



**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.....	2
List of Tables .....	3
Brief introduction and background.....	4
Design philosophy .....	5
Stage1 .....	5
MUX4X1.....	5
1-bit Full Adder.....	6
Ripple Adder .....	7
Arithmetic Unit 1 .....	8
D-flip flop .....	10
Analyzer .....	11
Stage2.....	12
Carry look ahead adder .....	12
Arithmetic Unit 2 .....	14
Results .....	15
MUX4X1 .....	15
1-bit Full Adder .....	15
Ripple Adder.....	16
Arithmetic Unit 1 .....	17
Carry look ahead adder.....	19
Arithmetic Unit 2.....	20
D- flip flop .....	22
Conclusion and Future work .....	23
References.....	24
Appendix .....	25

## List of Figures

Figure 1: 4-bit Arithmetic Unit .....	8
Figure 2: Carry Lookahead Generator [1] .....	12
Figure 3: Carry lookahead [2].....	12

## List of Tables

Table 1: Arithmetic unit table.....	4
Table 2: Delay of gates.....	4

## 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 $D = A + Y + C_{in}$	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 inverter gates. I built it as shown below:

```
1  module mux_4x1 ( x,y,z,v,s1,s0,f);
2      input x,y,z,v,s1,s0;
3      output f;
4      wire w0,w1,w2,w3,w4,w5;
5      not #3ns gate1(w0,s0);
6      not #3ns gate2(w1,s1);
7      and #7ns gate3(w2,x,w0,w1);
8      and #7ns gate4(w3,y,s0,w1);
9      and #7ns gate5(w4,z,w0,s1);
10     and #7ns gate6(w5,v,s0,s1);
11     or #7ns gate7(f,w2,w3,w4,w5);
12     endmodule
13
```

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:

```
15  module tstmux();
16      reg x,y,z,v,s1,s0;
17      wire f;
18
19      mux_4x1 mux(x,y,z,v,s1,s0,f);
20      initial
21          begin
22              {x,y,z,v,s1,s0}= 6'b000000;
23              repeat (63)
24                  #100ns {x,y,z,v,s1,s0} = {x,y,z,v,s1,s0} + 6'b000001;
25
26
27          end
28      endmodule
29
30
```

### 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:

```
1      module full_adder (x,y,cin,sum,cout);
2          input x,y,cin;
3          output sum,cout;
4          wire w1,w2,w3;
5          xor #11ns gate1(w1,x,y);
6          and #7ns gate2(w2,x,y);
7          xor #11ns gate3(sum,w1,cin);
8          and #7ns gate4(w3,w1,cin);
9          or #7ns gate5(cout,w3,w2);
10     endmodule
11
```

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:

```
13     module tstfa ();
14         reg x,y,cin;
15         wire sum,cout;
16         full_adder fa(x,y,cin,sum,cout);
17         initial
18
19             begin
20
21                 {x,y,cin} = 3'b000;
22                 repeat (7)
23                     #700ns {x,y,cin} = {x,y,cin} + 3'b001;
24                 end
25             endmodule
26
27
28
```

## 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:

```
14 module four_bit_adder(x,y,cin,sum,cout);
15     input [3:0] x,y;
16     input cin;
17     output [3:0] sum;
18     output cout;
19     wire[2:0]c;
20
21     full_adder fl1(cin,x[0],y[0],sum[0],c[0]);
22     full_adder fl2(c[0],x[1],y[1],sum[1],c[1]);
23     full_adder fl3(c[1],x[2],y[2],sum[2],c[2]);
24     full_adder fl4(c[2],x[3],y[3],sum[3],cout);
25 endmodule
26
```

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:

```
27 module tstfba ();
28     reg [3:0] x,y;
29     reg cin;
30     wire [3:0] sum;
31     wire cout;
32     four_bit_adder fob(x,y,cin,sum,cout);
33     initial
34
35         begin
36
37
38             {x[0],x[1],x[2],x[3],y[0],y[1],y[2],y[3],cin} = 9'b000000000;
39             repeat (511)
40                 #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;
41             end
42 endmodule
```



## Arithmetic Unit 1

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

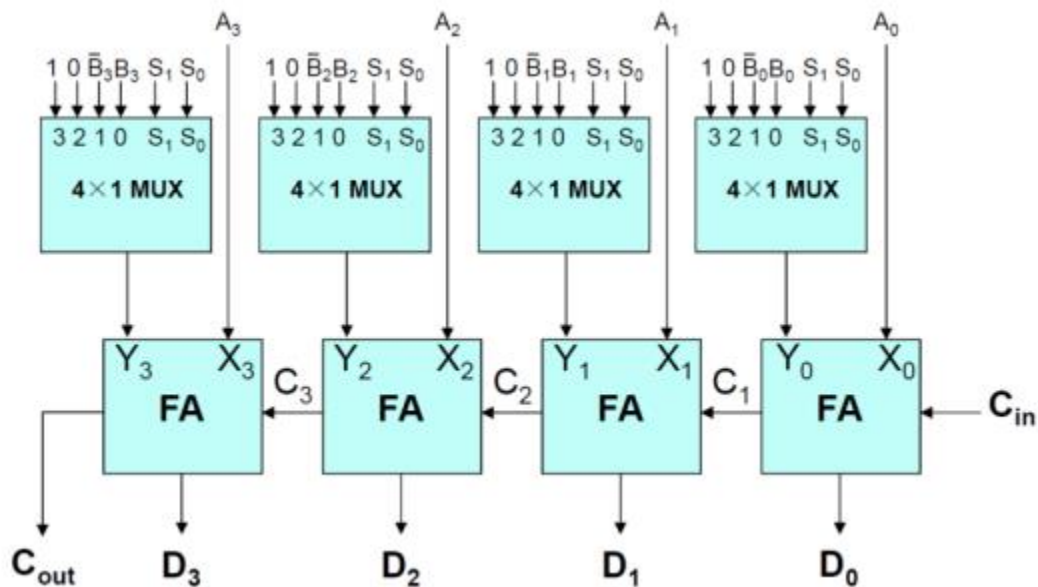


Figure 1: 4-bit Arithmetic Unit

I built the system below:

```

40 module system1 (A,S1,S0,B,Cin,D,Cout);
41     input [3:0] A,B;
42     input Cin,S1,S0;
43     output [3:0] D;
44     output Cout;
45     wire [3:0] Y;
46     wire [3:0] BP;
47     not #3ns g1(BP[0],B[0]);
48     not #3ns g2(BP[1],B[1]);
49     not #3ns g3(BP[2],B[2]);
50     not #3ns g4(BP[3],B[3]);
51     mux_4x1 m1(B[0],BP[0],1'b0,1'b1,S1,S0,Y[0]);
52     mux_4x1 m2(B[1],BP[1],1'b0,1'b1,S1,S0,Y[1]);
53     mux_4x1 m3(B[2],BP[2],1'b0,1'b1,S1,S0,Y[2]);
54     mux_4x1 m4(B[3],BP[3],1'b0,1'b1,S1,S0,Y[3]);
55     four_bit_adder frb(A,Y,Cin,D,Cout);
56 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.

```
58 module tstsys1();
59     reg [3:0] A,B;
60     reg Cin,S1,S0;
61     wire [3:0] D;
62     wire Cout;
63
64     system1 sys(A,S1,S0,B,Cin,D,Cout);
65
66     initial
67     begin
68
69
70         {A[0],A[1],A[2],A[3],S1,S0,B[0],B[1],B[2],B[3],Cin} =11'b000000000000;
71         repeat (15)
72         begin
73             #50ns {A[0],A[1],A[2],A[3]} = {A[0],A[1],A[2],A[3]} + 4'b0001 ;
74
75             #50ns {B[0],B[1],B[2],B[3]} = {B[0],B[1],B[2],B[3]} + 4'b0001 ;
76         end
77     end
78     always #100ns {S1,S0, Cin}={S1,S0, Cin}+1;
79
80 endmodule
81
82
83
```

### 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:

```
1  module DF (CLK,D,Q);
2      parameter n=5;
3      output reg [n-1:0] Q;
4      input [n-1:0] D;
5      input CLK;
6
7      always @ (negedge CLK)
8      begin
9          Q=D;
10     end
11 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.

```
13
14 module tstdf ();
15     parameter n=4;
16     reg [n-1:0] D;
17     reg CLK;
18     wire [n-1:0] Q;
19
20     DFF df (CLK,D,Q);
21     initial
22     begin
23         CLK=0;
24
25         {D[0],D[1],D[2],D[3]}= 4'b0000;
26         repeat (15)
27             #30ns D=D +1;
28
29             #400ns $finish;
30     end
31     always #40ns CLK=~CLK;
32 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:

```
12
13     module analy(CLK,goodresult,myresult);
14         input CLK;
15         input [4:0] goodresult, myresult;
16         always @ (negedge CLK)
17             if (myresult!=goodresult)
18
19                 $display ("ERROR!");
20
21         endmodule
22
23
24
```

## Stage2

### Carry look ahead adder

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

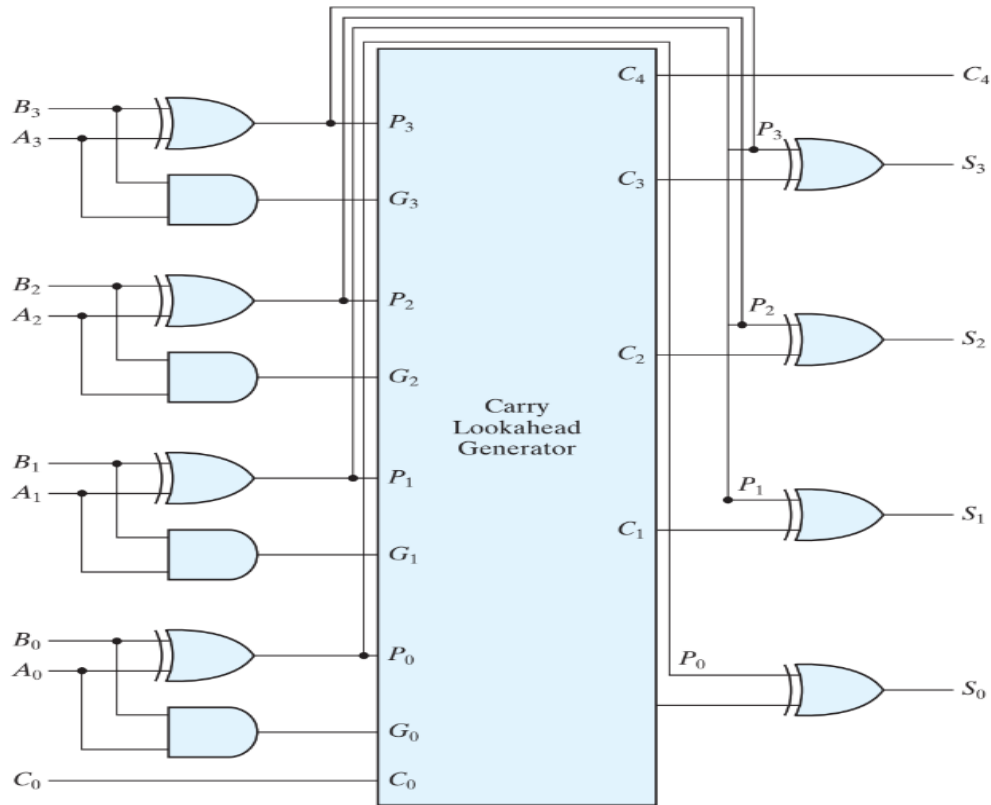


Figure 2: Carry Lookahead Generator [1]

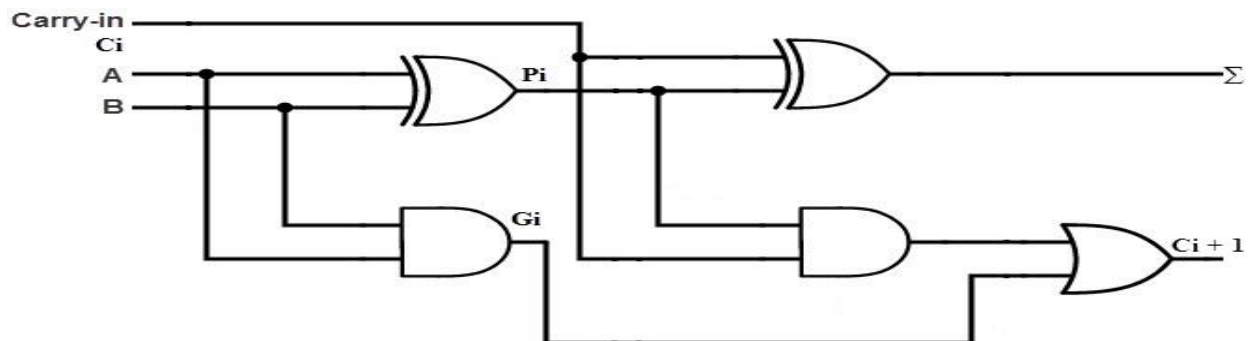


Figure 3: Carry lookahead [2]

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.

```

1  module carry_add (x,y,cin,S,cout);
2      input [3:0] x,y;
3      input cin;
4      output [3:0] S ;
5      output cout;
6      wire [3:0] G,P;
7      wire [3:0] F;
8      or (C[0],cin,1'b0);
9      wire [4:0] C;
10
11     genvar i;
12     generate
13     for (i=0;i<=3;i=i+1)
14     begin: mohammad
15         and #7ns (G[i],x[i],y[i]);
16         xor #11ns (P[i],x[i],y[i]);
17         and #7ns (F[i],C[i],P[i]);
18         or #7ns (C[i+1],F[i],G[i]);
19         xor #11ns (S[i],P[i],C[i]);
20     end
21     endgenerate
22     or (cout,C[4],1'b0);
23     endmodule
24
25

```

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.

```

26     module tstcar ();
27         reg [3:0] x,y;
28         reg cin;
29         wire [3:0] S;
30         wire cout;
31         carry_add ca(x,y,cin,S,cout);
32         initial
33
34             begin
35
36
37                 {x[0],x[1],x[2],x[3],y[0],y[1],y[2],y[3],cin} =9'b0000000000;
38                 repeat (15)
39                     begin
40                         #50ns {x[0],x[1],x[2],x[3]} = {x[0],x[1],x[2],x[3]} + 4'b0001 ;
41
42                         #50ns {y[0],y[1],y[2],y[3]} = {y[0],y[1],y[2],y[3]} + 4'b0001 ;
43                     end
44                 end
45                 always #100ns cin=cin+1;
46             endmodule
47
48

```

## 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:

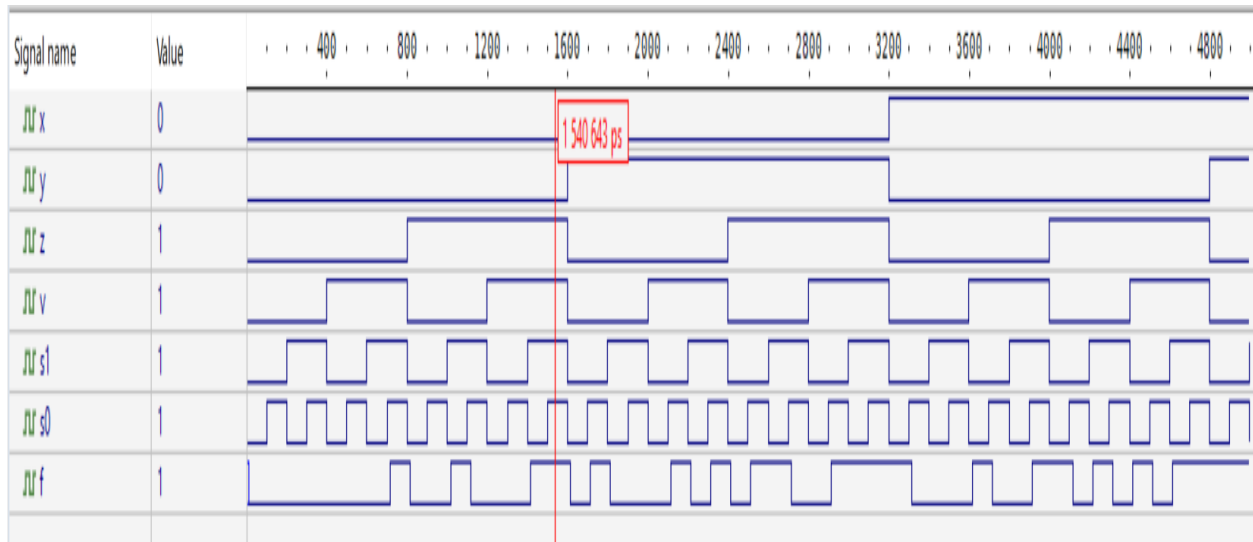
```
41 module system2 (A,S1,S0,B,Cin,D,Cout);
42     input [3:0] A,B;
43     input Cin,S1,S0;
44     output [3:0] D;
45     output Cout;
46     wire [3:0] Y;
47     wire [3:0] BP;
48     not #3ns g1(BP[0],B[0]);
49     not #3ns g2(BP[1],B[1]);
50     not #3ns g3(BP[2],B[2]);
51     not #3ns g4(BP[3],B[3]);
52     mux_4x1 m1(B[0],BP[0],1'b0,1'b1,S1,S0,Y[0]);
53     mux_4x1 m2(B[1],BP[1],1'b0,1'b1,S1,S0,Y[1]);
54     mux_4x1 m3(B[2],BP[2],1'b0,1'b1,S1,S0,Y[2]);
55     mux_4x1 m4(B[3],BP[3],1'b0,1'b1,S1,S0,Y[3]);
56     carry_add cr (A,Y,Cin,D,Cout);
57 endmodule
58
59
```

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.

```
63 module tstsys2();
64     reg [3:0] A,B;
65     reg Cin,S1,S0;
66     wire [3:0] D;
67     wire Cout;
68
69     system2 sys2(A,S1,S0,B,Cin,D,Cout);
70
71     initial
72     begin
73
74         {A[0],A[1],A[2],A[3],S1,S0,B[0],B[1],B[2],B[3],Cin} =11'b00000000000;
75         repeat (15)
76         begin
77             #50ns {A[0],A[1],A[2],A[3]} = {A[0],A[1],A[2],A[3]} + 4'b0001 ;
78
79             #50ns {B[0],B[1],B[2],B[3]} = {B[0],B[1],B[2],B[3]} + 4'b0001 ;
80         end
81     end
82     always #100ns {S1,S0, Cin}={S1,S0, Cin}+1;
83
84
85 endmodule
86
87
```

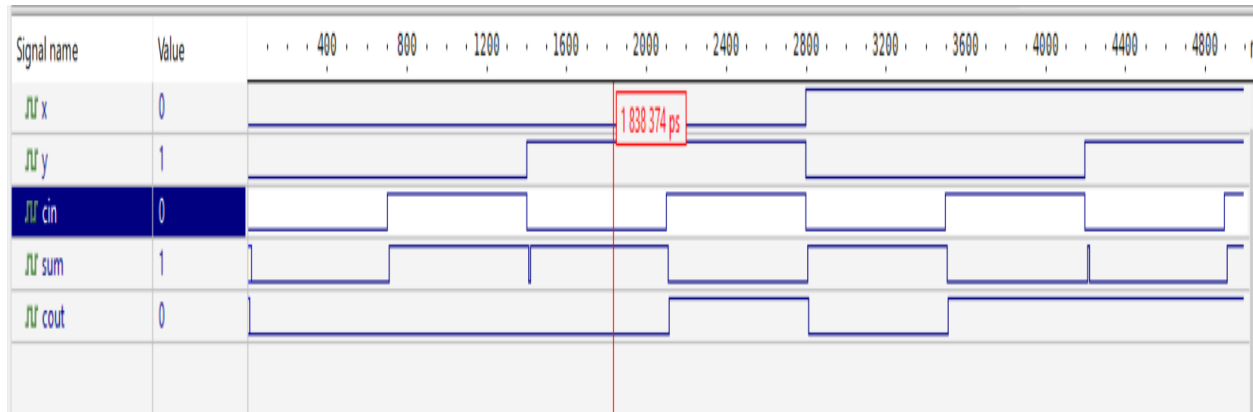
## Results

### MUX4X1



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.

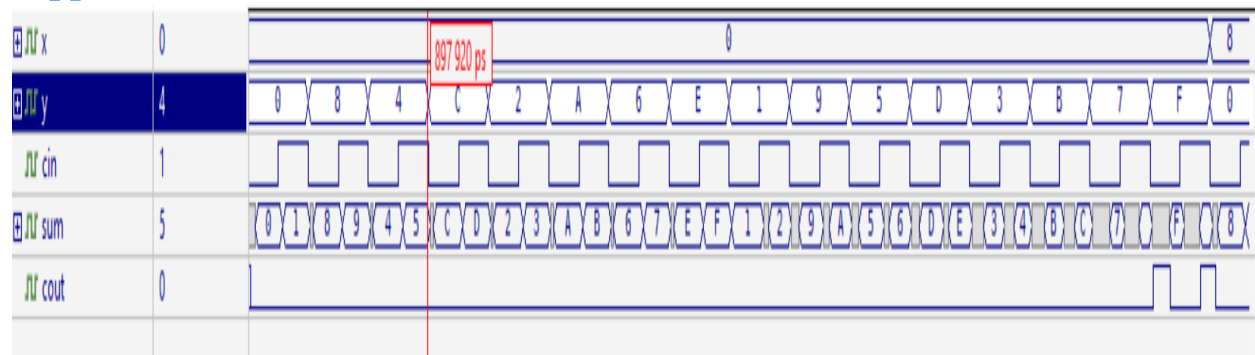
### 1-bit Full Adder



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.

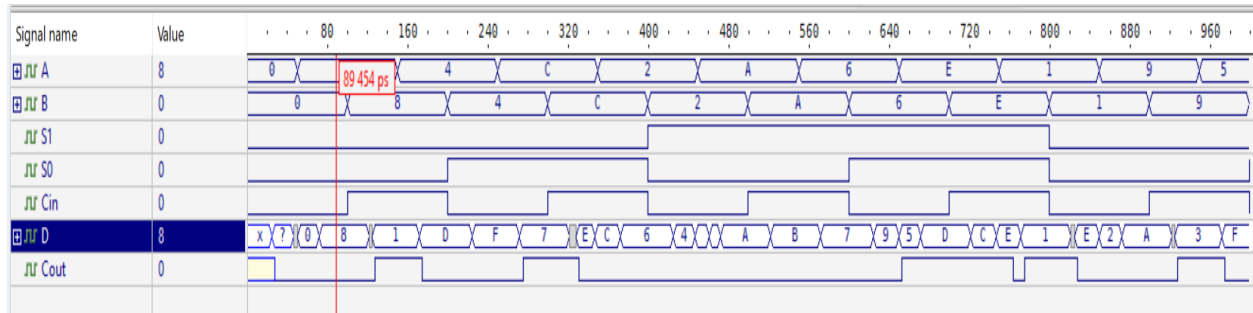


## Ripple Adder



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

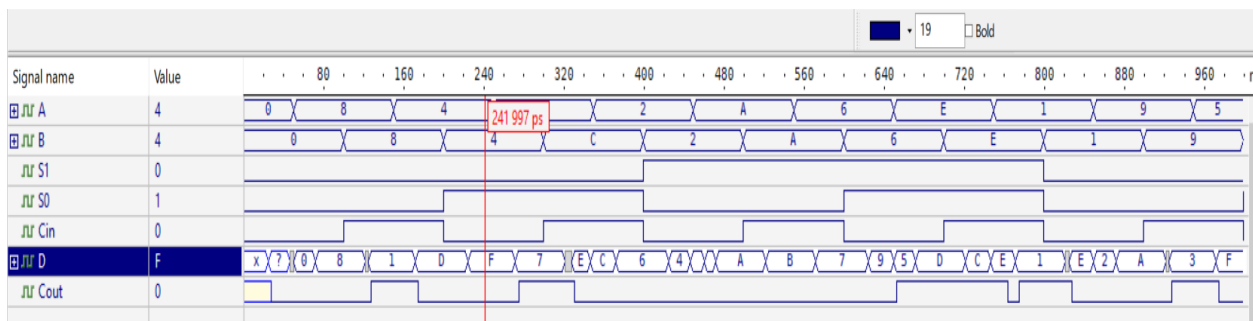
## Arithmetic Unit 1



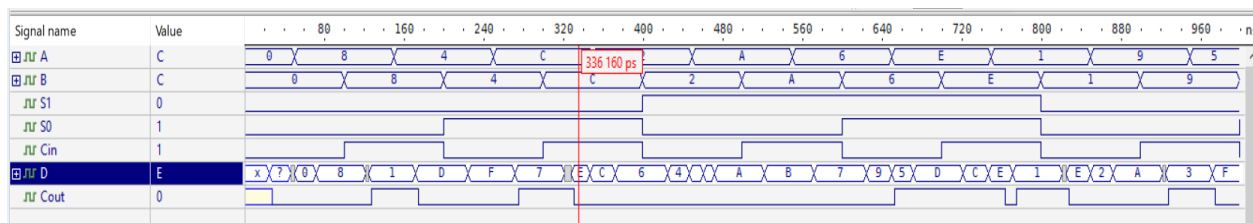
Case 1: when S1,S0,Cin = 000 , then D= A+B (8=8+0). And Cout =0.



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



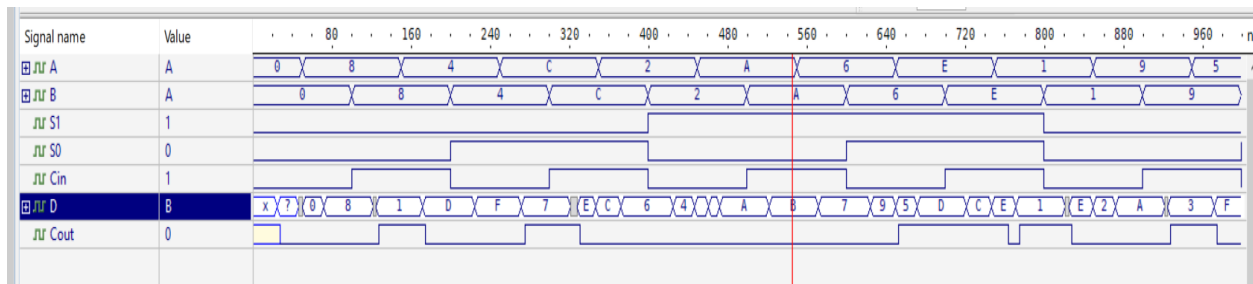
Case 3: when S1,S0,Cin = 010 , then D= A+Bnot (F=4+B+1). And Cout =0.



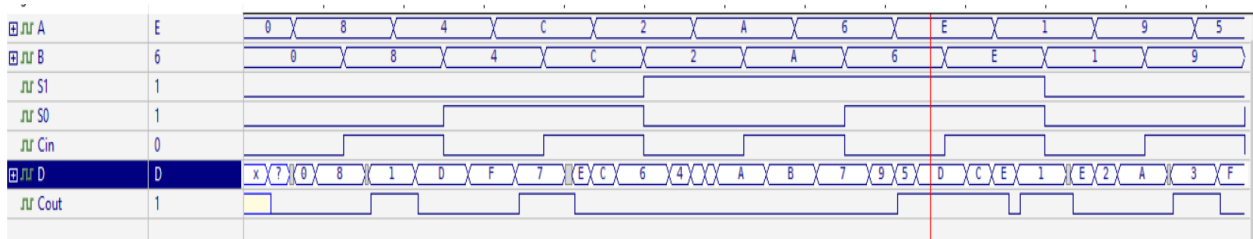
Case 4: when S1,S0,Cin = 011 , then D= A+Bnot+1 (E!=C+3+1). The output must be 0 and carry out =1 . There is a glitch because of the delay of the circuits.



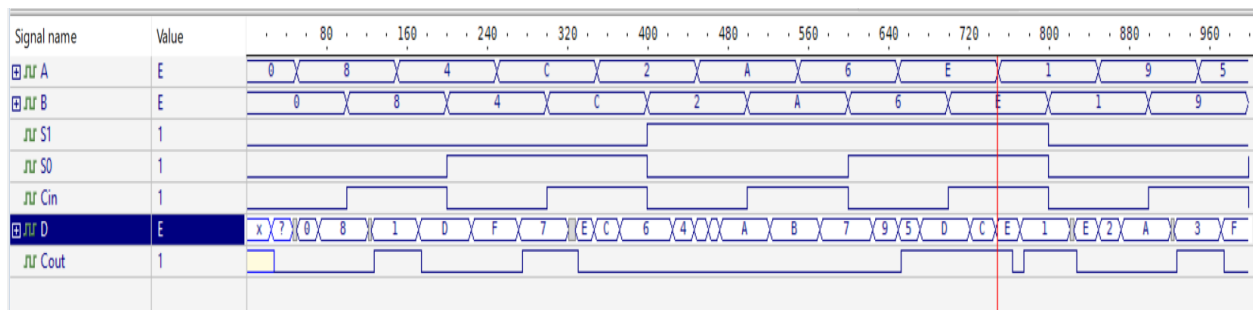
Case 5: when  $S1, S0, Cin = 100$ , then  $D = A$  ( $A=A$ ). And  $Cout = 0$ .



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



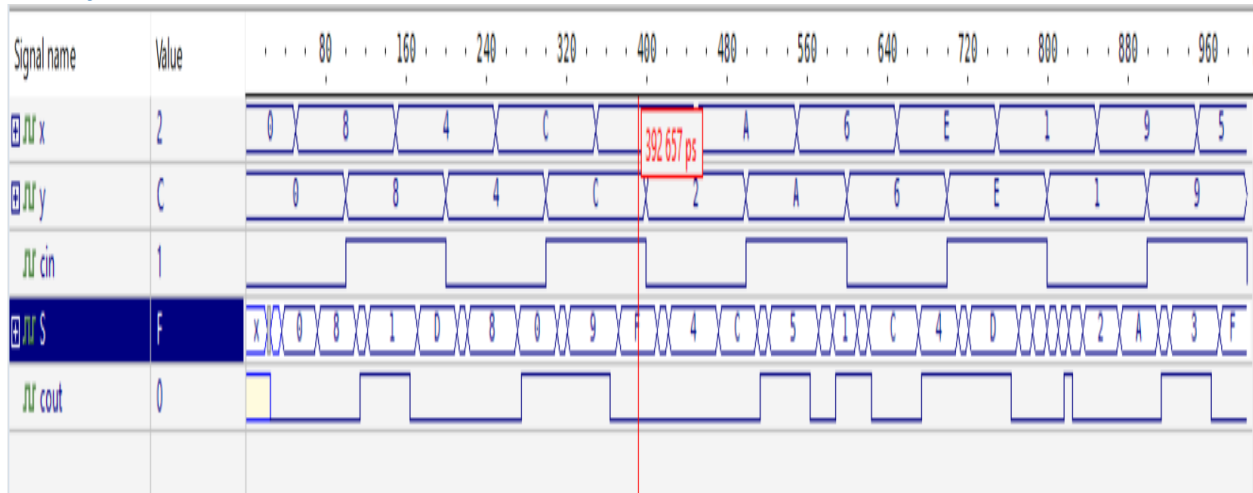
Case 7: when  $S1, S0, Cin = 110$ , then  $D = A-1$  ( $D=E-1$ ). And  $Cout=1$ .



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



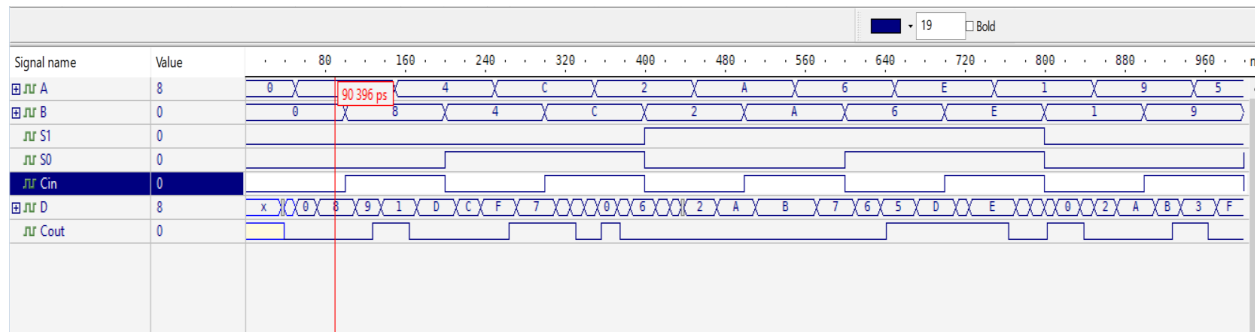
This adder works same as the ripple adder but the difference that the carry look ahead adder is faster than ripple adder but the output is the same.

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

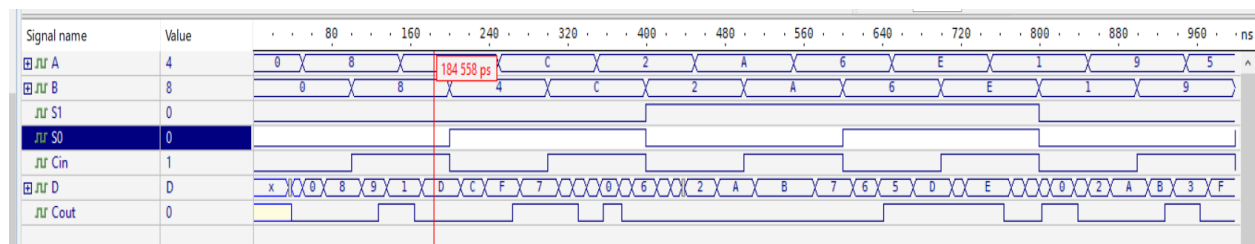
ERRORS:

# KERNAL: ERROR

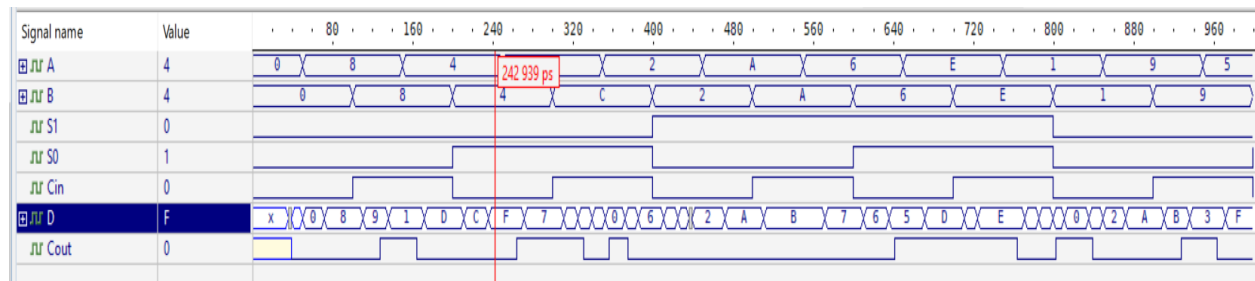
## Arithmetic Unit 2



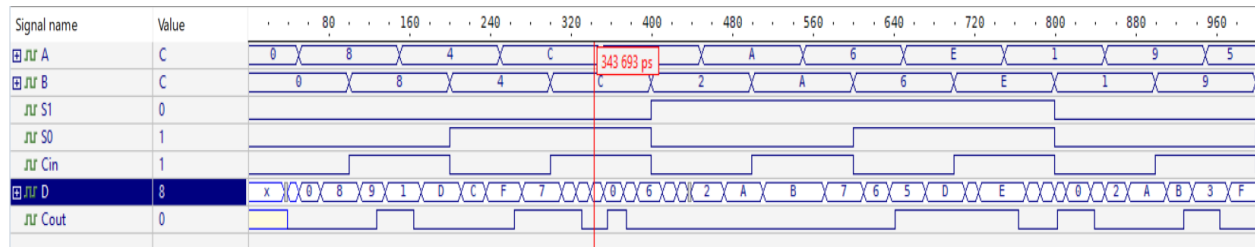
Case 1: when  $S1, S0, Cin = 000$  , then  $D = A + B$  ( $8 = 8 + 0$ ). And  $Cout = 0$ .



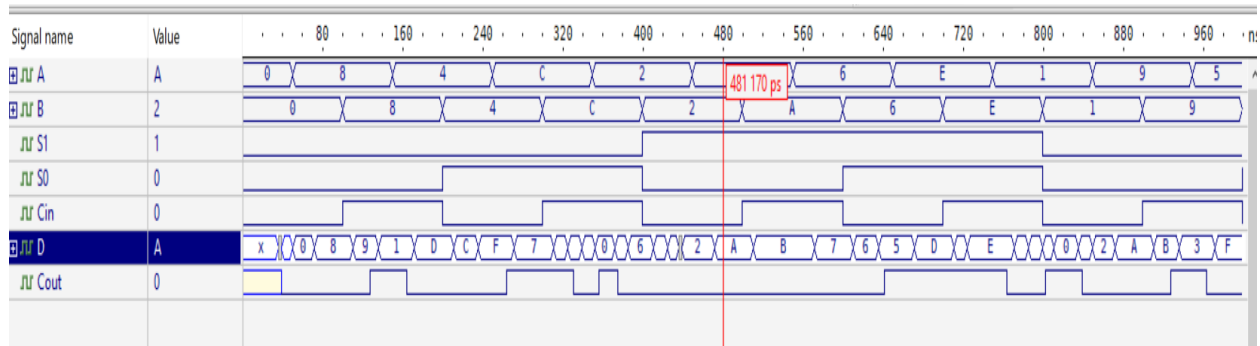
Case 2: when  $S1, S0, Cin = 001$  , then  $D = A + B + 1$  ( $D = 8 + 4 + 1$ ). And  $Cout = 0$ .



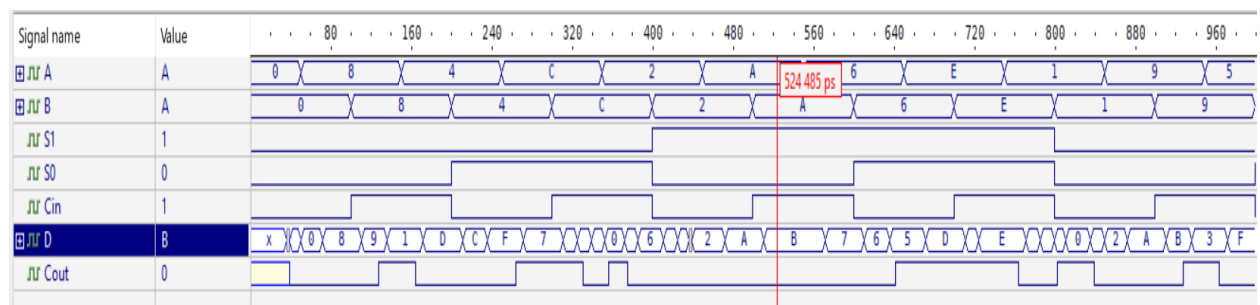
Case 3: when  $S1, S0, Cin = 010$  , then  $D = A + B_{not}$  ( $F = 4 + B$ ). And  $Cout = 0$ .



Case 4: when  $S1, S0, Cin = 011$  , then  $D = A + B_{not} + 1$  ( $8 \neq C + 3 + 1$ ). 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 approximatiy.



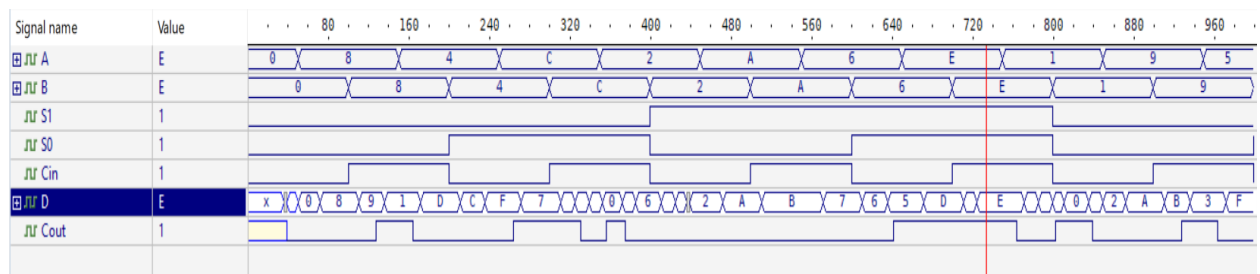
Case 5: when  $\text{S1}, \text{S0}, \text{Cin} = 100$ , then  $\text{D} = \text{A}$  ( $\text{A} = \text{A}$ ). And  $\text{Cout} = 0$ .



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

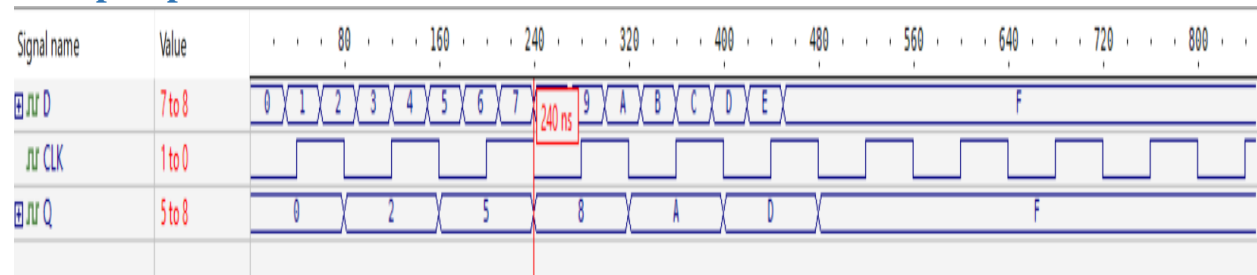


Case 7: when  $\text{S1}, \text{S0}, \text{Cin} = 110$ , then  $\text{D} = \text{A} - 1$  ( $5 = 6 - 1$ ). And  $\text{Cout} = 0$ .



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

## 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/1ANBtBi6OvSOncbuUXnbtLloIJS67WnMr>
- [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'b0000001;  
            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'b0000000000;

            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'b0000000001;

                                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'b000000000000;

            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'b0000000000;

            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'b000000000000;

            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
```