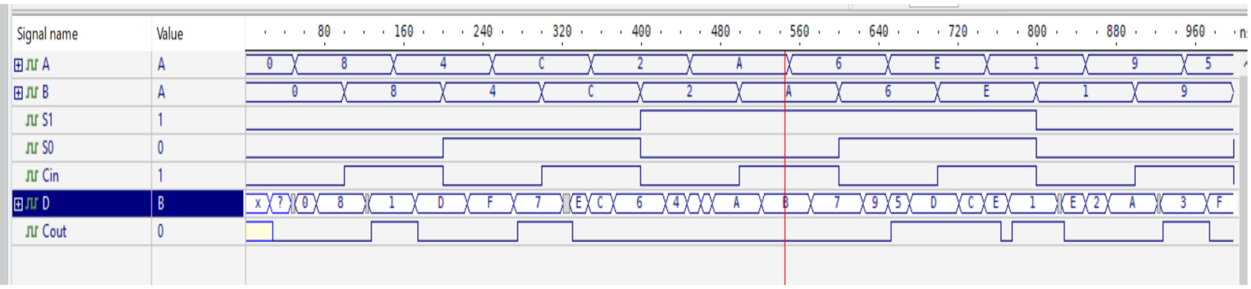
Signal name	Value
A	C
B	C
S1	0
S0	1
Cin	1
D	E
Cout	0

Case 4: when S1,S0,Cin = 011 , then D= A+Bnot+1 (E!=C+3+1). The output must be 0 and carry out =1 . There is a glitch because of the delay of the circuits.



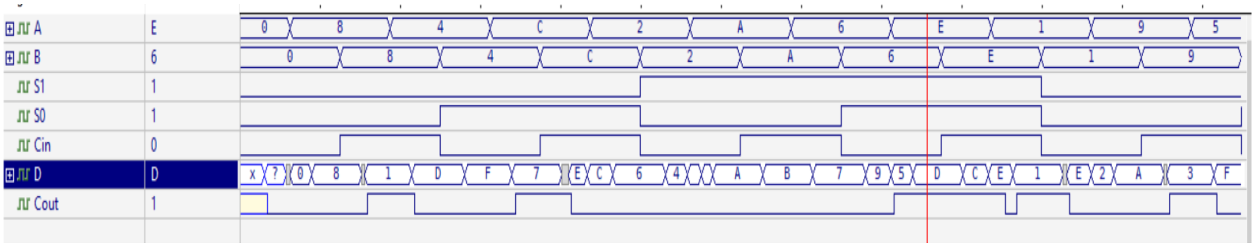
Signal name	Value
A	A
B	2
S1	1
S0	0
Cin	0
D	A
Cout	0

Case 5: when S1,S0,Cin = 100 , then D= A (A=A). And Cout =0.



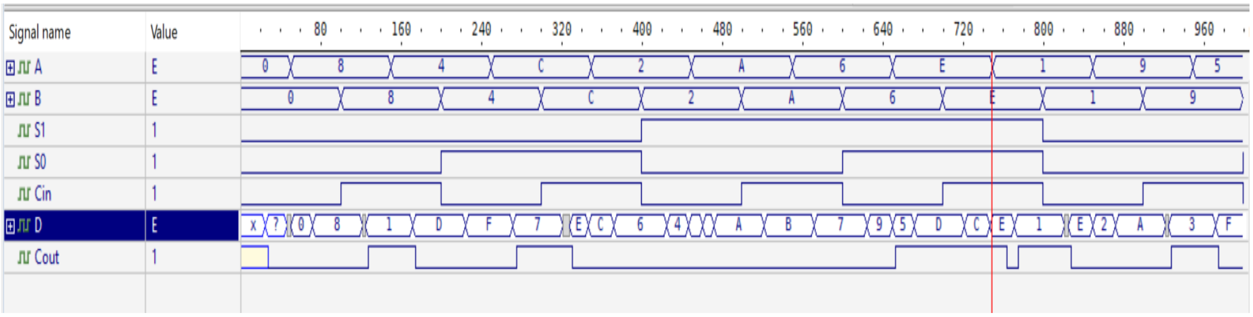
Signal name	Value
A	A
B	A
S1	1
S0	0
Cin	1
D	B
Cout	0

Case 6: when S1,S0,Cin = 101 , then D= A+1 (B=A+1). And Cout=0.



A	E
B	6
S1	1
S0	1
Cin	0
D	D
Cout	1

Case 7: when S1,S0,Cin = 110 , then D= A-1 (D=E-1). And Cout=1.

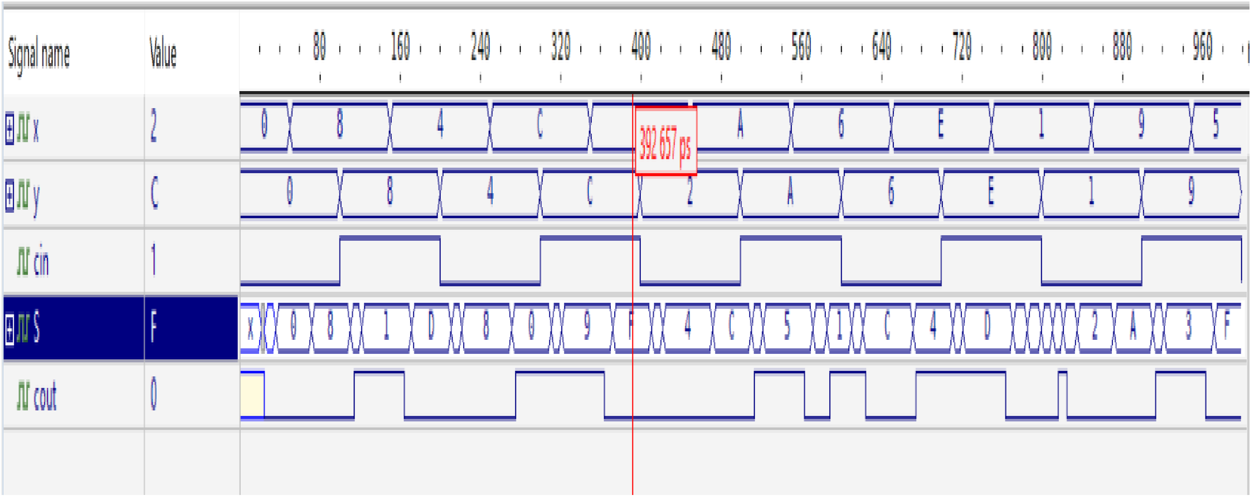


Signal name	Value
A	E
B	E
S1	1
S0	1
Cin	1
D	E
Cout	1

Case 8: when S1,S0,Cin = 111 , then D= A (E=E). And Cout =1.

So 7 cases are correct and 1 case are incorrect because of glitches.

Carry look ahead adder



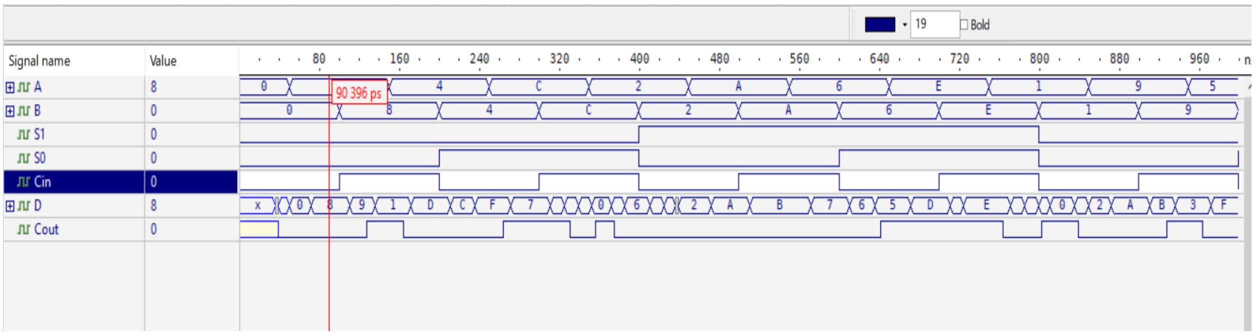
This adder works same as the ripple adder but the difference that the carry look ahead adder is faster than ripple adder but the output is the same.

For example (2+C+1=F) (DONE).

ERRORS:

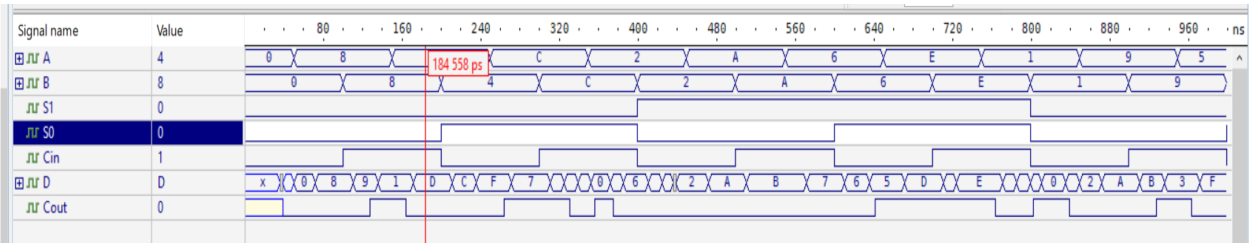
```
# KERNAL: ERROR
```

Arithmetic Unit 2



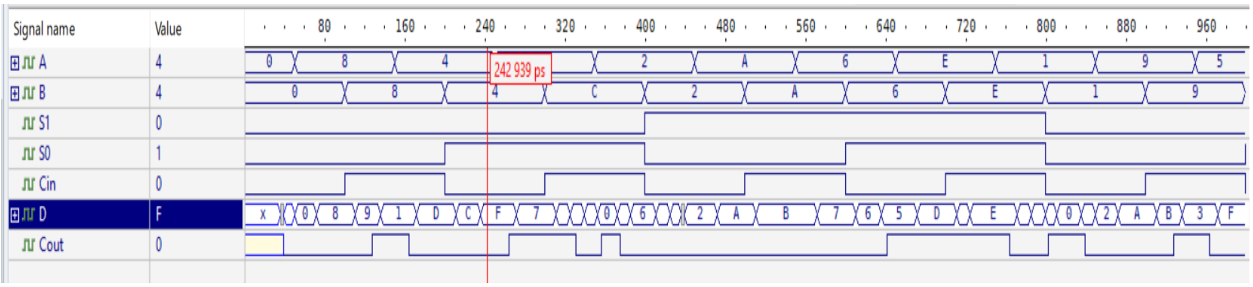
Signal name	Value
A	8
B	0
S1	0
S0	0
Cin	0
D	8
Cout	0

Case 1: when S1,S0,Cin = 000 , then D= A+B (8=8+0). And Cout=0.



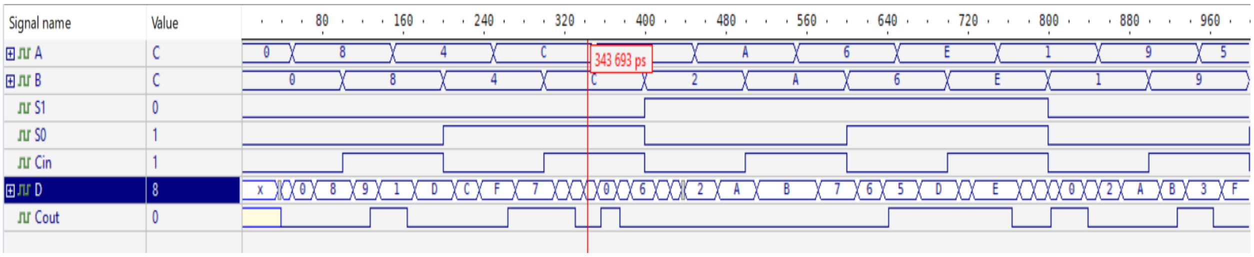
Signal name	Value
A	4
B	8
S1	0
S0	0
Cin	1
D	D
Cout	0

Case 2: when S1,S0,Cin = 001 , then D= A+B+1 (D=8+4+1). And Cout=0.



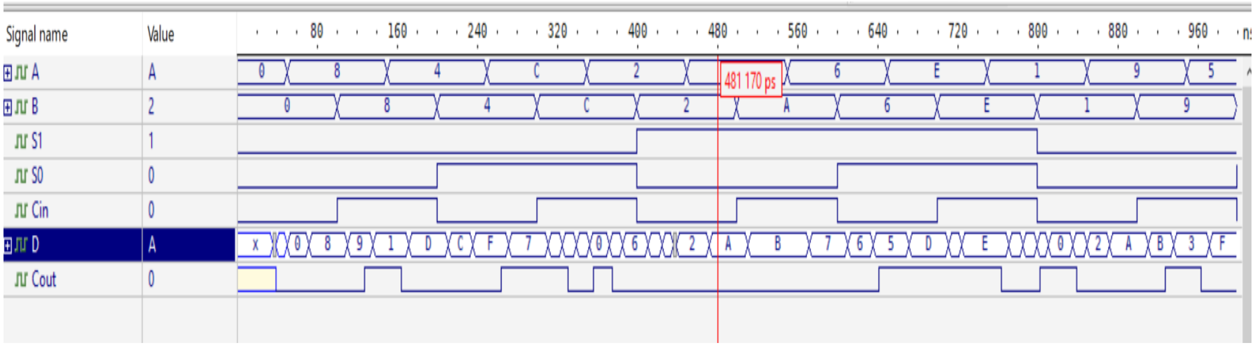
Signal name	Value
A	4
B	4
S1	0
S0	1
Cin	0
D	F
Cout	0

Case 3: when S1,S0,Cin = 010 , then D= A+Bnot (F=4+B). And Cout=0.

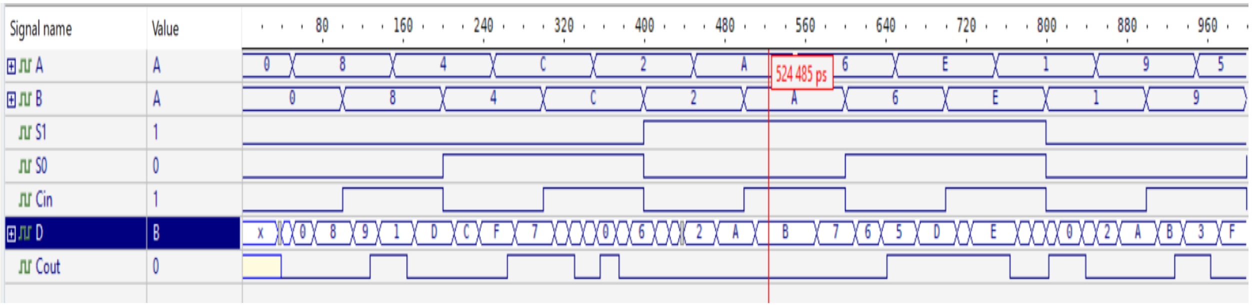


Signal name	Value
A	C
B	C
S1	0
S0	1
Cin	1
D	8
Cout	0

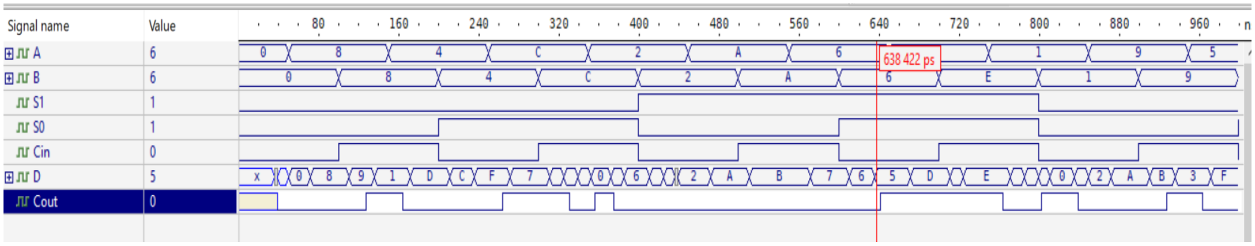
Case 4: when S1,S0,Cin = 011 , then D= A+Bnot+1 (8!=C+3+1). The output must be 0 and carry out =1. There is a glitch because of the delay of the circuits. The output is appear after 20ps approxematily.



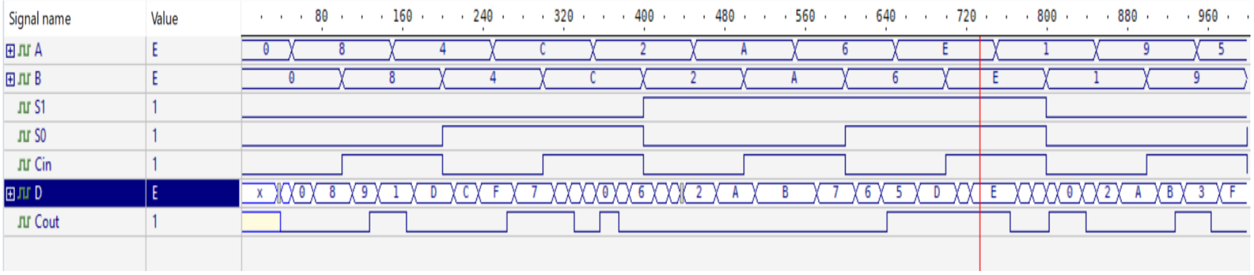
Case 5: when S1,S0,Cin = 100 , then D= A (A=A). And Cout =0.



Case 6: when S1,S0,Cin = 101 , then D= A+1 (B=A+1). And Cout=0.

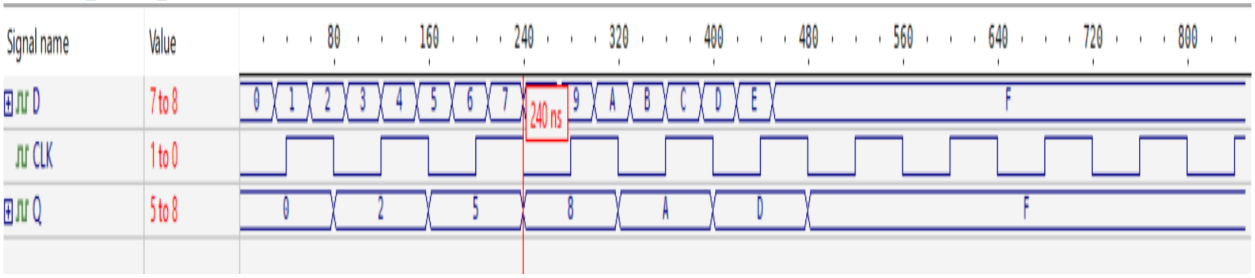


Case 7: when S1,S0,Cin = 110 , then D= A-1 (5=6-1). And Cout=0.



Case 8: when S1,S0,Cin = 111 , then D= A (E=E). And Cout=0.

D- flip flop



When the negative edge coming then Q=D.

ERRORS:

```
# KERNAL: ERROR
```

Conclusion and Future work

After I implemented an arithmetic unit, I did the test to my codes, and test if I have some errors because of glitches due to delay in some cases. The result of my codes equal the theoretical result. I used two types of adder a ripple adder in stage 1, and the look-ahead adder in stage 2, and I noticed the differences between two types and wrote them correctly in Active HDL student edition. Look-ahead adder is faster than ripple adder. And I become more mastering with Verilog HDL by making delays and printing an error and more familiar with the program active HDL student edition.

References

[1] ENCS2340 Book . (2022, 8 22). Retrieved from https://drive.google.com/drive/folders/1ANBtBi6OvSOncbuUXnbtLlolJS67WnMr

[2] Electronics Hub. (2022, 8 22). Retrieved from https://www.electronicshub.org/wp-content/uploads/2015/06/Full-adder1.jpg

Appendix

```
module mux_4x1 ( x,y,z,v,s1,s0,f) ;
    input x,y,z,v,s1,s0;
    output f;
    wire w0,w1,w2,w3,w4,w5;
   not #3ns gate1(w0,s0);
    not #3ns gate2(w1,s1);
    and #7ns gate3(w2,x,w0,w1);
    and #7ns gate4(w3,y,s0,w1);
    and #7ns gate5(w4,z,w0,s1);
    and #7ns gate6(w5,v,s0,s1);
    or #7ns gate7(f,w2,w3,w4,w5);
    endmodule



    module tstmux();
        reg x,y,z,v,s1,s0;
        wire f;

        mux_4x1 mux(x,y,z,v,s1,s0,f);
        initial
            begin
                {x,y,z,v,s1,s0}= 6'b000000;
                repeat (63)

                    #100ns  {x,y,z,v,s1,s0} = {x,y,z,v,s1,s0} + 6'b000001;
                end
        endmodule
```

```
module full_adder (x,y,cin,sum,cout);

             input x,y,cin;

             output sum,cout;

             wire w1,w2,w3;

             xor #11ns gate1(w1,x,y);

             and #7ns gate2(w2,x,y);

             xor #11ns gate3(sum,w1,cin);

             and #7ns gate4(w3,w1,cin);

             or #7ns gate5(cout,w3,w2);

         endmodule

         module tstfa ();

             reg x,y,cin;

             wire sum,cout;

             full_adder fa(x,y,cin,sum,cout);

             initial

                 begin

                     {x,y,cin} = 3'b000;

                 repeat (7)

                #700ns {x,y,cin} = {x,y,cin} + 3'b001;

                end

             endmodule
```

```
module four_bit_adder(x,y,cin,sum,cout);

        input [3:0] x,y;

        input cin;

        output [3:0] sum;

        output cout;

        wire[2:0]c;

        full_adder fl1(cin,x[0],y[0],sum[0],c[0]);

        full_adder fl2(c[0],x[1],y[1],sum[1],c[1]);

        full_adder fl3(c[1],x[2],y[2],sum[2],c[2]);

        full_adder fl4(c[2],x[3],y[3],sum[3],cout);

endmodule

                   module tstfba ();

                          reg [3:0] x,y;

                          reg cin;

                          wire [3:0] sum;

                          wire cout;

                          four_bit_adder fob(x,y,cin,sum,cout);

                          initial

          begin

         {x[0],x[1],x[2],x[3],y[0],y[1],y[2],y[3],cin} = 9'b000000000;

                                  repeat (511)
```

```
                                        #150ns                    {x[0],x[1],x[2],x[3],y[0],y[1],y[2],y[3],cin}=
{x[0],x[1],x[2],x[3],y[0],y[1],y[2],y[3],cin} + 9'b000000001;

                                        end

                                endmodule




module system1 (A,S1,S0,B,Cin,D,Cout);

          input [3:0] A,B;

          input Cin,S1,S0;

          output [3:0] D;

          output Cout;

          wire [3:0] Y;

          wire [3:0] BP;

          not #3ns g1(BP[0],B[0]);

          not #3ns g2(BP[1],B[1]);

          not #3ns g3(BP[2],B[2]);

          not #3ns g4(BP[3],B[3]);

          mux_4x1 m1(B[0],BP[0],1'b0,1'b1,S1,S0,Y[0]);

          mux_4x1 m2(B[1],BP[1],1'b0,1'b1,S1,S0,Y[1]);

          mux_4x1 m3(B[2],BP[2],1'b0,1'b1,S1,S0,Y[2]);

          mux_4x1 m4(B[3],BP[3],1'b0,1'b1,S1,S0,Y[3]);

          four_bit_adder frb(A,Y,Cin,D,Cout);

  endmodule
```

```
 module tstsys1();

     reg [3:0] A,B;

     reg Cin,S1,S0;

     wire [3:0] D;

     wire Cout;


     system1 sys(A,S1,S0,B,Cin,D,Cout);


     initial

         begin


             {A[0],A[1],A[2],A[3],S1,S0,B[0],B[1],B[2],B[3],Cin}
=11'b00000000000;

            repeat (15)

             begin

            #50ns  {A[0],A[1],A[2],A[3]} = {A[0],A[1],A[2],A[3]}  +  4'b0001 ;


        #50ns  {B[0],B[1],B[2],B[3]} = {B[0],B[1],B[2],B[3]}  +  4'b0001 ;

            end

         end

         always #100ns {S1,S0, Cin}={S1,S0, Cin}+1;

         endmodule
```

```
module carry_add (x,y,cin,S,cout);
    input [3:0] x,y;
    input cin;
    output [3:0] S  ;
    output cout;
    wire [3:0] G,P;
    wire [3:0] F;
    or (C[0],cin,1'b0);
    wire [4:0] C;

    genvar i;
    generate
    for (i=0;i<=3;i=i+1)
        begin: mohammad
            and #7ns (G[i],x[i],y[i]);
            xor #11ns (P[i],x[i],y[i]);
        and #7ns (F[i],C[i],P[i]);
     or #7ns (C[i+1],F[i],G[i]);
        xor #11ns (S[i],P[i],C[i]);
            end
        endgenerate
    or (cout,C[4],1'b0);
        endmodule
```

```
           module tstcar ();
             reg [3:0] x,y;
             reg cin;
             wire [3:0] S;
             wire cout;
             carry_add ca(x,y,cin,S,cout);
             initial

                         begin


           {x[0],x[1],x[2],x[3],y[0],y[1],y[2],y[3],cin} =9'b000000000;
          repeat (15)
            begin
          #50ns   {x[0],x[1],x[2],x[3]} = {x[0],x[1],x[2],x[3]}  +  4'b0001 ;

#50ns   {y[0],y[1],y[2],y[3]} = {y[0],y[1],y[2],y[3]}  +  4'b0001 ;
           end
   end
   always #100ns cin=cin+1;
             endmodule
```

```
module system2 (A,S1,S0,B,Cin,D,Cout);

    input [3:0] A,B;

    input Cin,S1,S0;

    output [3:0] D;

    output Cout;

    wire [3:0] Y;

    wire [3:0] BP;

    not #3ns g1(BP[0],B[0]);

    not #3ns g2(BP[1],B[1]);

    not #3ns g3(BP[2],B[2]);

    not #3ns g4(BP[3],B[3]);

    mux_4x1 m1(B[0],BP[0],1'b0,1'b1,S1,S0,Y[0]);

    mux_4x1 m2(B[1],BP[1],1'b0,1'b1,S1,S0,Y[1]);

    mux_4x1 m3(B[2],BP[2],1'b0,1'b1,S1,S0,Y[2]);

    mux_4x1 m4(B[3],BP[3],1'b0,1'b1,S1,S0,Y[3]);

    carry_add cr (A,Y,Cin,D,Cout);

endmodule
```

```
module tstsys2();

                reg [3:0] A,B;

          reg Cin,S1,S0;

          wire [3:0] D;

          wire Cout;


          system2 sys2(A,S1,S0,B,Cin,D,Cout);


          initial

                  begin


                        {A[0],A[1],A[2],A[3],S1,S0,B[0],B[1],B[2],B[3],Cin}
=11'b00000000000;

                       repeat (15)

                        begin

                       #50ns   {A[0],A[1],A[2],A[3]} = {A[0],A[1],A[2],A[3]}  +  4'b0001 ;


               #50ns   {B[0],B[1],B[2],B[3]} = {B[0],B[1],B[2],B[3]}  +  4'b0001 ;

                        end

                  end

                  always #100ns {S1,S0, Cin}={S1,S0, Cin}+1;


                  endmodule
```

```
            module DF (CLK,D,Q);

            parameter n=5;

output reg [n-1:0] Q;

input [n-1:0] D;

input CLK;


always @ (negedge CLK)

begin

            Q=D;

            end

  endmodule




  module tstdf ();

              parameter n=4;

              reg [n-1:0] D;

              reg CLK;

              wire [n-1:0] Q;


              DFF df (CLK,D,Q);

              initial

                          begin

                                      CLK=0;
```

```
                    {D[0],D[1],D[2],D[3]}= 4'b0000;

                    repeat (15)

                    #30ns D=D +1;


                              #400ns $finish;

                         end

                         always #40ns CLK=~CLK;

                         endmodule


module analy(CLK,goodresult,myresult);

                    input CLK;

                    input [4:0] goodresult, myresult;

                    always @ (negedge CLK)

                         if (myresult!=goodresult)


                                   $display ("ERROR!");


                         endmodule
```