

A decorative graphic on the left side of the slide, consisting of a network of white lines and circles on a blue gradient background. The lines are vertical and horizontal, with some diagonal segments, and the circles are of varying sizes, resembling a circuit board or a digital network.

# DIGITAL DESIGN

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

2022 FALL TERM

# LAB3

- Verilog
  - Bitwise and logic operations in Verilog
- Design mode in Verilog
  - 1. data flow
- Vivado
  - Schematic of “TRL analysis” (Gates)
  - Schematic of “Synthesis” (LUT of FPGA chip)

# BITWISE AND LOGICAL OPERATIONS IN VERILOG

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

# DESIGN MODE IN VERILOG - DATA FLOW

- 1. 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
$f(a,b,c) = abc + a'b$	<code>assign f = a &amp; b &amp; c   ~a &amp; b;</code>
$f(a,b,c) = \sum(2,4,5,6) = a'bc' + ab'c' + ab'c + abc'$	<code>assign f = ~a &amp; b &amp; ~c   a &amp; ~b &amp; ~c   a &amp; ~b &amp; c   a &amp; b &amp; ~c;</code>

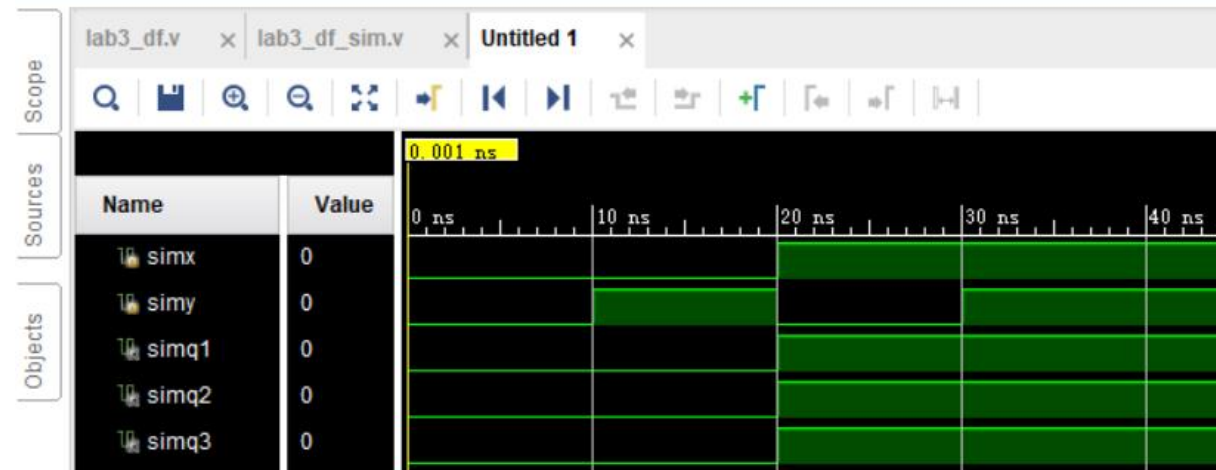
# DATA FLOW DESIGN

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

```
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
```

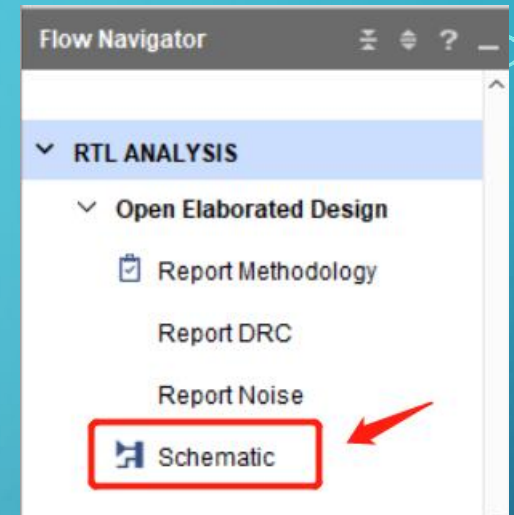
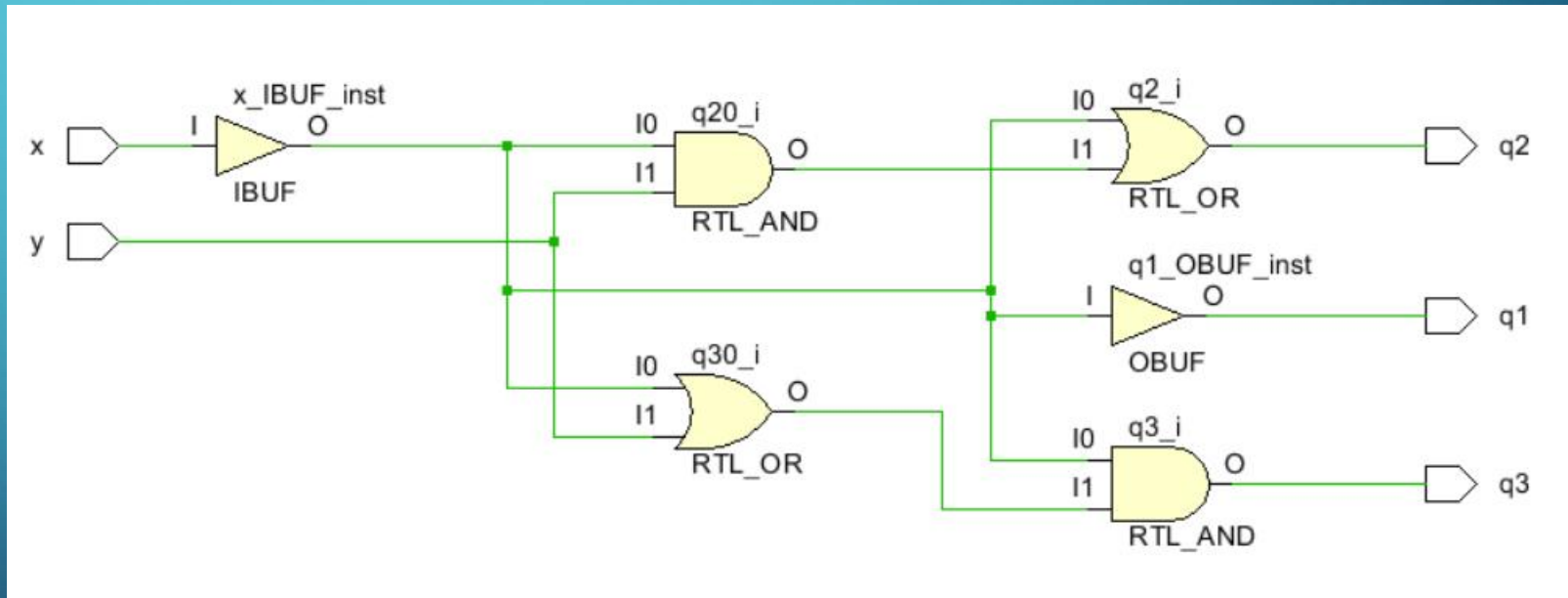
```
module lab3_df_sim( );  
    reg simx, simy;  
    wire simq1, simq2, simq3;  
    lab3_df u_df(  
        .x(simx), .y(simy), .q1(simq1), .q2(simq2), .q3(simq3) );  
  
    initial  
    begin  
        simx=0;  
        simy=0;  
        #10  
        simx=0;  
        simy=1;  
        #10  
        simx=1;  
        simy=0;  
        #10  
        simx=1;  
        simy=1;  
    end  
endmodule
```

SIMULATION - Behavioral Simulation - Functional - sim\_1 - lab3\_df\_sim



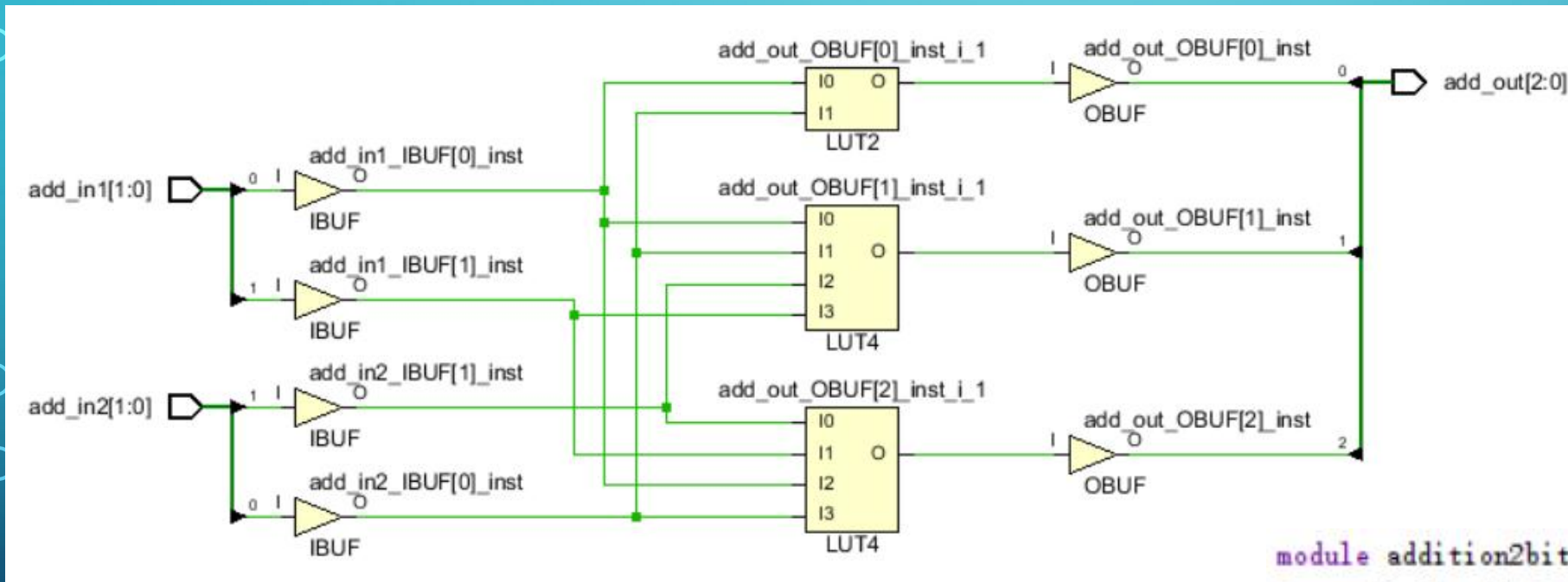
# SCHEMATIC IN 'RTL ANALYSIS'

```
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)



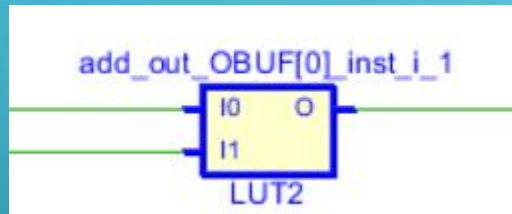
## SYNTHESIS

- ▶ Run Synthesis <sup>1</sup>
- ▼ Open Synthesized Design <sup>2</sup>
  - Constraints Wizard
  - Edit Timing Constraints
  - Set Up Debug
  - Report Timing Summary
  - Report Clock Networks
  - Report Clock Interaction
  - Report Methodology
  - Report DRC
  - Report Utilization
  - Report Power
  - Schematic <sup>3</sup>

```
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

The 'Cell Properties' window for the cell 'add\_out\_OBUF[0]\_inst\_i\_1'. The window displays the truth table for the LUT2 cell. The truth table is enclosed in a red box. Below the truth table, there is a button labeled 'Edit LUT Equation...'. At the bottom of the window, there is a tabbed interface with tabs for 'Properties', 'Power', 'Nets', 'Cell Pins', and 'Truth Table'. The 'Truth Table' tab is selected and highlighted with a red box and a red arrow.

I1	I0	O=I0 & !I1 + !I0 & I1
0	0	0
0	1	1
1	0	1
1	1	0



# 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

- Design a circuit to get the addition of two two-bit unsigned numbers:
  - **In the design, the operator “+” in verilog is not allowed here.**
  - Build a test bench to verify the function of your design.
  - Programme 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]' + a[1] a[0] 'b[1] b[0]+`

`a[1] a[0] b[1]' b[0] + a[1] a[0] b[1]' b[0] + 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).

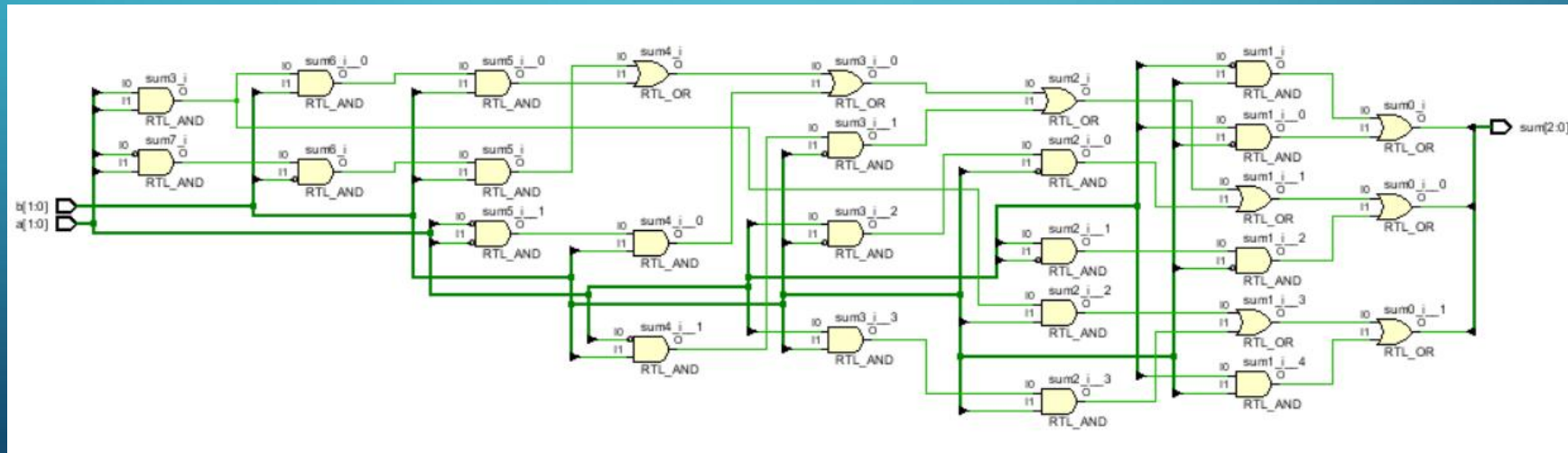
**assign** `sum[2] = ~a[1] & a[0] & b[1] & b[0] | a[1] & ~a[0] & b[1] & ~b[0] | ...`

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

- Design a circuit to get the addition of two two-bit unsigned numbers:
  - In the design:
    - the operator “+” in verilog is 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).

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

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

$$\begin{aligned}
 \text{SUM}[1] = & ( \sim a[1] \& a[0] \& \sim b[1] \& b[0] ) \mid ( a[1] \& a[0] \& b[1] \& b[0] ) \mid \\
 & ( \sim a[1] \& \sim a[0] \& b[1] ) \mid ( \sim a[1] \& b[1] \& \sim b[0] ) \mid \\
 & ( a[1] \& \sim b[1] \& \sim b[0] ) \mid ( a[1] \& \sim a[0] \& \sim b[1] );
 \end{aligned}$$

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

sum[2]		b[1]b[0]			
		00	01	11	10
a[1]a[0]	00				
	01			1	
	11		1	1	1
	10			1	1

sum[2]		b[1]b[0]			
		00	01	11	10
a[1]a[0]	00				
	01			1	
	11		1	1	1
	10			1	1

$$\text{sum}[2] = (a[1]\&a[0]\&b[0]) \mid (a[0]\&b[1]\&b[0]) \mid (a[1]\&b[1])$$