# COMP311-1: Logic Circuit Design
# Fall 2019, Prof. Taigon Song
# Project 2. Due: Dec. 2, 8:59am [Total: 90 + 10 points]

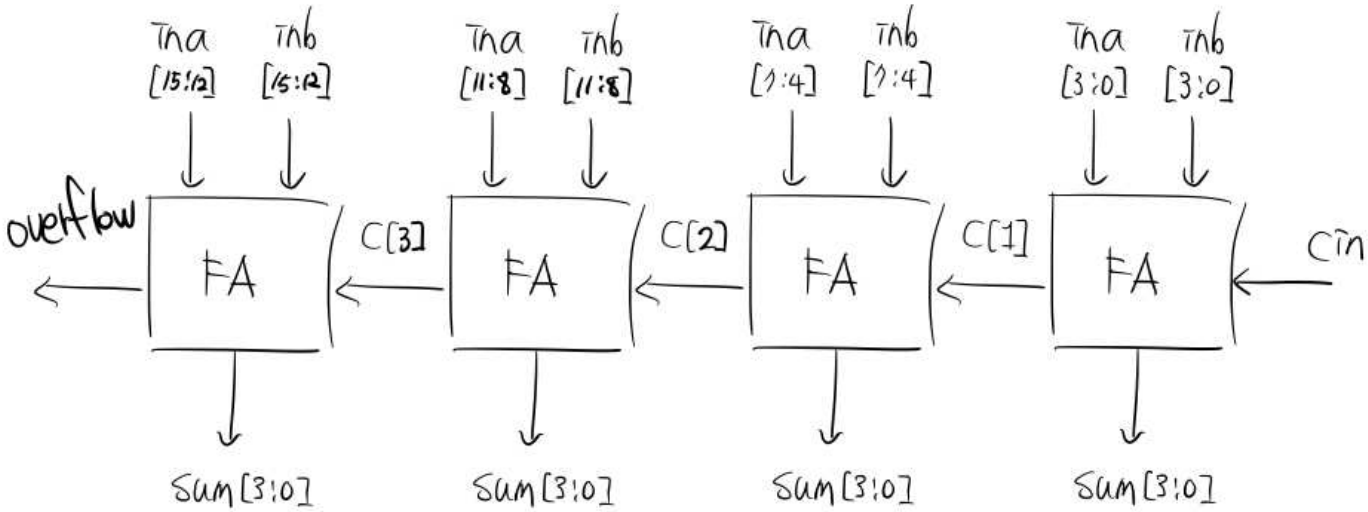
2015112138
Se Jin Kwon

| No. | Checksheet item | Done?[Y/N] |
|---|---|---|
| 1 | RCA adder design (no delay, 10 points) | Y |
| 2 | CLA adder design (no delay, 20 points) | Y |
| 3 | CSA adder design 1 (no delay, 15 points) | Y |
| 4 | CSA adder design 2 (no delay, 5 points) | Y |
| 5 | Four adder design with delay (10 points) | Y |
| 6 | Tradeoff between area and speed (Synthesis report, 10 points) | Y |
| 7 | Delay result/analysis (10 points) | Y |
| 8 | Any other discussion (10 points) | Y |
| 9 | Bonus: Submission date (+ ____ points) | |

# 1. RCA adder design

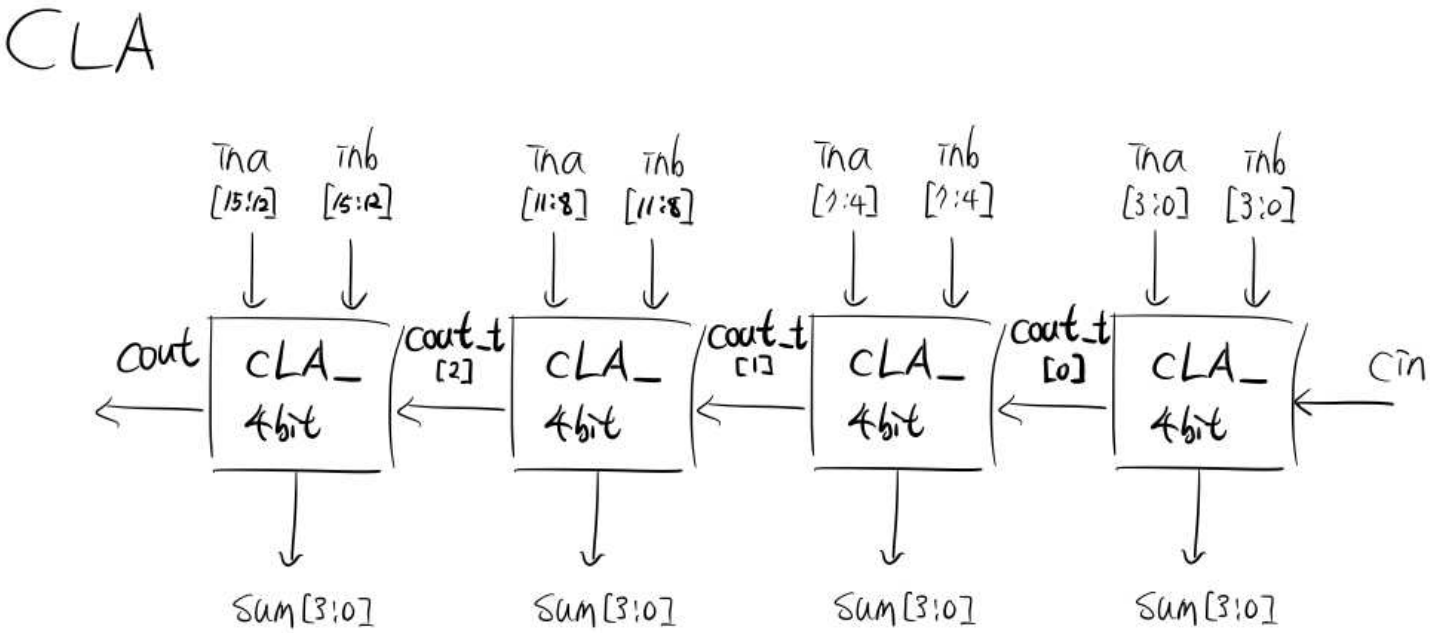
```
module RCA(output [15:0] sum,output cout,input [15:0] ina,inb, input cin);

    wire [4:1] c;
    wire s1,s2;

    fullAdder4 n1(sum[3:0],c[1],ina[3:0],inb[3:0],cin);
    fullAdder4 n2(sum[7:4],c[2],ina[7:4],inb[7:4],c[1]);
    fullAdder4 n3(sum[11:8],c[3],ina[11:8],inb[11:8],c[2]);
    fullAdder4 n4(sum[15:12],cout,ina[15:12],inb[15:12],c[3]);

endmodule
```

RCA.



## 2. CLA adder design

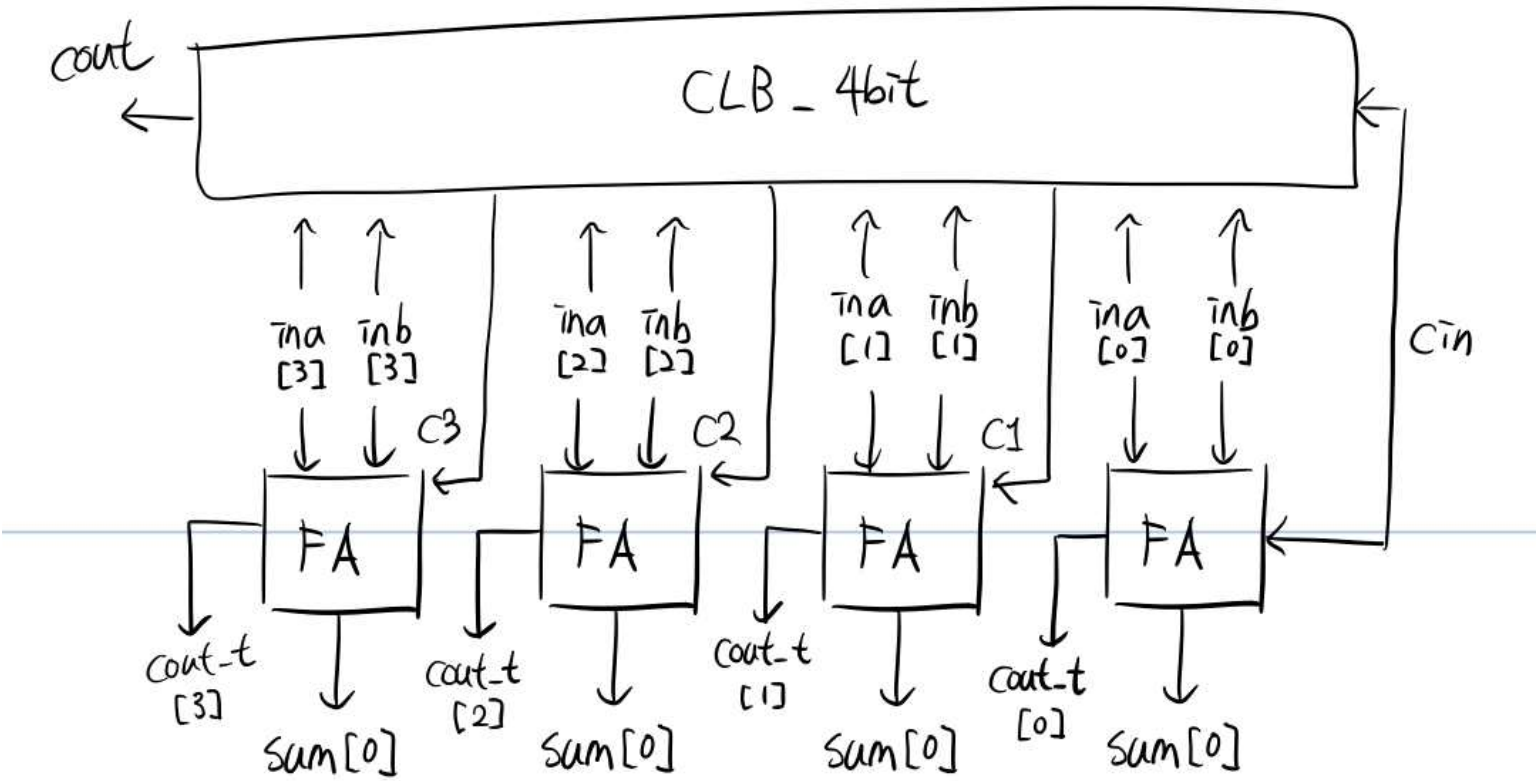
```
module CLA(output [15:0] sum,
        output cout,
        input [15:0] ina,inb,
        input cin
        );

        wire [2:0] cout_t;

        CLA_4bit n1(sum[3:0],cout_t[0],ina[3:0],inb[3:0],cin);
        CLA_4bit n2(sum[7:4],cout_t[1],ina[7:4],inb[7:4],cout_t[0]);
        CLA_4bit n3(sum[11:8],cout_t[2],ina[11:8],inb[11:8],cout_t[1]);
        CLA_4bit n4(sum[15:12],cout,ina[15:12],inb[15:12],cout_t[2]);

endmodule
```




```
module CLA_4bit(
        output [3:0] sum,
        output cout,
        input [3:0] ina,inb,
        input cin
        );

        wire c1,c2,c3;
        wire [3:0] cout_t;

        CLB_4bit n1(c1,c2,c3,cout,ina,inb,cin);
```

```
        fullAdder1  n2(sum[0],cout_t[0],ina[0],inb[0],cin);
        fullAdder1  n3(sum[1],cout_t[1],ina[1],inb[1],c1);
        fullAdder1  n4(sum[2],cout_t[2],ina[2],inb[2],c2);
        fullAdder1  n5(sum[3],cout_t[3],ina[3],inb[3],c3);

endmodule
```

CLA_4bit




```
`timescale  1ns/1ns
module  CLB_4bit(
        output  c1,c2,c3,cout,
        input  [3:0]  ina,inb,
        input  cin
        );

        wire  p0,p1,p2;
        wire  g0,g1,g2;

        wire  c1_t;
        wire  [3:0]  c2_t;
        wire  [7:0]  c3_t;
        wire  [12:0]  c4_t;
```

```
        // g0,p0
        and n1(g0,ina[0],inb[0]);
        xor n2(p0,ina[0],inb[0]);
        // g1,p1
        and n3(g1,ina[1],inb[1]);
        xor n4(p1,ina[1],inb[1]);
        // g2,p2
        and n5(g2,ina[2],inb[2]);
        xor n6(p2,ina[2],inb[2]);
        // g3,p3
        and n7(g3,ina[3],inb[3]);
        xor n8(p3,ina[3],inb[3]);

        //c1
        and k1(c1_t,p0,cin);
        or k2(c1,c1_t,g0);

        //c2
        and (c2_t[0],p1,p0,cin);
        and (c2_t[1],p1,g0);

        or (c2,g1,c2_t[1],c2_t[0]);



        //c3
        and (c3_t[0],p2,p1,p0,cin);
        and (c3_t[1],p2,p1,g0);
        and (c3_t[2],p2,g1);

        or (c3,g2,c3_t[2],c3_t[1],c3_t[0]);

        //cout
        and (c4_t[0],p3,p2,p1,p0,cin);
        and (c4_t[1],p3,p2,p1,g0);
        and (c4_t[2],p3,p2,g1);
        and (c4_t[3],p3,g2);

        or (cout,g3,c4_t[3],c4_t[2],c4_t[1],c4_t[0]);
endmodule
```

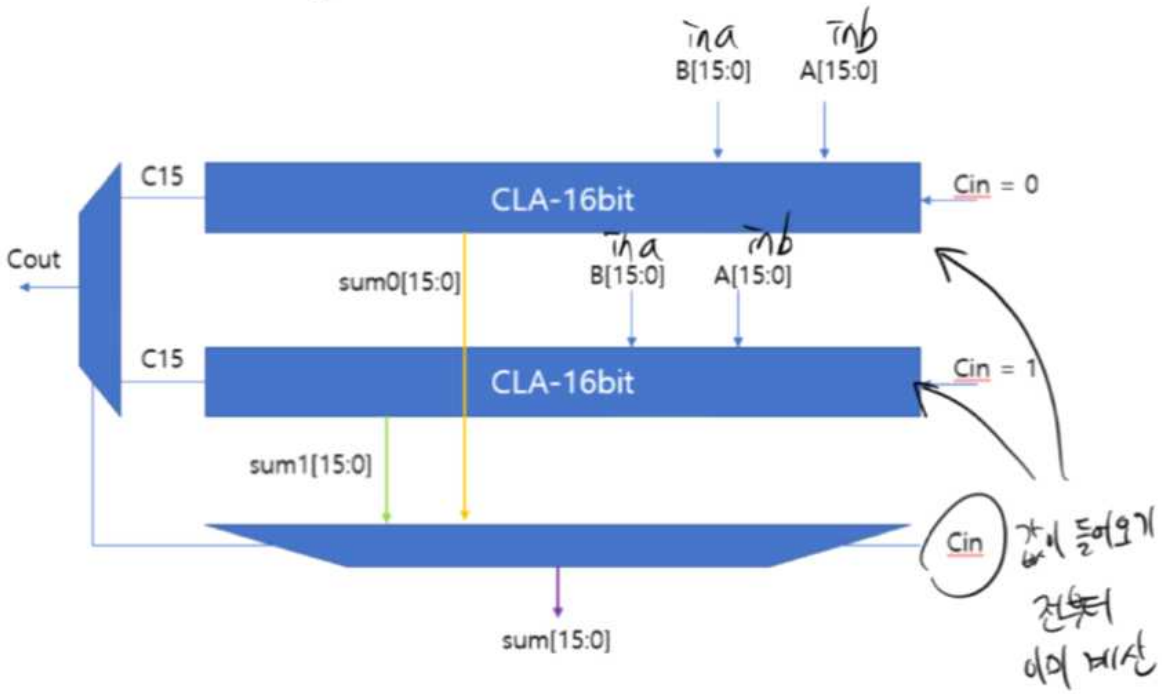
CLB_4bit

- $Gi = \overline{ina}[i] \cdot \overline{inb}[i]$
- $Pi = \overline{ina}[i] \oplus \overline{inb}[i]$
- $C1 = G0 + P0 \cdot Cin$
- $C2 = G1 + P1 \cdot G0 + P1 \cdot P0 \cdot Cin$
- $C3 = G2 + P2 \cdot G1 + P2 \cdot P1 \cdot G0 + P2 \cdot P1 \cdot P0 \cdot Cin$
- $Cout = G3 + P3 \cdot G2 + P3 \cdot P2 \cdot G1 + P3 \cdot P2 \cdot P1 \cdot G0 + P3 \cdot P2 \cdot P1 \cdot P0 \cdot Cin$

## 3. CSA adder design 1

```
module CSA_1(output [15:0] sum, output cout, input [15:0] ina,inb, input cin);

    wire [15:0] sum0;
    wire cout_0;
    wire [15:0] sum1;
    wire cout_1;

    reg cin_0;
    reg cin_1;

    CLA n1(sum0,cout_0,ina,inb,cin_0);
    CLA n2(sum1,cout_1,ina,inb,cin_1);

    MUX21 a0(cout,cout_0,cout_1,cin);

    MUX21 a1(sum,sum0,sum1,cin);


    initial begin
        cin_0 = 0;
        cin_1 = 1;
    end

endmodule
```

CSA - 1

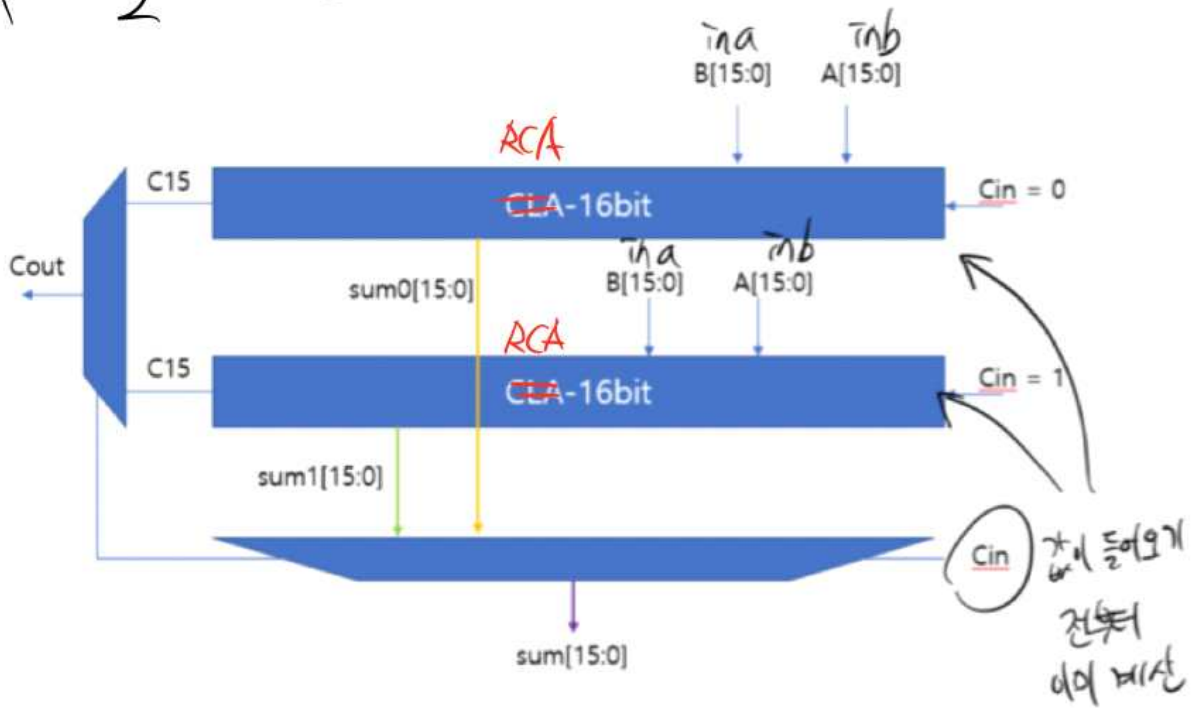



## 4. CSA adder design 2

```
module CSA_2(output [15:0] sum, output cout, input [15:0] ina,inb, input cin);

        wire [15:0] sum0;
        wire cout_0;
        wire [15:0] sum1;
        wire cout_1;

        reg cin_0;
        reg cin_1;

        RCA n1(sum0,cout_0,ina,inb,cin_0);
        RCA n2(sum1,cout_1,ina,inb,cin_1);

        MUX21 a0(cout,cout_0,cout_1,cin);

        MUX21 a1(sum,sum0,sum1,cin);


        initial begin
                cin_0 = 0;
```

```
            cin_1 = 1;
        end

endmodule
```

CSA-2



# 5. Four adder design with delay

## 1. RCA with delay

we need to change only “fullAdder1” module.

```
`timescale 1ns/1ns
module fullAdder1(output sum,c_out, input a,b,c_in);
        wire s1,c1,s2;

        xor #(6) n1(s1,a,b);
        and #(7) n2(c1,a,b);
        xor #(6) n3(sum,s1,c_in);
        and #(7) n4(s2,s1,c_in);
        xor #(6) n5(c_out,s2,c1);

endmodule
```

## 2. CLA with delay

we need to change only “CLB_4bit” module.

```
`timescale 1ns/1ns
module CLB_4bit(
        output c1,c2,c3,cout,
        input [3:0] ina,inb,
        input cin
        );

        wire p0,p1,p2;
        wire g0,g1,g2;

        wire c1_t;
        wire [3:0] c2_t;
        wire [7:0] c3_t;
        wire [12:0] c4_t;



        // g0,p0
        and #(7) n1(g0,ina[0],inb[0]);
        xor #(6) n2(p0,ina[0],inb[0]);
        // g1,p1
        and #(7) n3(g1,ina[1],inb[1]);
        xor #(6) n4(p1,ina[1],inb[1]);
        // g2,p2
        and #(7) n5(g2,ina[2],inb[2]);
        xor #(6) n6(p2,ina[2],inb[2]);
```

```
        // g3,p3
        and #(7) n7(g3,ina[3],inb[3]);
        xor #(6) n8(p3,ina[3],inb[3]);

        //c1
        and #(7) k1(c1_t,p0,cin);
        or #(7) k2(c1,c1_t,g0);

        //c2
        and #(7) (c2_t[0],p1,p0,cin);
        and #(7) (c2_t[1],p1,g0);

        or #(7) (c2,g1,c2_t[1],c2_t[0]);



        //c3
        and #(7) (c3_t[0],p2,p1,p0,cin);
        and #(7) (c3_t[1],p2,p1,g0);
        and #(7) (c3_t[2],p2,g1);

        or #(7) (c3,g2,c3_t[2],c3_t[1],c3_t[0]);

        //cout
        and #(7) (c4_t[0],p3,p2,p1,p0,cin);
        and #(7) (c4_t[1],p3,p2,p1,g0);
        and #(7) (c4_t[2],p3,p2,g1);
        and #(7) (c4_t[3],p3,g2);

        or #(7) (cout,g3,c4_t[3],c4_t[2],c4_t[1],c4_t[0]);


endmodule
```

### 3. CSA_1 with delay

Just add MUX delay to CSA_1 module

```
module CSA_1(output [15:0] sum, output cout, input [15:0] ina,inb, input cin);

        wire [15:0] sum0;
        wire cout_0;
        wire [15:0] sum1;
        wire cout_1;

        reg cin_0;
        reg cin_1;

        CLA n1(sum0,cout_0,ina,inb,cin_0);
```

```
        CLA n2(sum1,cout_1,ina,inb,cin_1);

        MUX21 #(3) a0(cout,cout_0,cout_1,cin);

        MUX21 #(3) a1(sum,sum0,sum1,cin);


        initial begin
                cin_0 = 0;
                cin_1 = 1;
        end

endmodule
```

4. CSA_2 with delay

Just add MUX delay to CSA_2 module

```
module CSA_2(output [15:0] sum, output cout, input [15:0] ina,inb, input cin);

        wire [15:0] sum0;
        wire cout_0;
        wire [15:0] sum1;
        wire cout_1;

        reg cin_0;
        reg cin_1;

        RCA n1(sum0,cout_0,ina,inb,cin_0);
        RCA n2(sum1,cout_1,ina,inb,cin_1);

        MUX21 #(3) a0(cout,cout_0,cout_1,cin);

        MUX21 #(3) a1(sum,sum0,sum1,cin);


        initial begin
                cin_0 = 0;
                cin_1 = 1;
        end

endmodule
```

=> reference

MUX design

```
`timescale 1ns/1ns
module MUX21 #(parameter delay=0) (output reg [15:0] sum,input [15:0] in0, in1, input sel);

        always @(*) begin
                #delay
                case (sel)
                        1'b0:sum = in0;
                        1'b1:sum = in1;
                endcase
        end

endmodule
```

# 6. Tradeoff between area and speed

---------- how fast

1. RCA

   XOR => 48 units
   AND => 32 units
   ---
   80 unit ---------- slowest (4)

2. CLA

   AND => 88 units
   XOR => 64 units
   OR => 16 units
   ---
   168 units ---------- fast (2)

3. CSA_1

   AND => 176 units
   XOR => 128 units
   OR => 32 units
   ---
   336 units ---------- fastest (1)

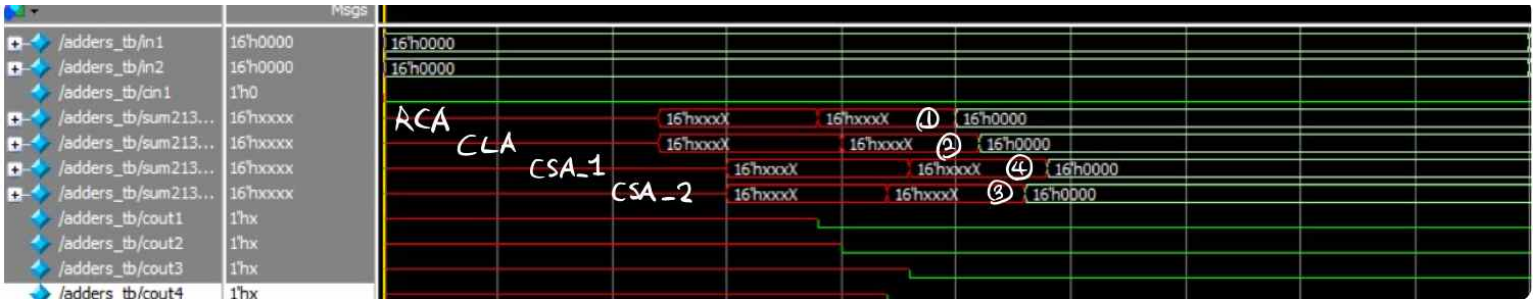
4. CSA_2

   XOR => 96 units
   AND => 64 units
   ---
   160 units ---------- slow (3)

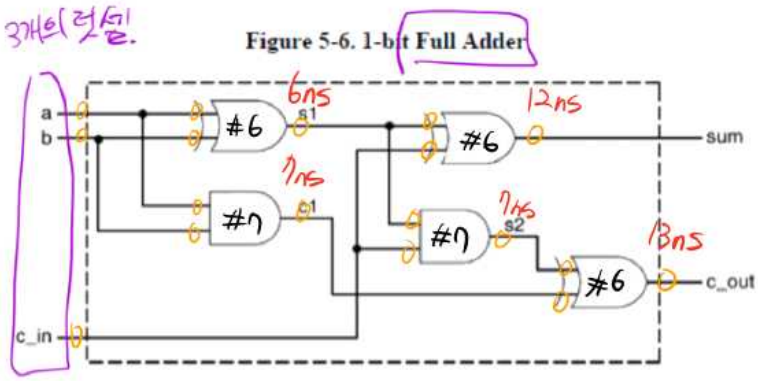
we can know tradeoff between area and speed
by correlation between above gate unit calculation and speed
The more gates the adder has, the better the performance of the adder.

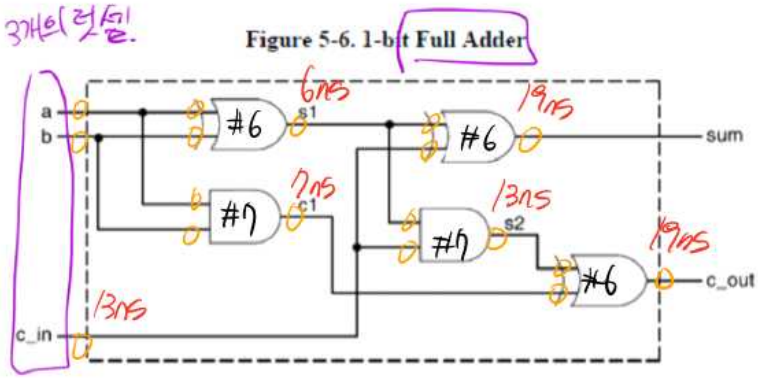
## 7. Delay result/analysis

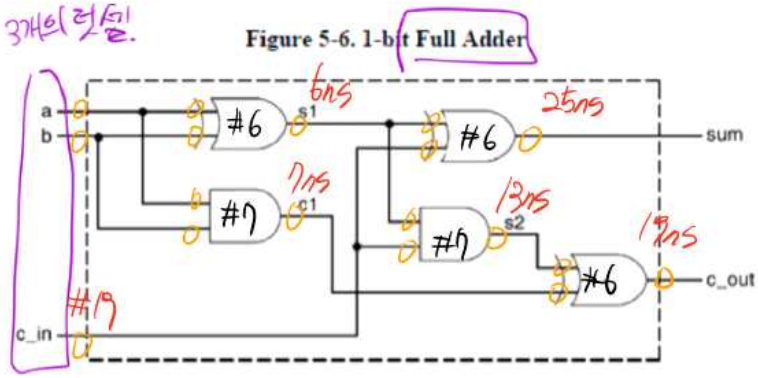
1)



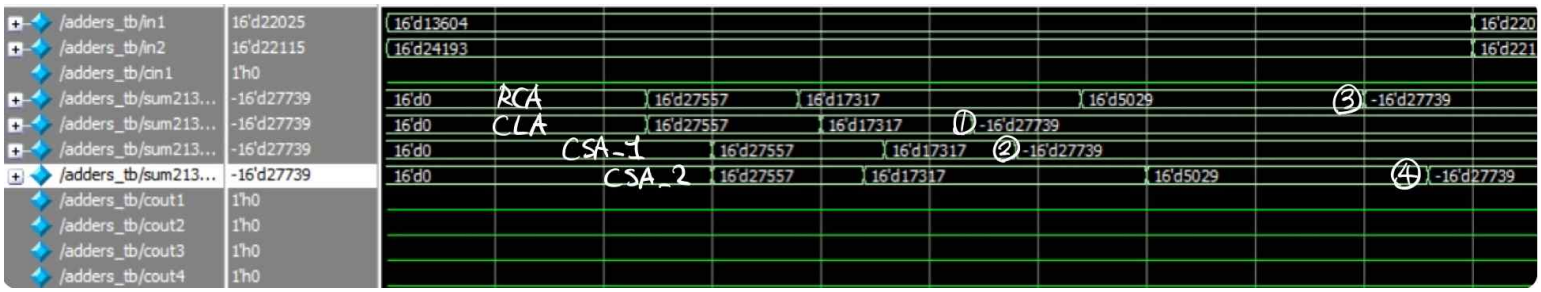

The reason why the RCA is fastest module is that
the result of AND gates can be decided by just one input.



2)




this part is the result that we expect.

fastest => CLA

fast => CSA_1

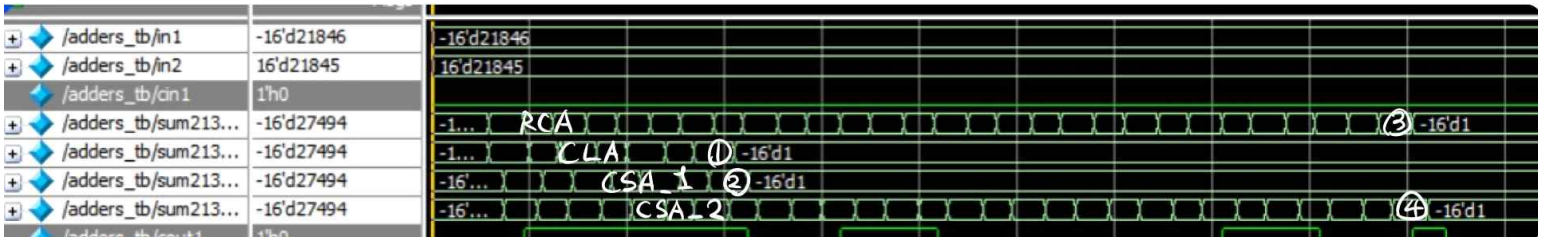
slow => RCA

slowest => CSA_2

At this time, speed of CSA is slower than CLA or RCA.

but CSA is much better than CLA or RCA when the cin1 changes suddenly.

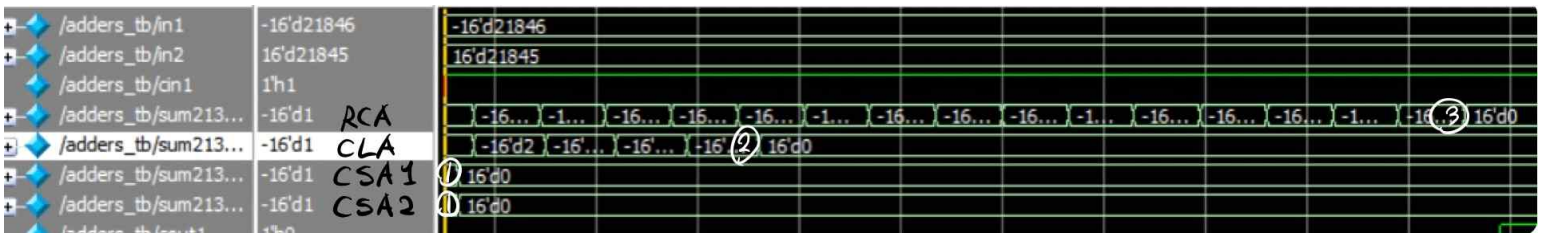
3)




The more difficult the AND gate is to predict the result,

the larger the performance difference between CLA and RCA.

4)




CSA1 and CSA2 show tremendous performance when the value of cin1 suddenly changes

because they calculate the value in advance.

## 8. Any other discussion

### 1. The reason Why RCA is the slowest adder

=> Since RCA always has to do the calculations in order, you have to wait until the adder in the previous order completes the calculation.

### 2. The reason Why CLA is faster than RCA

=> Because we calculate all carry at once and add all at once.

### 3. The reason why we use CSA

=> CSA calculates all situations of cin input in advance,
which is very useful when the cin input suddenly changes.

### 4. The reason why CSA-2 is slower than CSA-1

=>CSA-2 uses RCA by default, so it's slower than CSA-1 using CLA.

### 5. When one input of an AND gate inside the RSA is zero, the performance of the RSA is greatly improved.

This is because the delay is applied if the result can be specified regardless of the change in the input.