DIGITAL DESIGN

LAB3 BITWISE OPERATION IN VERILOG, GATES IN RTL VS LUT IN FPGA

WANGW6@SUSTECH.EDU.CN

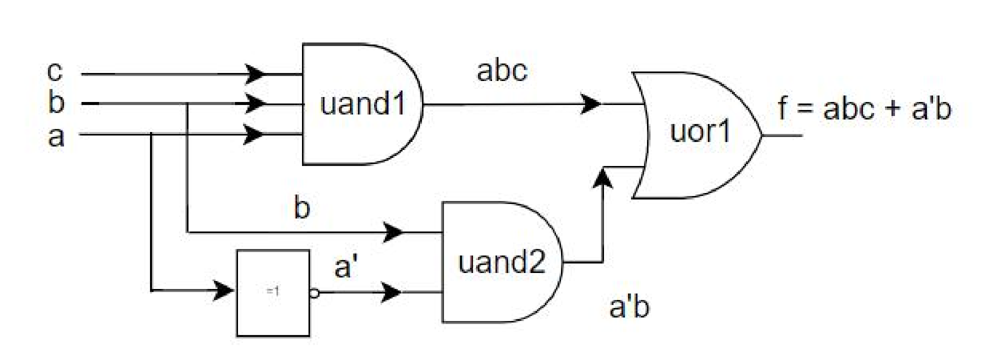
LAB3

- Verilog
  - Bitwise and logic operations in Verilog
- Design mode in Verilog
  - 1. Data Flow
  - 2. Data Flow vs Structrue Design(review)
- Vivado
  - Schematic of “TRL analysis” (Gates)
  - Schematic of “Synthesis” (LUT of FPGA chip)
  - active module (set as top)

DO THE DESIGN BY USING THE PRIMITIVE GATES(1)

Do the design to impliment the following circuit:

f(a,b,c) = abc + a'b




```
module lab3_1_gates(
   input a,
   input b,
   input c,
   output f
   );
   //f(a,b,c) = abc + a'b
   wire  not_a, and1_or1, and2_or1;

   and   uand1(and1_or1,   a, b, c);        //and1_or1 = abc
   not   unot1(not_a,  a);                  // nota = a'
   and   uand2(and2_or1,   not_a, b);   //and2_or2 = a'b
   or    uor1   ( f,    and1_or1, and2_or1);  //f = abc + a'b

endmodule
```

DO THE DESIGN BY USING THE PRIMITIVE GATES(2)

Do the design to impliment the following circuit:

f(a,b,c)

= ∑(2,4,5,6)

= a'bc'+ab'c'+ab'c+abc'

```
module lab3_2_gates(           //piece 1 /2 in Verilog
   input a,
   input b,
   input c,
   output f
   );
   //f(a,b,c) = a'bc' + ab'c' + ab'c + abc'
   wire  not_a, not_b,not_c;
   wire  and1_or1, and2_or1, and3_or1, and4_or1;
```

```
//piece 2 /2  in Verilog
   not   unot1(not_a,  a);                      // not_a = a'
   not   unot2(not_b,  b);                      // not_b = b'
   not   unot3(not_c,  c);                      // not_c = c'

   and   uand1(and1_or1, not_a, b, not_c);     //and1_or1 = a'bc'
   and   uand2(and2_or1,   a, not_b, not_c);   //and2_or1 = ab'c'
   and   uand3(and3_or1,   a, not_b, c);       //and3_or1 = ab'c
   and   uand4(and4_or1,   a, b, not_c);       //and4_or1 = abc'

   //f(a,b,c) = a'bc' + ab'c' + ab'c + abc'
   or     uor1   ( f, and1_or1, and2_or1, and3_or1, and4_or1);

endmodule
```

BITWISE AND LOGICAL OPERATIONS IN VERILOG(1)

Four-valued logic (The IEEE 1364 standard): 0, 1, Z (high impedance), and X (unknown logic value).

Operator :

~ & ^ ~^ ^~ | ! && ||

Priority:

~ ! >

& >

^ ~^ ^~ >

| >

&& >

||

Operator type	Operator symbols	Operation performed
Bitwise	~	Bitwise NOT (1's complement)
	&	Bitwise AND
	|	Bitwise OR
	^	Bitwise XOR
	~^ or ^~	Bitwise XNOR
Logical	!	NOT
	&&	AND
	||	OR

BITWISE AND LOGICAL OPERATIONS IN VERILOG(2)

Tips:

While the **bit-width of the operand is 1**, the bitwise operation is **same** as the corresponding logical operation.

While the **bit-width of the operand is more than 1**, the bitwise operation **is NOT alway same** as the corresponding logical operation.

a	b	a | b	a || b	a & b	a && b	~a	!a
2'b01	2'b10	2'b11	2'b01	2'b00	2'b01	2'b10	2'b00
2'b11	2'b11	2'b11	2'b01	2'b11	2'b01	2'b00	2'b00
2'b00	2'b10	2'b10	2'b01	2'b00	2'b00	2'b11	2'b01
…	…	…	…	…	…	…	…

The relationship between boolean and number in Verilog:

1) Zero is taken as **False**, None Zero is taken as **True**
2) **False** is represented by zero, **True** is represented by one.

DESIGN MODE IN VERILOG - DATA FLOW

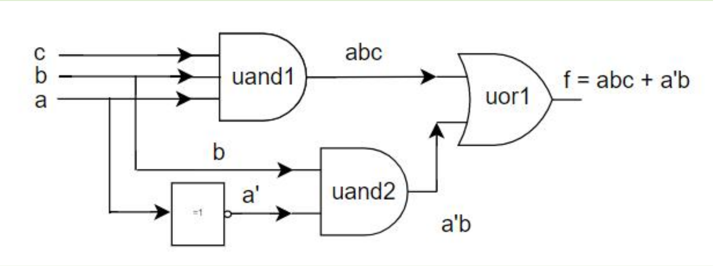
- Data flow design： using “assign “ as *continuous assignment* , to transfer the data from input ports through variables to the output ports .

logical expression	data flow in Verilog
f(a,b,c) = abc + a’b	assign f = a & b &c | ~a &b;
f(a,b,c) = ∑(2,4,5,6) = a’bc’+ab’c’+ab’c+abc’	assign f = ~a&b&~c | a&~b&~c | a& ~b&c | a&b&~c;

TIPS:
The priority of operator “ & ” is higher than the operator “ | ”

DATA FLOW VS STRUCTURE DESIGN(BASED ON THE PRIMITIVE GATES)

Data Flow in Verilog **logical expression: f(a,b,c)= abc + a’b**	**Structure Design in Verilog (Based on the primitive Gates)** **logical expression: f(a,b,c)= abc + a’b**
assign *f = a & b &c | ~a &b;*	wire *not_a, and1_or1, and2_or1;* and *uand1(and1_or1, a, b, c);* not *unot1(not_a, a);* and *uand2(and2_or1, not_a, b);* or *uor1(f, and1_or1, and2_or2);*
wire *and_abc, and_na_b;* assign *and_abc = a & b &c;* assign *and_na_b = ~a &b;* assign *f = and_abc | and_na_b;* TIPS: Both 1 *continuous* assignment statement or several *continuous* assignment statements are ok.	wire *not_a, and1_or1, and2_or1;* or *uor1(f, and1_or1, and2_or2);* not *unot1(not_a, a);* and *uand2(and2_or1, not_a, b);* and *uand1(and1_or1, a, b, c);* TIPS: The **order** in which statements **not in the range of 'beigin' and 'end'** are written does not affect the description of the circuit, as **Verilog** has the **parallelism** characteristic.



WANGW6@SUSTECH.EDU.CN

DATA FLOW DESIGN

Demo: a) q1 = x b) q2=x + xy c) q3=x(x + y)

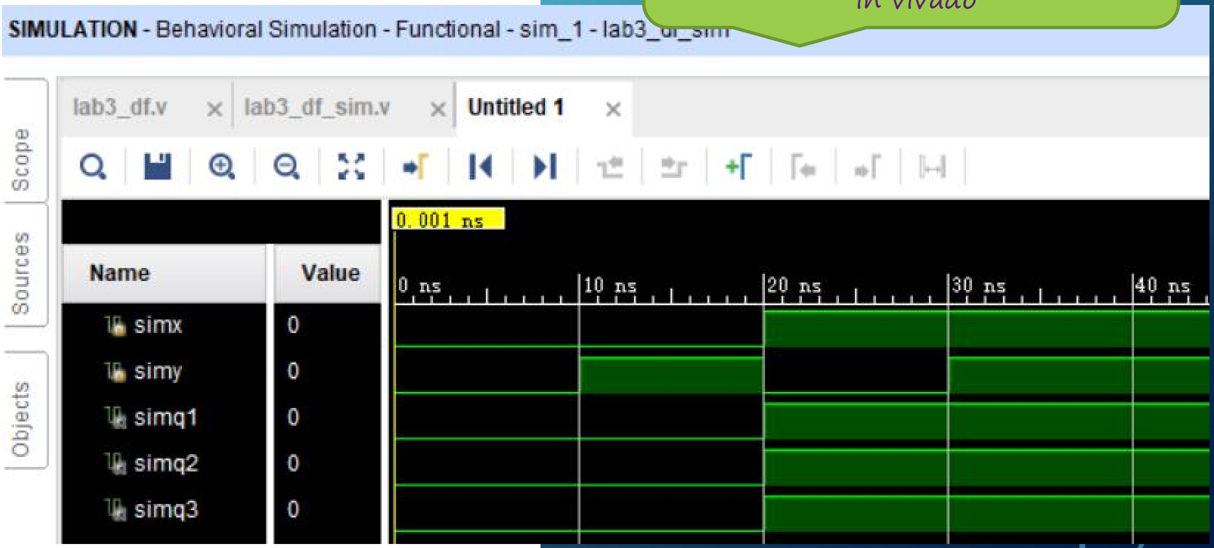
circuit design, Design source file in vivado

```
module lab3_df(
    input x,
    input y,
    output q1,
    output q2,
    output q3
    );
    assign q1 = x;
    assign q2 = x | (x & y);
    assign q3 = x & (x | y);

endmodule
```

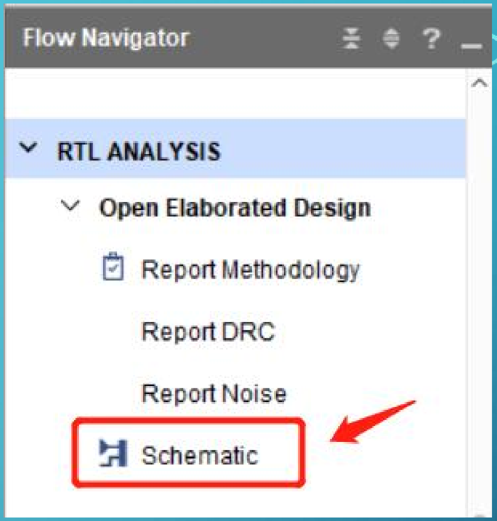
Testbench, Simulation source file in vivado

```
module lab3_df_sim( );
    reg simx, simy;
    wire simq1, simq2, simq3;
    lab3_df u_df(
    .x(simx), .y(simy), .q1(simq1), .q2(simq2), .q3(simq3) );

    initial
    begin
        simx=0;
        simy=0;
     #10
        simx=0;
        simy=1;
     #10
        simx=1;
        simy=0;
     #10
        simx=1;
        simy=1;
    end
endmodule
```

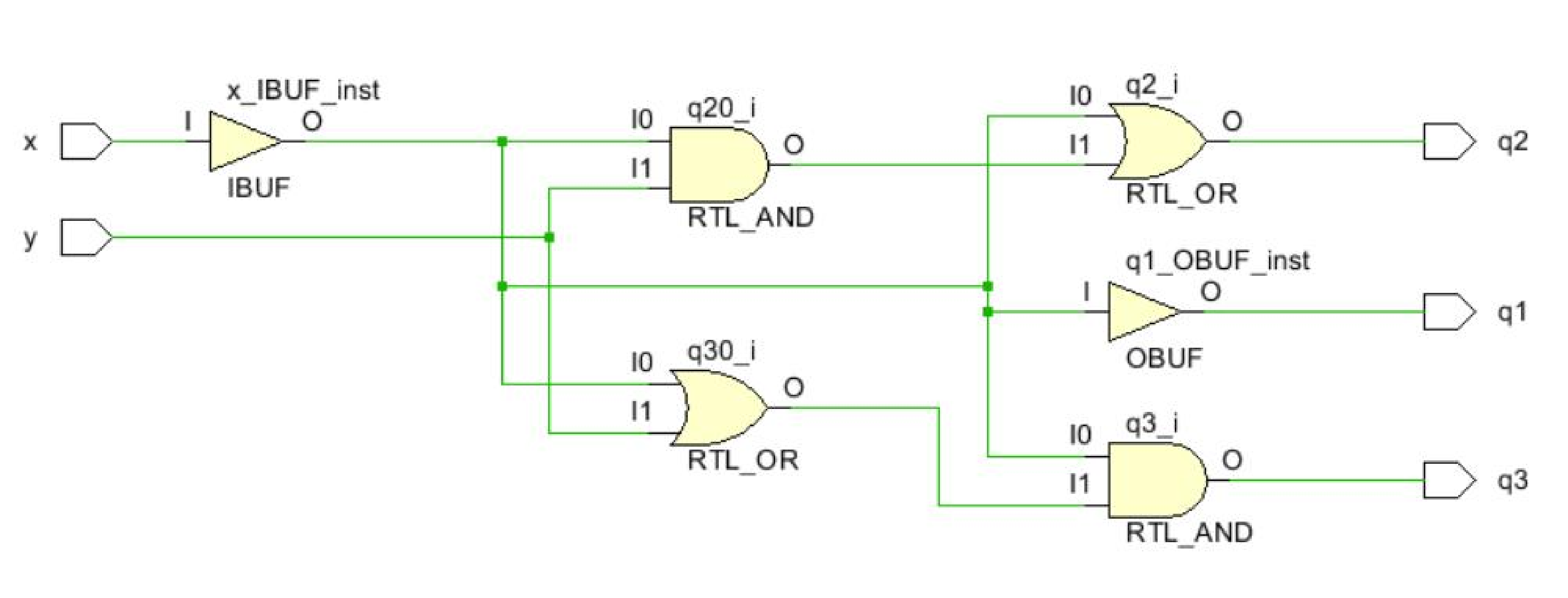
waveform generated by the simulator based on the testbecn and design in vivado



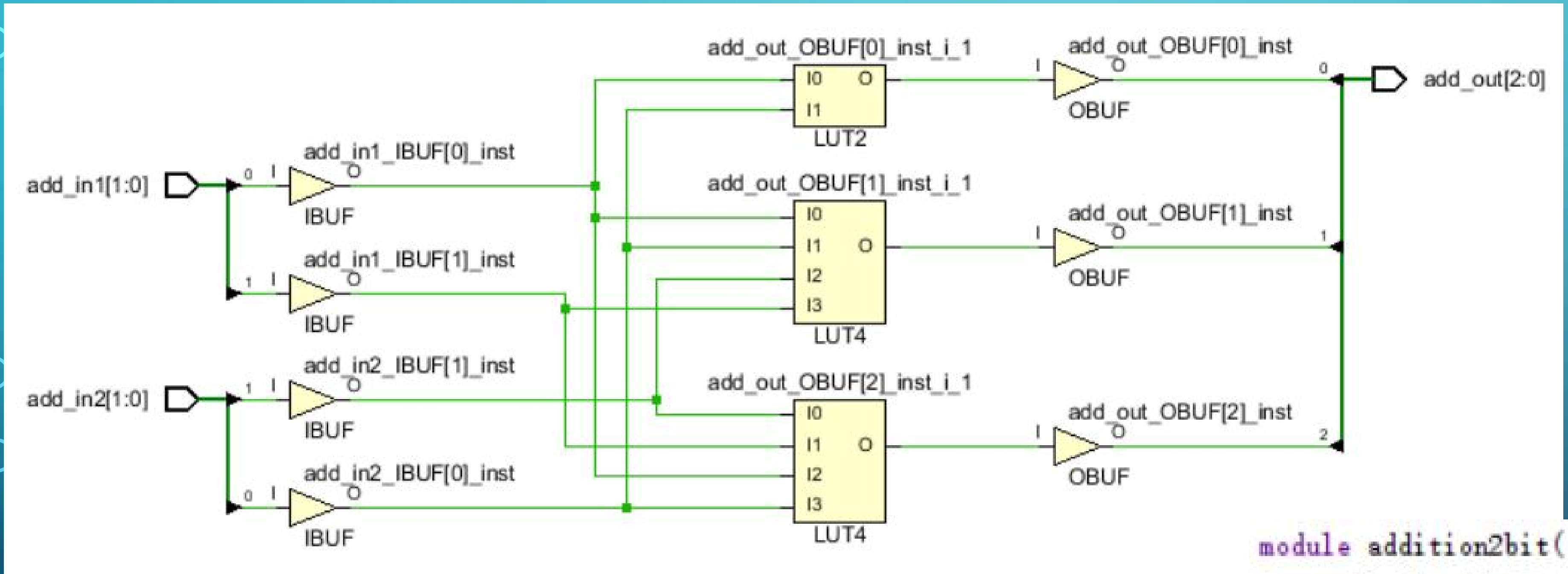
SCHEMATIC IN ‘RTL ANALYSIS’

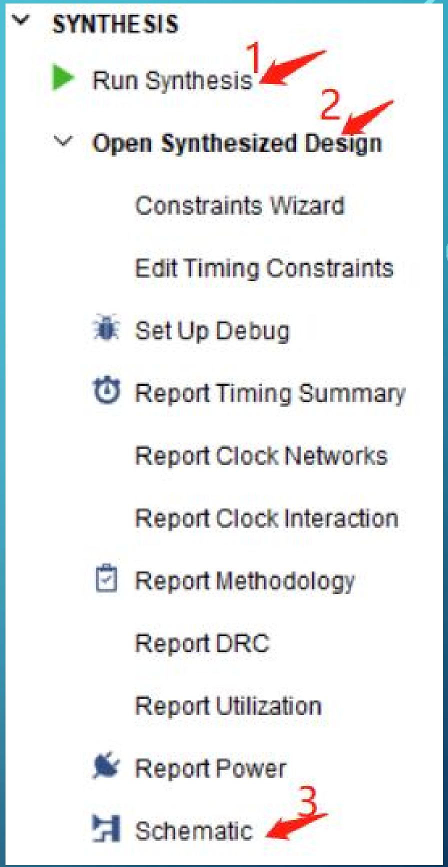



```
module lab3_df(
    input x,
    input y,
    output q1,
    output q2,
    output q3
    );
    assign q1 = x;
    assign q2 = x | (x & y);
    assign q3 = x & (x | y);

endmodule
```



SCHEMATIC IN 'SYNTHESIS'(1)

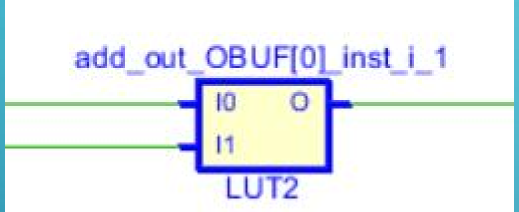




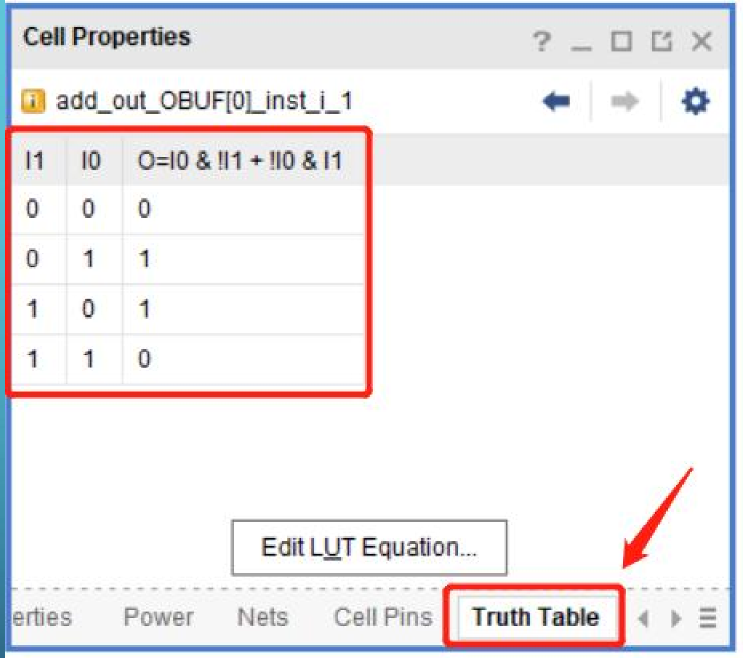

```
module addition2bit(
input [1:0] add_in1,
input [1:0] add_in2,
output [2:0] add_out
    );
    assign add_out = add_in1+add_in2;
endmodule
```

SCHEMATIC IN 'SYNTHESIS'(2)

- **Double click** the LUT in schematic window



- In the '**Cell Properties**' window , choose '**Truth Table**' , the truth table of the cell is shown



PRACTICE1

- 1. Do the circuit design:
  - There are 3 inputs: x, y and z, 3 output: o1,o2 and o3(all of them are 1 bit width)
  - The logical expression between inputs and outputs are:

  o1= xyz+xyz’ , o2=xy(z+z’) , o3=xy

  Implement the circuit by using data flow

- 2. Get the schematic of the circuit in “RTL analysis” and “Synthesis” respectively, describe the differences between them.

PRACTICE1

- 3. create testbench, do simulation to verify function of the design.

   make your conclusion about the following the by using the waveform of simluation.

   xyz+xyz' = xy(z+z') = xy

- 4. generate bitstream file, test the circuit on the board
- 5. For 3 circuit : o1= xyz+xyz', o2= xy(z+z'), o3= xy, Which circit is better,why ?

PRACTICE2(OPTIONAL)

- Design a circuit to get the addition of two two-bit unsigned numbers:
  - **In the design, the operator "+" in verilog in not allowed here.**
  - Build a test bench to verify the function of your design.
  - Programe the the FPGA chip with the bitstream file, then test the design.

a[1]	a[0]	b[1]	b[0]	sum[2]	sum[1]	sum[0]
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

TIPS1

- List the Truth-table of the circuit.
- Recode it's logical expression about every bit of output and the inputs.

  sum[0] = ...; sum[1]=....; sum[2]=....;

  sum[2] = a[1]' a[0] b[1] b[0] + a[1] a[0]' b[1] b[0]' + ...

- Using bitwise operator "&" ,"|" and "~" to express the logical expression in verilog(Don't forget the keyword "assign" in verilog).

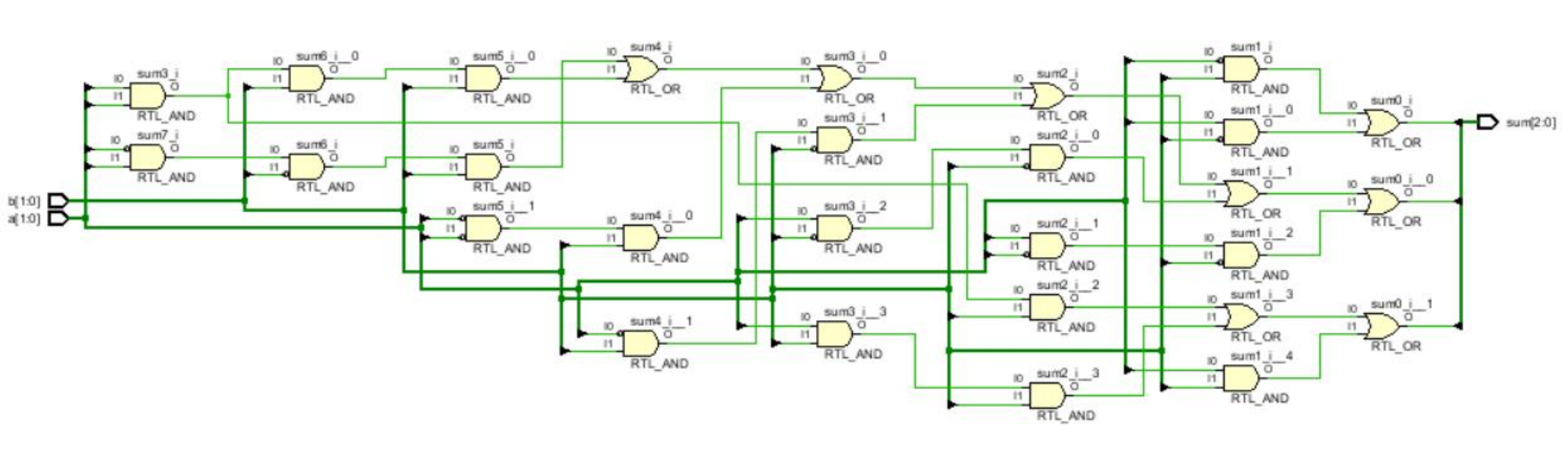
  assign sum[2] = ~a[1] & a[0] & b[1] & b[0] | a[1] & ~a[0] & b[1] & ~b[0] | ...

a[1]	a[0]	b[1]	b[0]	sum[2]	sum[1]	sum[0]
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

Q: How many gates needed in this circuit ? is it too much ?

PRACTICE3(OPTIONAL)

- Design a circuit to get the addition of two two-bit unsigned numbers:
  - **In the design:**
    - **the operator “+” in verilog in NOT allowed here.**
    - **using gates as less as possible.**
  - Build a test bench to verify the function of your design.



TIP2

- Simplify the circuit by using karnaugh map.

a[1]	a[0]	b[1]	b[0]	sum[0]
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

Before the simplification, there are only ? not gate(s), ? and gate(s) and ? or gate(s) in the circuit.

sum[0]		b[1]b[0]			
		00	01	11	10
a[1]a[0]	00		1	1	
	01	1			1
	11	1			1
	10		1	1	

sum[0]		b[1]b[0]			
		00	01	11	10
a[1]a[0]	00		1	1	
	01	1			1
	11	1			1
	10		1	1	

After simplified by using karnaugh map, the circuit about sum[0] and a,b in Verilog is:

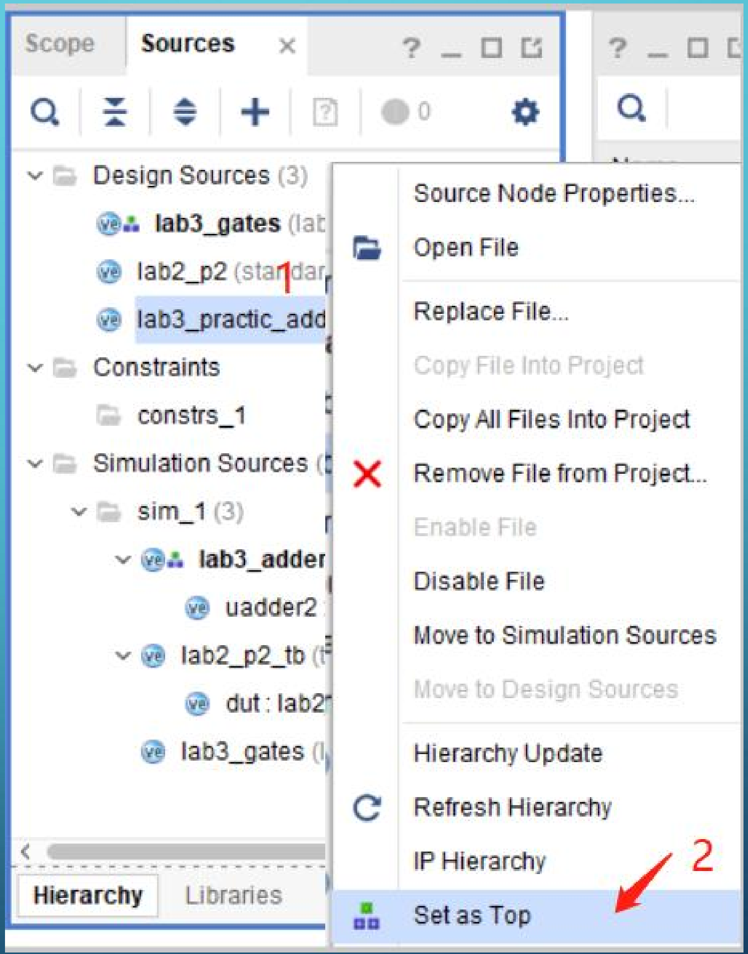
```
assign sum[0]= ~a[0]&b[0] + ~b[0]&a[0];
```

There are only ? not gate(s), ? and gate(s) and ? or gate(s).

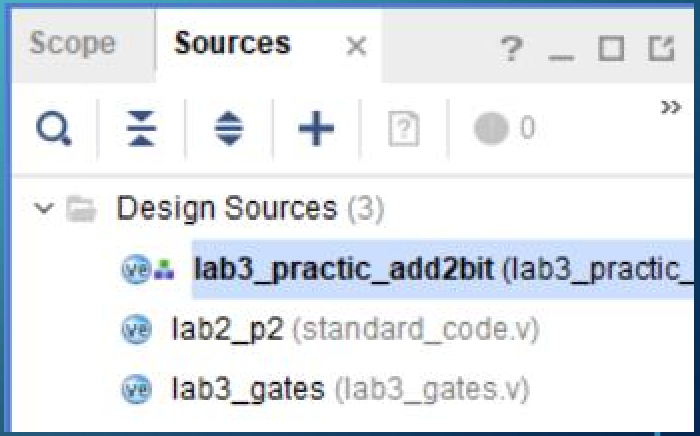
VIVADO OPERATION TIPS

- In Vivado project, top module is treated as active.
  - in Design source, active module work with active constraints file to generate the bitstream file.
  - in Simulation source, active module is runned by the simulator to generate the waveform.
- There is only 1 top module in Design Sources and 1 top module in Simultion Sources.
- The top module could be set by manual.
  - 1. left click on the file to choose the one*(in the demo on the right picture it is lab3_practice_add2bit)*
  - 2. right click on it to invoke the pop-up window, click "Set as Top" in it.

Before setting, "**lab3_gates**" is top module in Design sources



After setting, the top module in Design sources has changed tobe **lab3_practic_add2bit**.



WANGW6@SUSTECH.EDU.CN