DIGITAL DESIGN

ASSIGNMENT REPORT

ASSIGNMENT ID : 1

Student Name: 安钧文

Student ID: 12012109

PART 1: DIGITAL DESIGN THEORY

Provide your answers here:

Digital Design Theory

1. (a) ~~$16\times10^3 = 16000 \Rightarrow 1.6\times10^4$ bytes~~ $16\times1024 = 16384$ bytes

(b) ~~$32\times10^6 = 1.2\times10^7 \Rightarrow 3.2\times10^7$ bytes~~ $32\times(1024)^2 = 33554432$ bytes

(c) ~~$3.2\times10^9 = 3.2\times10^9 \Rightarrow 3.2\times10^9$ bytes~~ $3.2\times(1024)^3 \approx 3435973837$ bytes

2. If the number is unsigned:

largest 111111111111

decimal 4095

hexadecimal ~~##~~FFF

If signed:

largest 111111111111

decimal 2047

hexadecimal ~~##~~ 7FF

3. (a)

2	248	0
2	124	0
2	62	0
2	31	1
2	15	1
2	7	1
2	3	1
	1	

$(248)_{10} = (11111000)_2$

(b) $16\overline{)\begin{matrix}248\\15\end{matrix}}\ 8$

$(248)_{10} = (\text{F8})_{16}$ ∠ $(F_{16}=1111)_2$, $(8_{16}=1000)_2$

$(\text{F8})_{16} = (11111000)_2$

Convert to hexadecimal is faster, as it requires less operations.

4.

a. $99999999 - 25273036 = 74726963$

$\Leftarrow 74726963 + 1 = 74726964$

9's complement: 74726963

10's complement: 74726964.

b. $99999999 - 64322610 = 35677389$

9's complement: 35677389

10's complement: 35677390

5. (a) 15's complement: FFFF - C6BF = 3940

Then 16's complement: 3941.

(b) $(C)_{16} = 1100$

$(6)_{16} = 0110$

$(B)_{16} = 1011$

$(F)_{16} = 1111$

$\Rightarrow (C6BF)_{16} = (1100011010111111)_2$

(c) 1's complement: 1111111111111111 - 1100011010111111 = 0011100101000000

2's complement: 0011100101000001

(d) 0011 1001 0100 0001

3 9 4 1

$\Rightarrow (0011100101000001)_{16} = (3941)_{16}$

it equals to the answer in (a).

6. (a)

2 | 23 1

2 | 11 1

2 | 5 1

2 | 2 0

1

0.5625 x 2 = 1.125 ... 1

0.125 x 2 = 0.25 ... 0

0.25 x 2 = 0.5 ... 0

0.5 x 2 = 1 ... 1

$(23.5625)_{10} = (10111.1001)_2$

(b) $(\frac{5}{3})_{10} \approx (1.10101010)_2$

$(1.10101010)_2 = (1.6640625)_{10}$

$|1.6640625 - \frac{5}{3}| < 0.0027$

(c) $(1.10101010)_2 = (1.AA)_{16}$

$(1.AA)_{16} = (1.6640625)_{10}$

The answer is the same.

Because the conversions are all on a determined number, and conversions will not change the value of the number.

7.

(a) 975

(b) 1001 - 011 = 1110 convert excess-3 code to BCD

0111 - 011 = 0100

0101 - 011 = 0010

in excess-3 code : 642

(c) 713

(d) 754

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

8.

(a) we first calculate in binary form.

A & B = 01001010

In hexadecimal, result is 4A

(b) A | B = 11011110

Result is DE

(c) A XOR B = 10010100

Result is 94

(d) NOT A = 00100101

Result is 25

(e) NOT B = 10110001

Result is B1

(f) from (a)

NAND = 10110101

Result is B5

(g) from (b)

NOR = 00100001

Result is 21

PART 2: DIGITAL DESIGN LAB (TASK1)

DESIGN

- Verilog design

```
module sum #(parameter width=1)(
add1, add2, sum1
    );
    input[width-1:0] add1, add2;
    output [width:0] sum1;
    assign sum1=add1+add2;

endmodule
```

- Truth-table

  1bit Addition:

Addend	Augend	Result
1	1	2((10$_2$)
1	0	1
0	1	1
0	0	0

2bit Addition:

Addend	Augend	Result
00	00	0
00	10	$2((10)_2)$
00	01	1
00	11	$3((11)_2)$
01	00	1
01	01	$2((10)_2)$
01	10	$3((11)_2)$
01	11	$4((100)_2)$
11	00	$3((11)_2)$
11	01	$4((100)_2)$
11	10	$5((101)_2)$
11	11	$6((110)_2)$
10	00	$2((10)_2)$
10	01	$3((11)_2)$
10	10	$4((100)_2)$
10	11	$5((101)_2)$

SIMULATION

- Using Verilog

  1bit Addition:

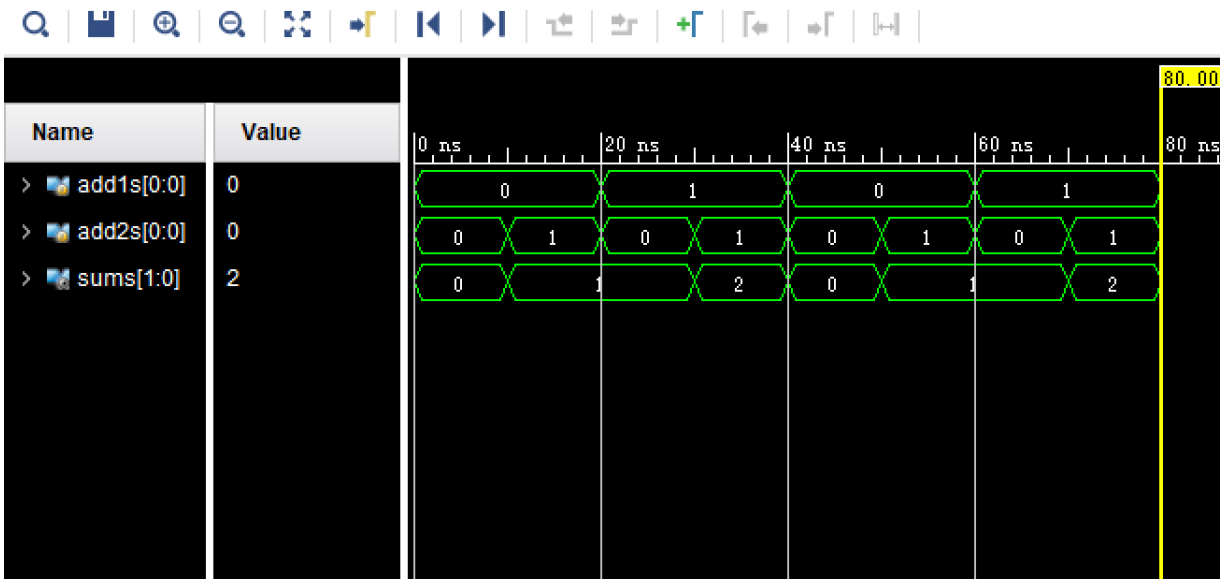
```
module sum_sim(

    );
    reg[0:0] add1s,add2s;
    wire [1:0]sums;

    sum #(1)u(.add1(add1s),.add2(add2s),.sum1(sums));
    initial
    begin
    {add1s,add2s}=2'b00;
    #10 {add1s,add2s}={add1s,add2s}+1;
    #10 {add1s,add2s}={add1s,add2s}+1;
    #10 {add1s,add2s}={add1s,add2s}+1;
    #10 {add1s,add2s}={add1s,add2s}+1;
    #10 {add1s,add2s}={add1s,add2s}+1;
    #10 {add1s,add2s}={add1s,add2s}+1;
    #10 {add1s,add2s}={add1s,add2s}+1;
    #10 {add1s,add2s}={add1s,add2s}+1;
//        #10 {add1s,add2s}={add1s,add2s}+1;
//    #10 {add1s,add2s}={add1s,add2s}+1;
//    #10 {add1s,add2s}={add1s,add2s}+1;
//    #10 {add1s,add2s}={add1s,add2s}+1;
//    #10 {add1s,add2s}={add1s,add2s}+1;
//    #10 {add1s,add2s}={add1s,add2s}+1;
//    #10 {add1s,add2s}={add1s,add2s}+1;

    $finish;
    end
```

2bit Addition:

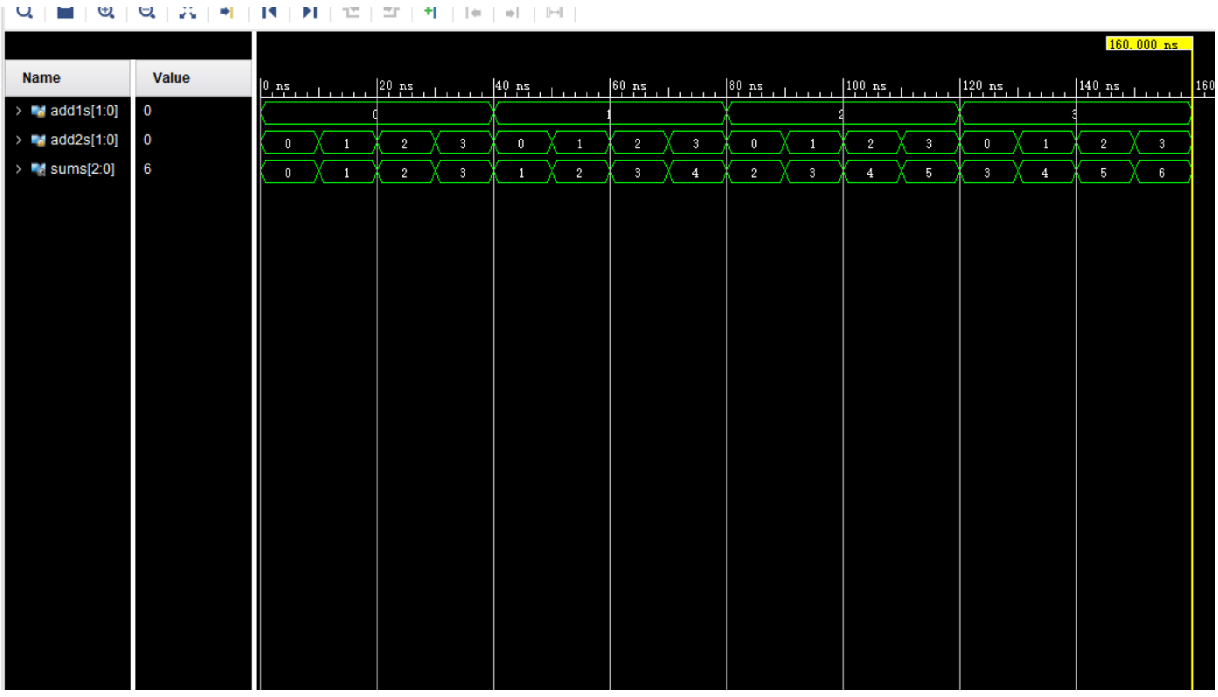
```
module sum_sim(

    );
    reg[1:0] add1s,add2s;
    wire [2:0]sums;

    sum #(2)u(.add1(add1s),.add2(add2s),.sum1(sums));
    initial
    begin
    {add1s,add2s}=4'b0000;
    #10 {add1s,add2s}={add1s,add2s}+1;
    #10 {add1s,add2s}={add1s,add2s}+1;
    #10 {add1s,add2s}={add1s,add2s}+1;
    #10 {add1s,add2s}={add1s,add2s}+1;
    #10 {add1s,add2s}={add1s,add2s}+1;
    #10 {add1s,add2s}={add1s,add2s}+1;
    #10 {add1s,add2s}={add1s,add2s}+1;
    #10 {add1s,add2s}={add1s,add2s}+1;
        #10 {add1s,add2s}={add1s,add2s}+1;
    #10 {add1s,add2s}={add1s,add2s}+1;
    #10 {add1s,add2s}={add1s,add2s}+1;
    #10 {add1s,add2s}={add1s,add2s}+1;
    #10 {add1s,add2s}={add1s,add2s}+1;
    #10 {add1s,add2s}={add1s,add2s}+1;
    #10 {add1s,add2s}={add1s,add2s}+1;
    #10 {add1s,add2s}={add1s,add2s}+1;

    $finish;
    end
endmodule
```

- Wave form of simulation result

  1bit:



  2bit:



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

- The simulation covered all the test cases while result is the same as the truth-table, and that the function of the design meets the expectation. To be noted that the simulation is presented in decimal form for convenience purposes.

  For example, in 2bit addition, 11+11=110, which in decimal form is 3+3=6, it matches in simulation and truth table. In 1bit addition, 1+1=10, which in decimal form is 1+1=2, it matches in simulation and truth table.

THE DESCRIPTION OF OPERATION

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

- Problem: Vivado is not as intelligent as JetBrains IDEs, as it doesn't have auto-filling features, auto error correcting, and it's difficult to spot syntax errors in vivado.
  Solution: First think the design through before implementing into vivado, and be careful of spelling mistakes, semi-colons, (and) matching. Spend more effort and eventually I will get used to it.

- Problem: Removing redundant simulation time.
  Solution: Simulation can be ended with $finish().

- Problem: Value of sum in simulation is "Z" while simulating 2bit addition.
  Solution: Instantiate parameter width in simulation file to #(2).

PART 2: DIGITAL DESIGN LAB (TASK2)

DESIGN

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

- *Verilog design while using data flow*

  //1bit

```
module distributive1bit_df(
a,b,c,left1,right1,left2,right2
    );
    input[0:0] a,b,c;
    output[0:0] left1,right1,left2,right2;

    assign left1=a & (b|c);
    assign right1=(a&b)|(a&c);
    assign left2=a|(b&c);
    assign right2=(a|b)&(a|c);

endmodule
```

  //2bit

```
module distributive2bit_df(
a,b,c,left1,right1,left2,right2
    );
    input[1:0] a,b,c;
    output[1:0] left1,right1,left2,right2;

    assign left1=a & (b|c);
    assign right1=(a&b)|(a&c);
    assign left2=a|(b&c);
    assign right2=(a|b)&(a|c);
endmodule
```

- *Verilog design while using structured design*

//1bit

```
module distributive1bit_sd(
a,b,c,left1,right1,left2,right2
);
input[0:0]a,b,c;
output[0:0]left1,right1,left2,right2;

wire borc,aandb,aandc,bandc,aorb,aorc;

or or1(borc,b,c);
and and1(left1,borc,a);

and and2(aandb,a,b);
and and3(aandc,a,c);
or or2(right1,aandb,aandc);

and and4(bandc,b,c);
or or3(left2,bandc,a);

or or4(aorb,a,b);
or or5(aorc,a,c);
and and5(right2,aorb,aorc);



endmodule
```

//2bit

```
module distributive2bit_sd(
a,b,c,left1,right1,left2,right2
    );
    input[1:0]a,b,c;
    output[1:0]left1,right1,left2,right2;
    wire[1:0] borc,aandb,aandc,bandc,aorb,aorc;

    or or1_1(borc[0],b[0],c[0]);
    or or1_2(borc[1],b[1],c[1]);
    and and1_1(left1[0],borc[0],a[0]);
    and and1_2(left1[1],borc[1],a[1]);
    and and2_1(aandb[0],a[0],b[0]);
    and and2_2(aandb[1],a[1],b[1]);
    and and3_1(aandc[0],a[0],c[0]);
    and and3_2(aandc[1],a[1],c[1]);
    or or2_1(right1[0],aandb[0],aandc[0]);
    or or2_2(right1[1],aandb[1],aandc[1]);

    and and4_1(bandc[0],b[0],c[0]);
    and and4_2(bandc[1],b[1],c[1]);
    or or3_1(left2[0],bandc[0],a[0]);
    or or3_2(left2[1],bandc[1],a[1]);
    or or4_1(aorb[0],a[0],b[0]);
    or or4_2(aorb[1],a[1],b[1]);
    or or5_1(aorc[0],a[0],c[0]);
    or or5_2(aorc[1],a[1],c[1]);
    and and5_1(right2[0],aorb[0],aorc[0]);
    and and5_2(right2[1],aorb[1],aorc[1]);
endmodule
```

- *Truth-table*

//1bit

A	B	C	left1	right1	left2	right2
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	0	1	1
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1

//2bit

A	B	C	left1	right1	left2	right2	A	B	C	left1	right1	left2	right2
00	00	00	00	00	00	00	10	00	00	00	00	10	10
00	00	01	00	00	00	00	10	00	01	00	00	10	10
00	00	10	00	00	00	00	10	00	10	10	10	10	10
00	00	11	00	00	00	00	10	00	11	10	10	10	10
00	01	00	00	00	00	00	10	01	00	00	00	10	10
00	01	01	00	00	01	01	10	01	01	00	00	11	11
00	01	10	00	00	00	00	10	01	10	10	10	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	01	00	00	00	00	10	10	01	10	10	10	10
00	10	10	00	00	10	10	10	10	10	10	10	10	10
00	10	11	00	00	10	10	10	10	11	10	10	10	10
00	11	00	00	00	00	00	10	11	00	10	10	10	10
00	11	01	00	00	01	01	10	11	01	10	10	11	11
00	11	10	00	00	10	10	10	11	10	10	10	10	10
00	11	11	00	00	11	11	10	11	11	10	10	11	11
01	00	00	00	00	00	00	11	00	00	00	00	11	11
01	00	01	01	01	01	01	11	00	01	01	01	11	11
01	00	10	00	00	01	01	11	00	10	10	10	11	11
01	00	11	01	01	01	01	11	00	11	11	11	11	11
01	01	00	01	01	01	01	11	01	00	01	01	11	11
01	01	01	01	01	01	01	11	01	01	01	01	11	11
01	01	10	01	01	01	01	11	01	10	11	11	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	01	01	01	01	01	11	10	01	11	11	11	11
01	10	10	00	00	11	11	11	10	10	10	10	11	11
01	10	11	01	01	11	11	11	10	11	11	11	11	11
01	11	00	01	01	01	01	11	11	00	11	11	11	11
01	11	01	01	01	01	01	11	11	01	11	11	11	11
01	11	10	01	01	11	11	11	11	10	11	11	11	11
01	11	11	01	01	11	11	11	11	11	11	11	11	11

SIMULATION

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

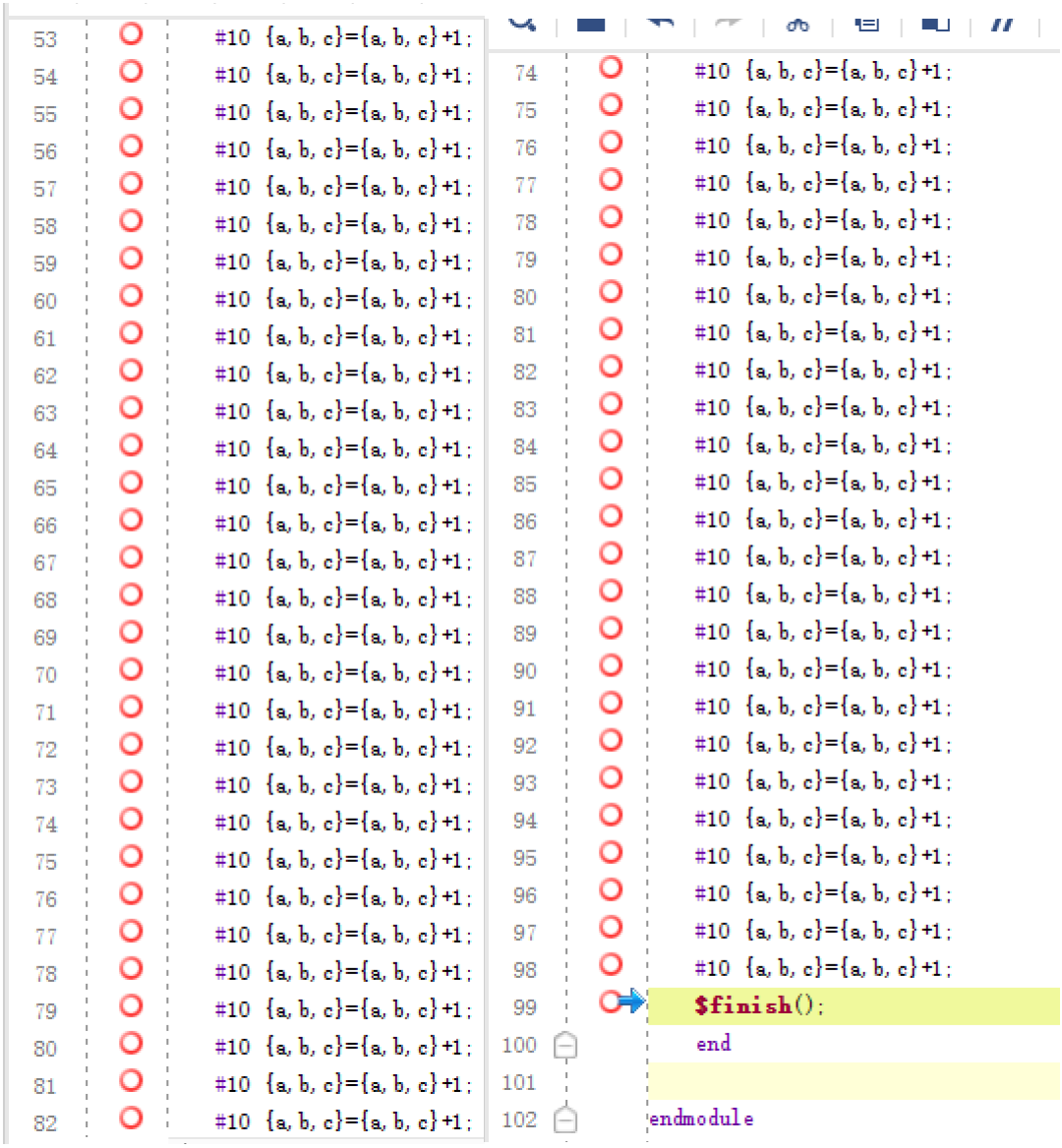
- *Using Verilog*

//1bit

```
module distrubitive_sim(

    );
    reg[0:0] a,b,c;
    wire[0:0] left1,right1,left2,right2;

    distributive1bit_df u1_df(.a(a),.b(b),.c(c),.left1(left1),.left2(left2),.right1(right1),.right2(right2));
    distributive1bit_sd u1_sd(.a(a),.b(b),.c(c),.left1(left1),.left2(left2),.right1(right1),.right2(right2));

    initial
    begin
    {a,b,c}=3'b000;
    #10 {a,b,c}={a,b,c}+1;
    #10 {a,b,c}={a,b,c}+1;
    #10 {a,b,c}={a,b,c}+1;
    #10 {a,b,c}={a,b,c}+1;
    #10 {a,b,c}={a,b,c}+1;
    #10 {a,b,c}={a,b,c}+1;
    #10 {a,b,c}={a,b,c}+1;
    #10 $finish();
    end

endmodule
```

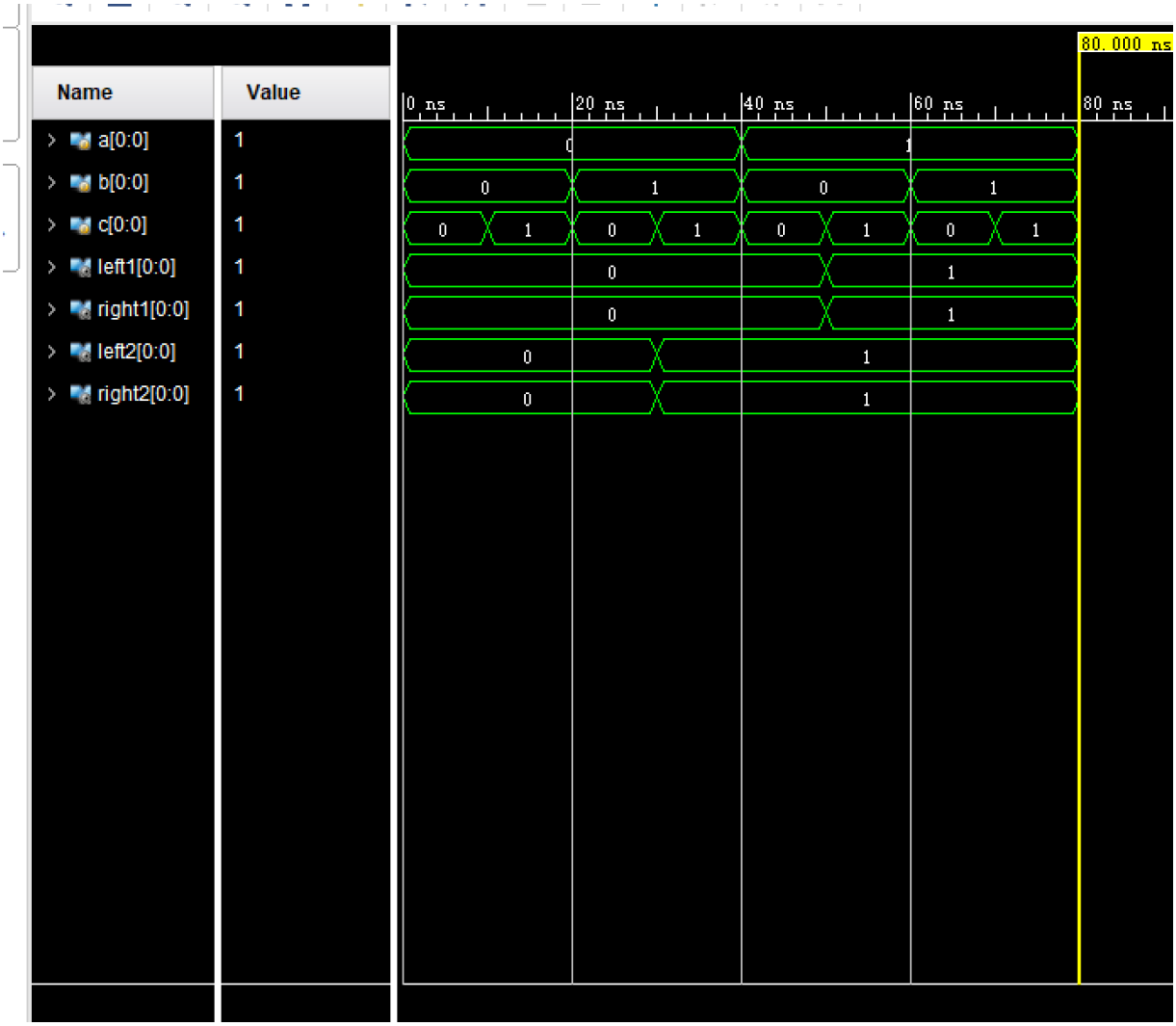
//2bit

```
module distributive2bit_sim(

    );
reg[1:0] a,b,c;
    wire[1:0] left1,right1,left2,right2;

    distributive2bit_df u2_df(.a(a),.b(b),.c(c),.left1(left1),.left2(left2),.right1(right1),.right2(right2));
    distributive2bit_sd u2_sd(.a(a),.b(b),.c(c),.left1(left1),.left2(left2),.right1(right1),.right2(right2));

    initial
    begin
    {a,b,c}=6'b000000;
    #10 {a,b,c}={a,b,c}+1;
    #10 {a,b,c}={a,b,c}+1;
    #10 {a,b,c}={a,b,c}+1;
    #10 {a,b,c}={a,b,c}+1;
    #10 {a,b,c}={a,b,c}+1;
    #10 {a,b,c}={a,b,c}+1;
    #10 {a,b,c}={a,b,c}+1;
        #10 {a,b,c}={a,b,c}+1;
    #10 {a,b,c}={a,b,c}+1;
    #10 {a,b,c}={a,b,c}+1;
    #10 {a,b,c}={a,b,c}+1;
    #10 {a,b,c}={a,b,c}+1;
    #10 {a,b,c}={a,b,c}+1;
    #10 {a,b,c}={a,b,c}+1;
    #10 {a,b,c}={a,b,c}+1;
    #10 {a,b,c}={a,b,c}+1;
    #10 {a,b,c}={a,b,c}+1;
    #10 {a,b,c}={a,b,c}+1;
    #10 {a,b,c}={a,b,c}+1;
```

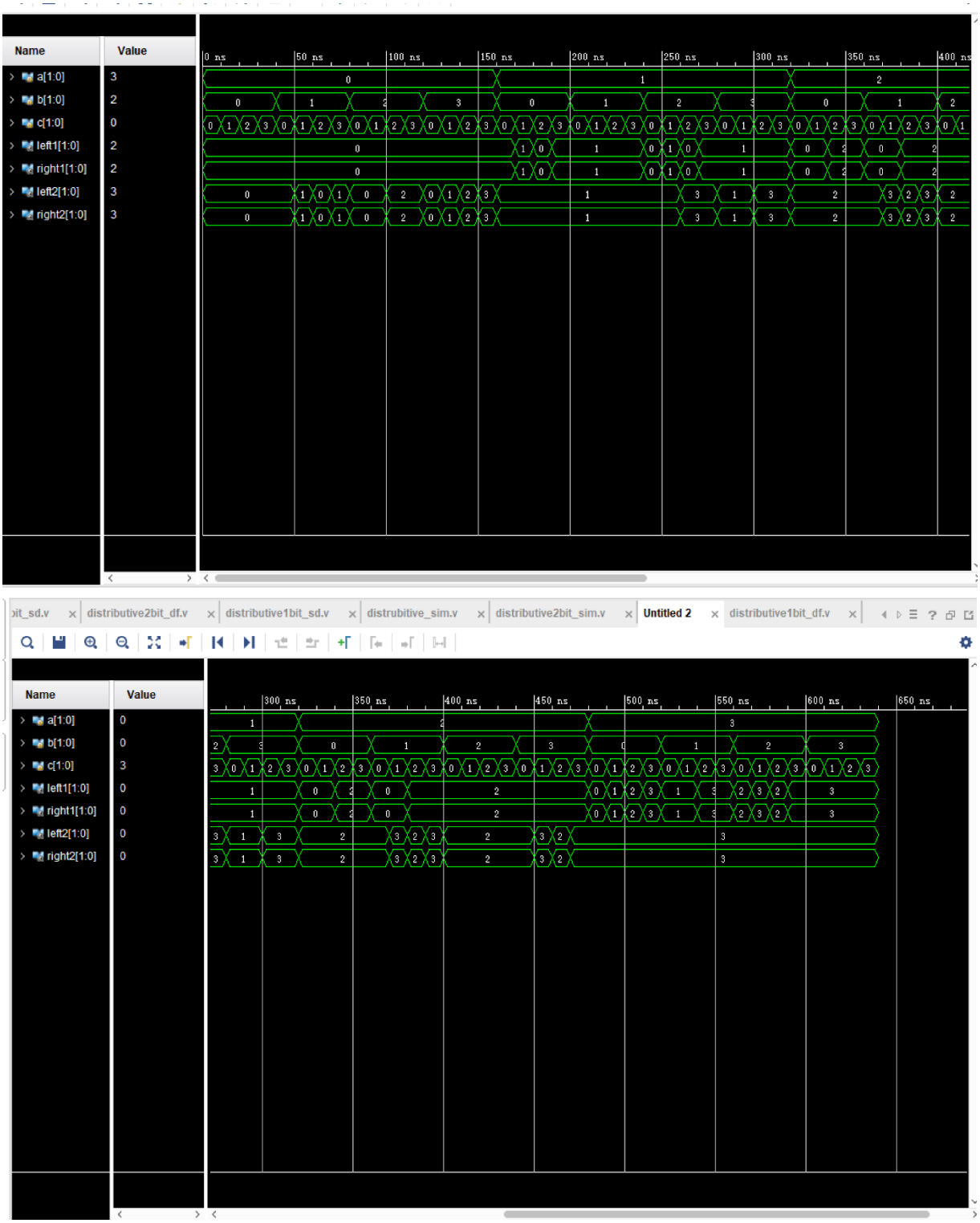

#10 {a, b, c}={a, b, c}+1;
$finish();
end
endmodule

- *Wave form of simulation result*

//1bit



//2bit


Name
Value
a[1:0]
b[1:0]
c[1:0]
left1[1:0]
right1[1:0]
left2[1:0]
right2[1:0]
distributive2bit_df.v
distributive1bit_sd.v
distrubitive_sim.v
distributive2bit_sim.v
Untitled 2
distributive1bit_df.v

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

  In the design, I named the result of left and right side of equation(a) to left1 and right1, equation(b) to left2 and right2. I used both data flow and structured design to calculate left1, right1, left2 and right2 in 1bit and 2bit.

  As we can see from the truth table, left1 and right1, left2 and right2 share the same truth value, meaning that distribution law is verified in both 1bit and 2bit calculations. To be noted that the truth table is presented in binary form, while simulation results are in decimal form for convenient purposes as it's easier for eyes to compare the results.

  In the simulation, I iterated through every situation in both 1bit (8 situations) and 2bit (64 situations), in both data flow and structured design, and the value of left1 and right1, left2 and right2 matches the truth table, again verifying distribution law and the correctness of the design.

THE DESCRIPTION OF OPERATION

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

- Problem: In simulation, right2 is X.

  Solution: Since I simulated both structured design and data flow, right2 should be the same in both design. I found bug in df where I mistakenly wrote a&b | b&c (b&c should be a&c). After correcting, right2 has the correct value.

- Problem: Primitive gates and only used in 1bit calculations, but the inputs are 2bit.

  Solution: I used a[0] to represent the first bit of a (a 2bit input), and a[1] to represent the second bit of a, by dividing 2bit values into 1bit, I am able to use primitive gates to calculate.

- Problem: I created two simulation files in the same project, but vivado only simulated one of them.

  Solution: Vivado only simulates the “top” sim file, by right clicking on a sim file and selecting “set as top”, I can simulate the other sim file.