DIGITAL DESIGN

ASSIGNMENT REPORT

ASSIGNMENT ID : 1

Student Name: 刘乐奇

Student ID: 12011327

PART 1: DIGITAL DESIGN THEORY

Provide your answers here:

1. (a) 1 6384 bytes

   (b) 3355 4432 bytes

   (c) 34 3597 3836.8 bytes

2. Without sign: In binary: $(1111\ 1111\ 1111)_2$

   In decimal: $(4095)_{10}$

   In hexadecimal: $(fff)_{16}$

   With sign: In binary: $(0111\ 1111\ 1111)_2$

   In decimal: $(2047)_{10}$

   In hexadecimal: $(7ff)_{16}$

3. (a) $(248)_{10} = (1111\ 1000)_2$

   (b) $(248)_{10} = (f8)_{16} = (1111\ 1000)_2$

   I think method (b) may be faster. Because when we divide the same number but by a larger number, it will cost less time. In the meanwhile, when convert hexadecimal to binary, we can convert 4-bit a time.

4. a. 9's complement: $(747269063)_{10}$

   10's complement: $(747269064)_{10}$

   b. 9's complement: $(35677389)_{10}$

   10's complement: $(35677390)_{10}$

5. (a) $(3941)_{16}$

(b) $(1100\ 0110\ 1011\ 1111)_2$

(c) $(0011\ 1001\ 0100\ 0001)_2$

(d) $(3941)_{16}$ which is the same as the result in (a)

6. (a) $(10111.1010111111001)_2$

(b) $(5/3)_{10} = (1.10101010)_2$ with 8 precisions

$(1.10101010)_2 = (1.6640625)_{10}$ which has the difference $(1.6666666)_{10}$ - $(1.6640625)_{10} = (0.0026041)_{10}$

(c) $(1.10101010)_2 = (1.aa)_{16} = (1.6640625)_{10}$ whose result is the same as directly converted to decimal. Because hexadecimal is equivalent as binary in some degree. When converting binary to hexadecimal, we usually do conversion from every 4 bits in binary to 1 bit in hexadecimal, vice versa.

7. (a) $(975)_{BCD}$

(b) $(642)_{excess-3}$

(c) $(713)_{84-2-1}$

(d) $(754)_{6311}$

(e) $(1001\ 0111\ 0101)_2 = (2421)_{10}$

8. (a) A AND B = $(0100\ 1010)_2 = (4a)_{16}$

(b) A OR B = $(1101\ 1110)_2 = (de)_{16}$

(c) A XOR B = $(1001\ 0100)_2 = (94)_{16}$

(d) NOT A = $(0010\ 0101)_2 = (25)_{16}$

(e) NOT B = $(1011\ 0001)_2 = (b1)_{16}$

(f) A NAND B = $(1011\ 0101)_2 = (b5)_{16}$

(g) A NOR B = $(0010\ 0001)_2 = (21)_{16}$

PART 2: DIGITAL DESIGN LAB (TASK1)

DESIGN

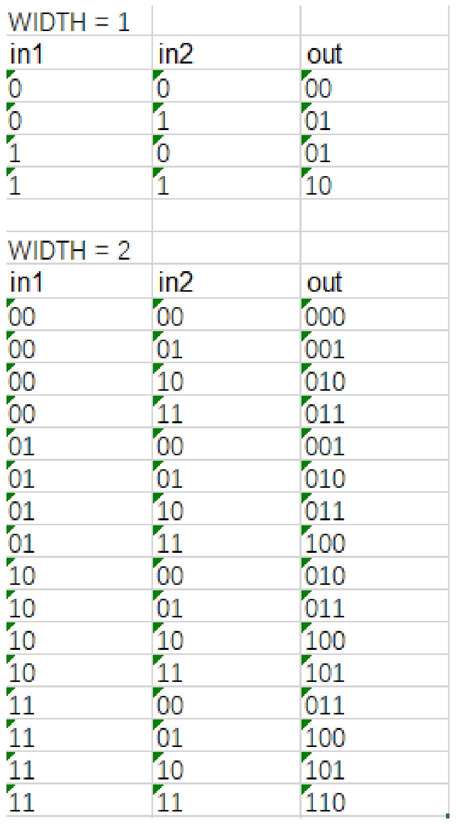
Describe the design of your system by providing the following information:

- ***Verilog design (provide the Verilog code)***

//UnisignedAddition.v

```
`timescale 1ns / 1ps


module UnsignedAddition(in1,in2,out);
    parameter   WIDTH=2;
    input [WIDTH-1:0] in1;
    input [WIDTH-1:0] in2;
    output [WIDTH:0] out;


    wire [WIDTH:0] c1, c2, tmp1, tmp2;


    assign tmp1 = in1^in2;
    assign c1 = in1&in2;
    assign tmp2 = tmp1^(c1<<1);
    assign c2 = tmp1&(c1<<1);
    assign out = tmp2^(c2<<1);


endmodule
```

- ***Truth-table***



WIDTH = 1		
in1	in2	out
0	0	00
0	1	01
1	0	01
1	1	10

WIDTH = 2		
in1	in2	out
00	00	000
00	01	001
00	10	010
00	11	011
01	00	001
01	01	010
01	10	011
01	11	100
10	00	010
10	01	011
10	10	100
10	11	101
11	00	011
11	01	100
11	10	101
11	11	110

SIMULATION

Describe how you build the test bench and do the simulation.

- ***Using Verilog (provide the Verilog code)***

```
//UnisignedAddition_sim.v
    `timescale 1ns / 1ps


    module UnsignedAddition_sim();
        reg [0:0] in1_1_sim,in2_1_sim;
        wire [1:0] out_1_sim;


        reg [1:0] in1_2_sim,in2_2_sim;
```

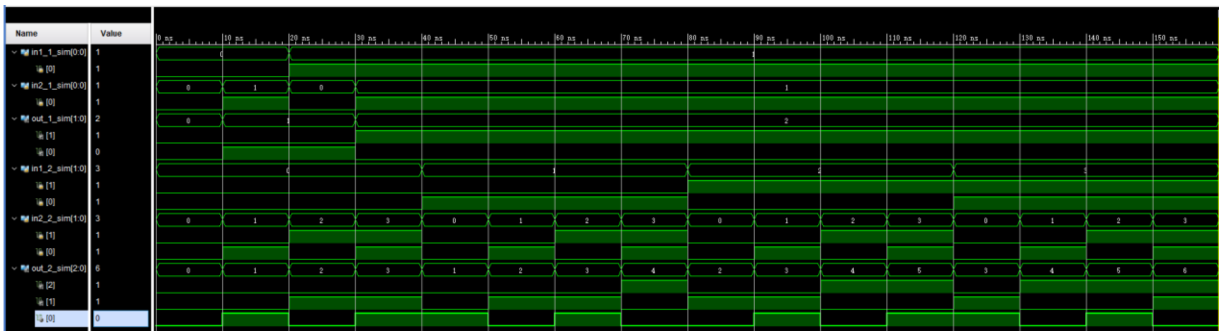
```
wire [2:0] out_2_sim;

UnsignedAddition #(2) ub(
.in1(in1_2_sim),
.in2(in2_2_sim),
.out(out_2_sim)
);

  UnsignedAddition #(1) ua(
.in1(in1_1_sim),
.in2(in2_1_sim),
.out(out_1_sim)
);

initial begin
in1_2_sim = 2'b00; in2_2_sim = 2'b00; in1_1_sim = 1'b0; in2_1_sim =
1'b0;
#10
in1_2_sim = 2'b00; in2_2_sim = 2'b01; in1_1_sim = 1'b0; in2_1_sim =
1'b1;
#10
in1_2_sim = 2'b00; in2_2_sim = 2'b10; in1_1_sim = 1'b1; in2_1_sim =
1'b0;
#10
in1_2_sim = 2'b00; in2_2_sim = 2'b11; in1_1_sim = 1'b1; in2_1_sim =
1'b1;
#10
in1_2_sim = 2'b01; in2_2_sim = 2'b00;
```

```
    #10
    in1_2_sim = 2'b01; in2_2_sim = 2'b01;
    #10
    in1_2_sim = 2'b01; in2_2_sim = 2'b10;
    #10
    in1_2_sim = 2'b01; in2_2_sim = 2'b11;
    #10
    in1_2_sim = 2'b10; in2_2_sim = 2'b00;
    #10
    in1_2_sim = 2'b10; in2_2_sim = 2'b01;
    #10
    in1_2_sim = 2'b10; in2_2_sim = 2'b10;
    #10
    in1_2_sim = 2'b10; in2_2_sim = 2'b11;
    #10
    in1_2_sim = 2'b11; in2_2_sim = 2'b00;
    #10
    in1_2_sim = 2'b11; in2_2_sim = 2'b01;
    #10
    in1_2_sim = 2'b11; in2_2_sim = 2'b10;
    #10
    in1_2_sim = 2'b11; in2_2_sim = 2'b11;
    #10
    $finish();
    end

endmodule
```

- ***Wave form of simulation result (provide screen shots)***

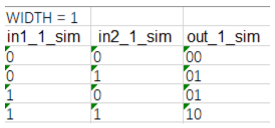
For width = 1, input in1_1_sim, in2_1_sim, ouput out_1_sim;

For width = 2, input in1_2_sim, in2_2_sim, ouput out_2_sim;



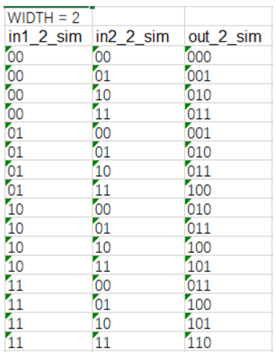
The description on whether the simulation result is same as the truth-table, is the function of the design meet the expectation.

We can know from the waveform that when the width is 1, the result is in the below:



WIDTH = 1		
in1_1_sim	in2_1_sim	out_1_sim
0	0	00
0	1	01
1	0	01
1	1	10

When the width is 2, the result is in the below:



WIDTH = 2		
in1_2_sim	in2_2_sim	out_2_sim
00	00	000
00	01	001
00	10	010
00	11	011
01	00	001
01	01	010
01	10	011
01	11	100
10	00	010
10	01	011
10	10	100
10	11	101
11	00	011
11	01	100
11	10	101
11	11	110

Comparing the above results with truth-table, the simulation result is well matched. The function I designed well meets the expectation.

THE DESCRIPTION OF OPERATION

Describe the problem occurred while in the lab and your solution. Any suggestions are welcomed.

- ***Problems and solutions***

(1) Different with other IDE such as IDEA, pycharm, etc., the error in Vivado is more hard to find. Once I wrote “Endmodule” instead of “endmodule”, I could not find that though Vivado had warned me because of the obscure indicator.

(2) Initially, I had used “+” for addition. However, I thought it might be simple so that I changed my code to current one, which can hold on at most 2-bit unsigned addition by using bitwise logic operation.

(3) To simplify my code, I used parameter WIDTH to control the width of input.

PART 2: DIGITAL DESIGN LAB (TASK2)

DESIGN

Describe the design of your system by providing the following information:

- ***Verilog design while using data flow (provide the Verilog code)***

```
//distributive1bit_df.v
        `timescale 1ns / 1ps


        module distributive1bit_df(a,b,c,outa1,outa2,outb1,outb2);
            input a,b,c;
            output outa1,outa2, outb1,outb2;

            ///to verify (a)
            wire xa,ya,za;

            assign xa = b|c;
```

```
                assign outa1 = a&xa;

                assign ya = a&b;
                assign za = a&c;
                assign outa2 = ya|za;

                //to verify (b)
                wire xb,yb,zb;

                assign xb = b&c;
                assign outb1 = a|xb;

                assign yb = a|b;
                assign zb = a|c;
                assign outb2 = yb&zb;

endmodule
```

//distributive2bit_df.v

```
        `timescale 1ns / 1ps

      module distributive2bit_df(a,b,c,outa1,outa2,outb1,outb2);
                input [1:0] a,b,c;
                output [1:0] outa1,outa2,outb1,outb2;

                //to verify (a)
                wire [1:0] xa,ya,za;
```

```
            assign xa = b|c;
            assign outa1 = a&xa;

            assign ya = a&b;
            assign za = a&c;
            assign outa2 = ya|za;

            //to verify (b)
            wire [1:0] xb,yb,zb;

            assign xb = b&c;
            assign outb1 = a|xb;

            assign yb = a|b;
            assign zb = a|c;
            assign outb2 = yb&zb;

endmodule
```

- ***Verilog design while using structured design (provide the Verilog code)***

```
//distributive1bit_sd.v
   `timescale 1ns / 1ps

        module distributive1bit_sd(a,b,c,outa1,outa2,outb1,outb2);
            input a,b,c;
            output outa1,outa2,outb1,outb2;

            //to verify (a)
            wire xa,ya,za;

            or ora_1(xa,b,c);
            and anda_1(outa1,a,xa);

            and anda_2(ya,a,b);
            and anda_3(za,a,c);
            or ora_2(outa2,ya,za);

            //to verify (b)
            wire xb,yb,zb;

            and andb_1(xb,b,c);
            or orb_1(outb1,a,xb);

            or orb_2(yb,a,b);
            or orb_3(zb,a,c);
            and andb_2(outb2,yb,zb);

endmodule
```

//distributive2bit_sd.v

```
          `timescale 1ns / 1ps

        module distributive2bit_sd(a,b,c,outa1,outa2,outb1,outb2);
            input [1:0] a,b,c;
            output [1:0] outa1,outa2,outb1,outb2;
```

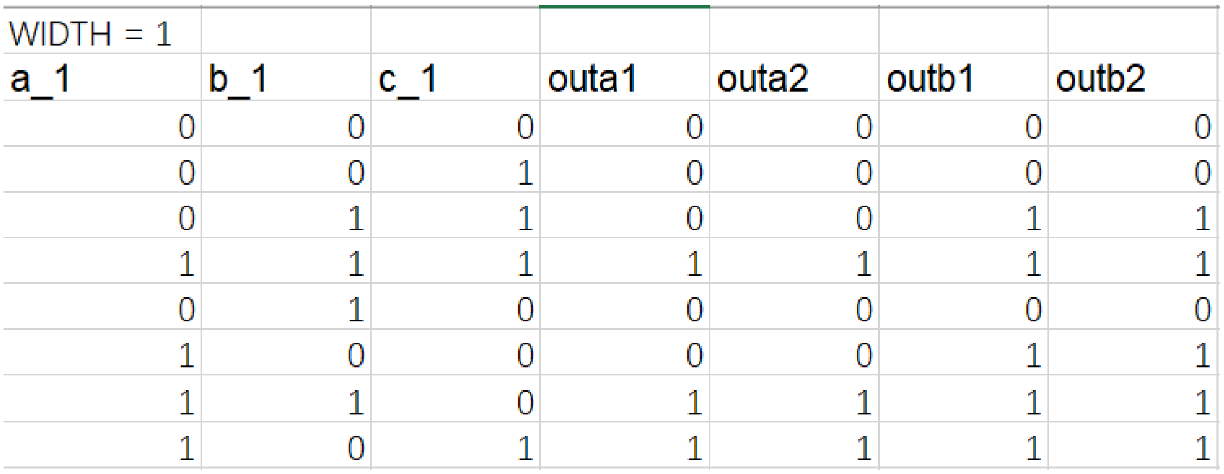
```
//to verify (a)
wire [1:0] xa,ya,za;

or ora_1_1(xa[0],b[0],c[0]);
or ora_1_2(xa[1],b[1],c[1]);
and anda_1_1(outa1[0],a[0],xa[0]);
and anda_1_2(outa1[1],a[1],xa[1]);

and anda_2_1(ya[0],a[0],b[0]);
and anda_2_2(ya[1],a[1],b[1]);
and anda_3_1(za[0],a[0],c[0]);
and anda_3_2(za[1],a[1],c[1]);
or ora_2_1(outa2[0],ya[0],za[0]);
or ora_2_2(outa2[1],ya[1],za[1]);

//to verify (b)
wire [1:0] xb,yb,zb;

and andb_1_1(xb[0],b[0],c[0]);
and andb_1_2(xb[1],b[1],c[1]);
or orb_1_1(outb1[0],a[0],xb[0]);
or orb_1_2(outb1[1],a[1],xb[1]);

or orb_2_1(yb[0],a[0],b[0]);
or orb_2_2(yb[1],a[1],b[1]);
or orb_3_1(zb[0],a[0],c[0]);
or orb_3_2(zb[1],a[1],c[1]);
and andb_2_1(outb2[0],yb[0],zb[0]);
```

and andb_2_2(outb2[1],yb[1],zb[1]);

endmodule

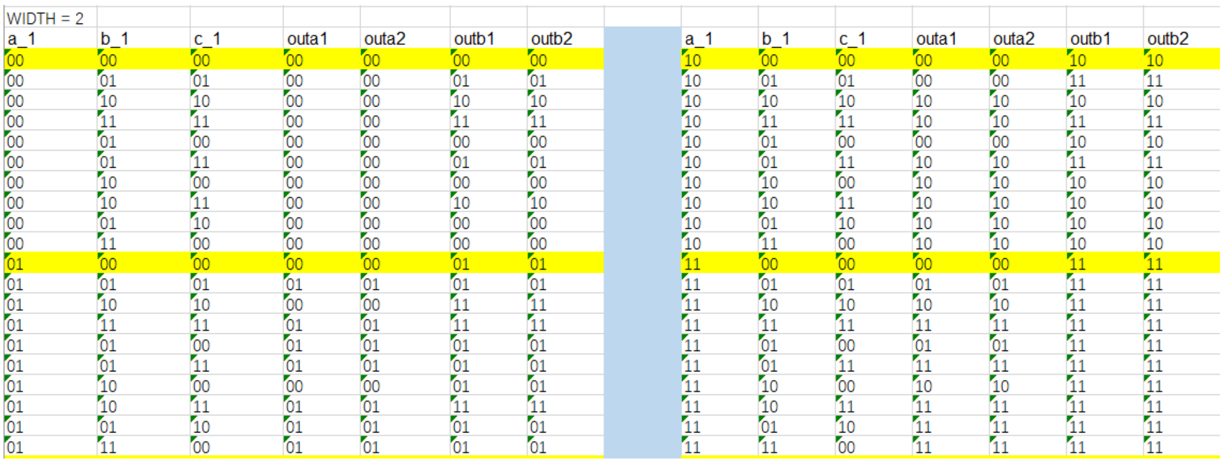
- ***Truth-table***

For 1-bit input, outa1 = A(B+C), outa2 = AB+AC, outb1 = A+BC, outb2 = (A+B)(A+C)



WIDTH = 1						
a_1	b_1	c_1	outa1	outa2	outb1	outb2
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	1	0	0	1	1
1	1	1	1	1	1	1
0	1	0	0	0	0	0
1	0	0	0	0	1	1
1	1	0	1	1	1	1
1	0	1	1	1	1	1

For 2-bit input, outa1 = A(B+C), outa2 = AB+AC, outb1 = A+BC, outb2 = (A+B)(A+C)



WIDTH = 2														
a_1	b_1	c_1	outa1	outa2	outb1	outb2		a_1	b_1	c_1	outa1	outa2	outb1	outb2
00	00	00	00	00	00	00		10	00	00	00	00	10	10
00	01	01	00	00	01	01		10	01	01	00	00	11	11
00	10	10	00	00	10	10		10	10	10	10	10	10	10
00	11	11	00	00	11	11		10	11	11	10	10	11	11
00	01	00	00	00	00	00		10	01	00	00	00	10	10
00	01	11	00	00	01	01		10	01	11	10	10	11	11
00	10	00	00	00	00	00		10	10	00	10	10	10	10
00	10	11	00	00	10	10		10	10	11	10	10	10	10
00	01	10	00	00	00	00		10	01	10	10	10	10	10
00	11	00	00	00	00	00		10	11	00	10	10	10	10
01	00	00	00	00	01	01		11	00	00	00	00	11	11
01	01	01	01	01	01	01		11	01	01	01	01	11	11
01	10	10	00	00	11	11		11	10	10	10	10	11	11
01	11	11	01	01	11	11		11	11	11	11	11	11	11
01	01	00	01	01	01	01		11	01	00	01	01	11	11
01	01	11	01	01	01	01		11	01	11	11	11	11	11
01	10	00	00	00	01	01		11	10	00	10	10	11	11
01	10	11	01	01	11	11		11	10	11	11	11	11	11
01	01	10	01	01	01	01		11	01	10	11	11	11	11
01	11	00	01	01	01	01		11	11	00	11	11	11	11

It is clearly that outa1 == outa2, outb1 == outb2, no matter whether it is 1-bit or 2-bit.

SIMULATION

Describe how you build the test bench and do the simulation.

- ***Using Verilog (provide the Verilog code)***

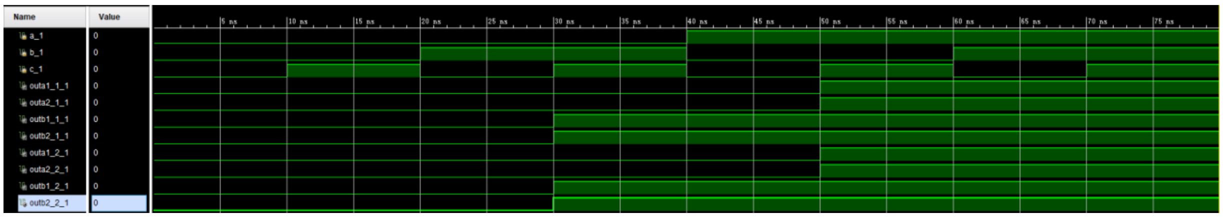
```
//distributive_sim.v
       `timescale 1ns / 1ps

      module distributive_sim();
      //test for df-1-bit
          reg a_1,b_1,c_1;
          wire outa1_1_1,outa2_1_1,outb1_1_1,outb2_1_1;

          distributive1bit_df d1(
              .a(a_1),
              .b(b_1),
              .c(c_1),
              .outa1(outa1_1_1),
              .outa2(outa2_1_1),
              .outb1(outb1_1_1),
              .outb2(outb2_1_1)
              );
      //test for sd-1-bit
           wire outa1_2_1,outa2_2_1,outb1_2_1,outb2_2_1;

           distributive1bit_sd s1(
                .a(a_1),
                .b(b_1),
                .c(c_1),
                .outa1(outa1_2_1),
                .outa2(outa2_2_1),
                .outb1(outb1_2_1),
                .outb2(outb2_2_1)
```

```
                    );
//test for df-2-bit
        reg [1:0] a_2,b_2,c_2;
        wire [1:0] outa1_1_2,outa2_1_2,outb1_1_2,outb2_1_2;

          distributive2bit_df d2(
                .a(a_2),
                .b(b_2),
                .c(c_2),
                .outa1(outa1_1_2),
                .outa2(outa2_1_2),
                .outb1(outb1_1_2),
                .outb2(outb2_1_2)
                );

//test for sd-2-bit
        wire [1:0] outa1_2_2,outa2_2_2,outb1_2_2,outb2_2_2;

          distributive2bit_sd s2(
                .a(a_2),
                .b(b_2),
                .c(c_2),
                .outa1(outa1_2_2),
                .outa2(outa2_2_2),
                .outb1(outb1_2_2),
                .outb2(outb2_2_2)
                );
```

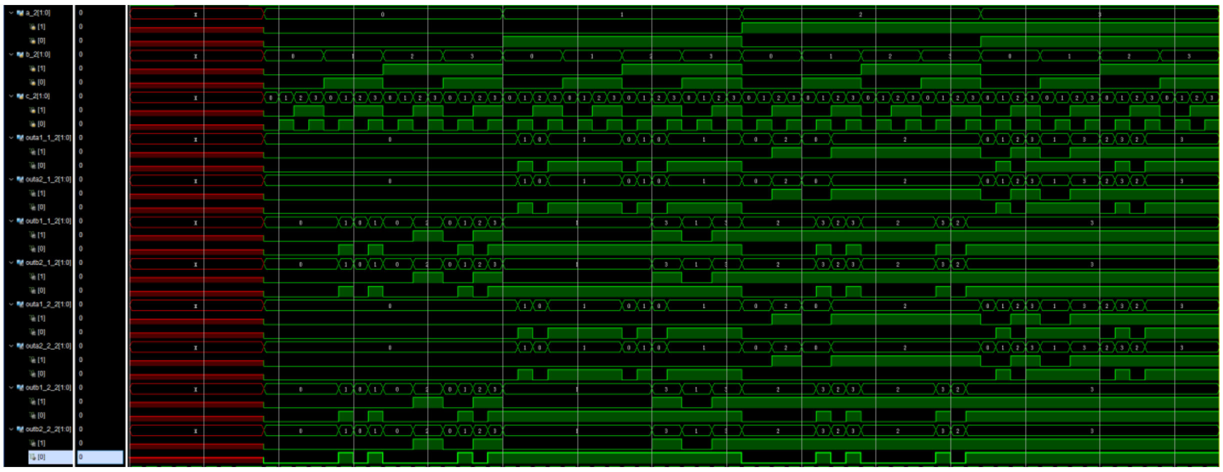
```
    //test case
        integer i;
        initial begin
            {a_1,b_1,c_1} = 1'b0;

            for (i=1; i<=7; i=i+1) begin
                #10 {a_1,b_1,c_1} = {a_1,b_1,c_1} + 1'b1;
            end
        #10
            {a_1,b_1,c_1} = 1'b0;
        #20
            {a_2,b_2,c_2} = 2'b00;
            for (i=1; i<=64; i=i+1) begin
                #10 {a_2,b_2,c_2} = {a_2,b_2,c_2} + 2'b01;
            end
        #20
        $finish();
        End

endmodule
```

- ***Wave form of simulation result (provide screen shots)***

For 1-bit. Start from begin.

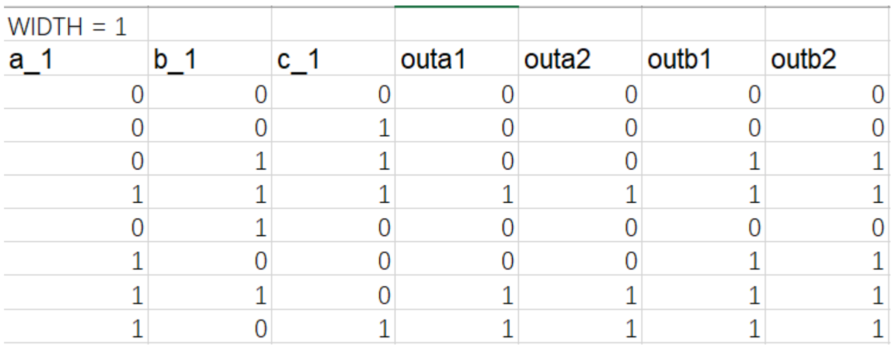


For 2-bit. Start after 1-bit, that is, after X was over.



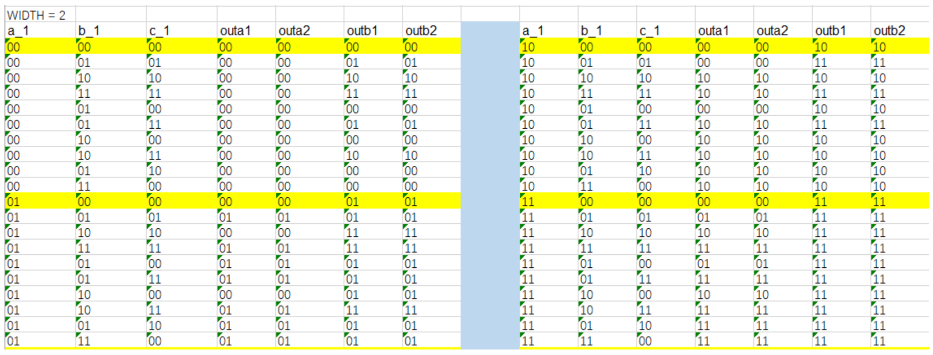
- ***The description on whether the simulation result is same as the truth-table, is the function of the design meet the expectation***

We can know from the waveform that when the width is 1, the result is in the below:



WIDTH = 1						
a_1	b_1	c_1	outa1	outa2	outb1	outb2
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	1	0	0	1	1
1	1	1	1	1	1	1
0	1	0	0	0	0	0
1	0	0	0	0	1	1
1	1	0	1	1	1	1
1	0	1	1	1	1	1

When the width is 2, the result is in the below:



WIDTH = 2														
a_1	b_1	c_1	outa1	outa2	outb1	outb2		a_1	b_1	c_1	outa1	outa2	outb1	outb2
00	00	00	00	00	00	00		10	00	00	00	00	10	10
00	01	01	00	00	01	01		10	01	01	00	00	11	11
00	10	10	00	00	10	10		10	10	10	10	10	10	10
00	11	11	00	00	11	11		10	11	11	10	10	11	11
00	01	00	00	00	00	00		10	01	00	00	00	10	10
00	01	11	00	00	01	01		10	01	11	10	10	11	11
00	10	00	00	00	00	00		10	10	00	10	10	10	10
00	10	11	00	00	10	10		10	10	11	10	10	10	10
00	01	10	00	00	00	00		10	01	10	10	10	10	10
00	11	00	00	00	00	00		10	11	00	10	10	10	10
01	00	00	00	00	01	01		11	00	00	00	00	11	11
01	01	01	01	01	01	01		11	01	01	01	01	11	11
01	10	10	00	00	11	11		11	10	10	10	10	11	11
01	11	11	01	01	11	11		11	11	11	11	11	11	11
01	01	00	01	01	01	01		11	01	00	01	01	11	11
01	01	11	01	01	01	01		11	01	11	11	11	11	11
01	10	00	00	00	01	01		11	10	00	10	10	11	11
01	10	11	01	01	11	11		11	10	11	11	11	11	11
01	01	10	01	01	01	01		11	01	10	11	11	11	11
01	11	00	01	01	01	01		11	11	00	11	11	11	11

Both two waveforms well meet the truth tables, that is, we can draw a conclusion that the two distributive laws are right and fit no matter whether the inputs are 1-bit or not.

THE DESCRIPTION OF OPERATION

Describe the problem occurred while in the lab and your solution. Any suggestions are welcomed.

- ***Problems and solutions***

(1) I think it is hard to distinguish the variables when there are plenty of variables. So, naming the variables should be taken as an important thing. Here, I used underline and number to make a difference.

(2) To ensure the simulation can cover all the cases, I used for-loop to assign the value and choose a little larger range.