Digital Logic

Lab2

Design Flow

Verilog: Basics

- Verilog is:
  - Case sensitive
  - Based on the programming language C
- Comments
  - Single Line

    // [end of line]
  - Multiple Line

    /*

    */
- List element separator: ,
- Statement terminator: ;
- Useful links: https://www.asic-world.com/

Verilog: Identifiers

- Identifiers are names used to give an object, such as a register or a function or a module, a name so that it can be referenced from other places in a description.
  - Identifiers must begin with an alphabetic character or the underscore character (a-z A-Z _)
  - Identifiers may contain alphabetic characters, numeric characters, the underscore, and the dollar sign (a-z A-Z 0-9 _ $)
  - Identifiers can be up to 1024 characters long.
- Identifier Examples
  - Scalar: A, C0, my&clk, a_1,
  - Vector: sel[0:2], f0[0:5], ACC[31:0],

Verilog: Numbers

- Syntax: <size>'<radix><value>;
- Binary Values for numbers
  - 0
  - 1
  - X,x - Unknown
  - Z,z – High impedance state (open circuit)
- Numbers
  - 6'hCA = 001010
  - 6'hA = 001010
  - 6'b1 = 000001
  - 4'b0101= 0101
  - 8'bxxxx = 0000xxxx
  - 'hF = 00000000000000000000000000001111

'd'/D: Decimal
'b'/B: Binary
'o'/O: Octonary
'h'/H: Hexadecimal

Verilog: Keywords

- Keyword is a special identifier reserved in the language for defining the language structure. Keyword all lowercase
  - **module**, **endmodule**
  - **input**, **output**
  - **wire**, tri, **reg**
  - assign
  - **initial**, always
  - **begin**, **end**
  - *Primitive* gates : **and**, **nand**, **or**, **nor**, **xor**, **xnor**, **not**, **buf**

module and port

- Modules are the building blocks of Verilog designs
  - You create the design hierarchy by instantiating modules in other modules.
  - Module name should be same as file name
- Ports allow communication between a module and its environment.
  - All but the top-level modules in a hierarchy have ports.

```
module myCircuit (sw, led );
   input [23:0] sw;
   output [23:0] led;

   assign led=sw;
endmodule
```

or

```
module myCircuit (
   input [23:0] sw,
   output [23:0] led
   );
   assign led=sw;
endmodule
```

wire and reg

- Verilog Language has two primary data types:
  - Nets - represent structural connections between components.
    - **wire**: Interconnecting wire - no special resolution function
      - default value is ‘X’
    - tri: Net pulls-down or pulls-up when not driven
  - Registers - represent variables used to store data.
    - **reg**: Registers store the last value assigned to them until another assignment statement changes their value
      - default value is ‘Z’

Primitives

- Primitive Gates
  - buf, not, and, or, nand, nor, xor, xnor
  - Syntax:
    - gate_operator instance_identifier (output, input_1, input_2, …)
  - Examples:

```
and A1 (F, A, B); //F = A B
or O1 (w, a, b, c)
    O2 (x, b, c, d, e);    //w=a+b+c,x=b+c+d+e
```

```
// F = AB+C

module test(
    input A,input B,input C,output F);

wire and1_or1;
and  and1(and1_or1, A, B);
or     or1(F, and1_or1, C);

endmodule
```

Test bench

- An HDL description that provides the stimulus to a design is called a test bench. It's a virtual platform for simulating input and output verification in real environments .
- Within the test bench:
  - The inputs to the circuit are declared with keyword reg and the outputs are declared with the keyword wire.
  - The module *my_hw_tb* is instantiated with the instance name dut (Every instantiation of a module must include a unique instance name).

Example:

```
module my_hw_tb( );
reg [23:0] sw_sim=24'h00_0000;        //sw_sim is used to connect to the input of the  tested module
wire [23:0] led_sim;                  // led_sime is used to connect to the output of the  tested module
hw dut(.sw(sw_sim), .led(led_sim));      //instantiate the unit, do the connection
// Add your stimulis here
endmodule
```

Test bench: delay

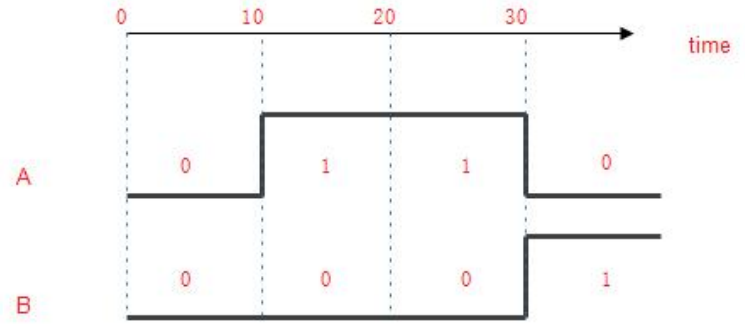
- timescale
  - `timescale 1ns / 1ps
    - The first number specifies the unit of measurement for time delays. The second number specifies the precision for which the delays are rounded off, in this case to 0.001 ns. If no timescale is specified, a simulator may display dimensionless values or default to a certain time unit, usually 1 ns (=10^{-9} s). Our examples will use only the default time unit.

- Propagation delay
  - # 10 (10 ns delay for timescale 1ns/1ps)

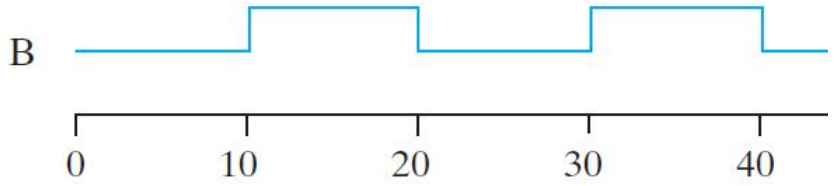
Test bench: stimuli

- Initial Blocks
  - An initial block, is executed only once when simulation starts. This is useful in writing test benches. If we have multiple initial blocks, then all of them are executed at the beginning of simulation.

```
initial begin
    A = 1'b0; B = 1'b0;
    #10 A = 1'b1;
    #20 A = 1'b0; B = 1'b1;
end
```



- Always Blocks
  - an always block executes always, unlike initial blocks which execute only once (at the beginning of simulation). A second difference is that an always block should have a sensitive list or a delay associated with it.

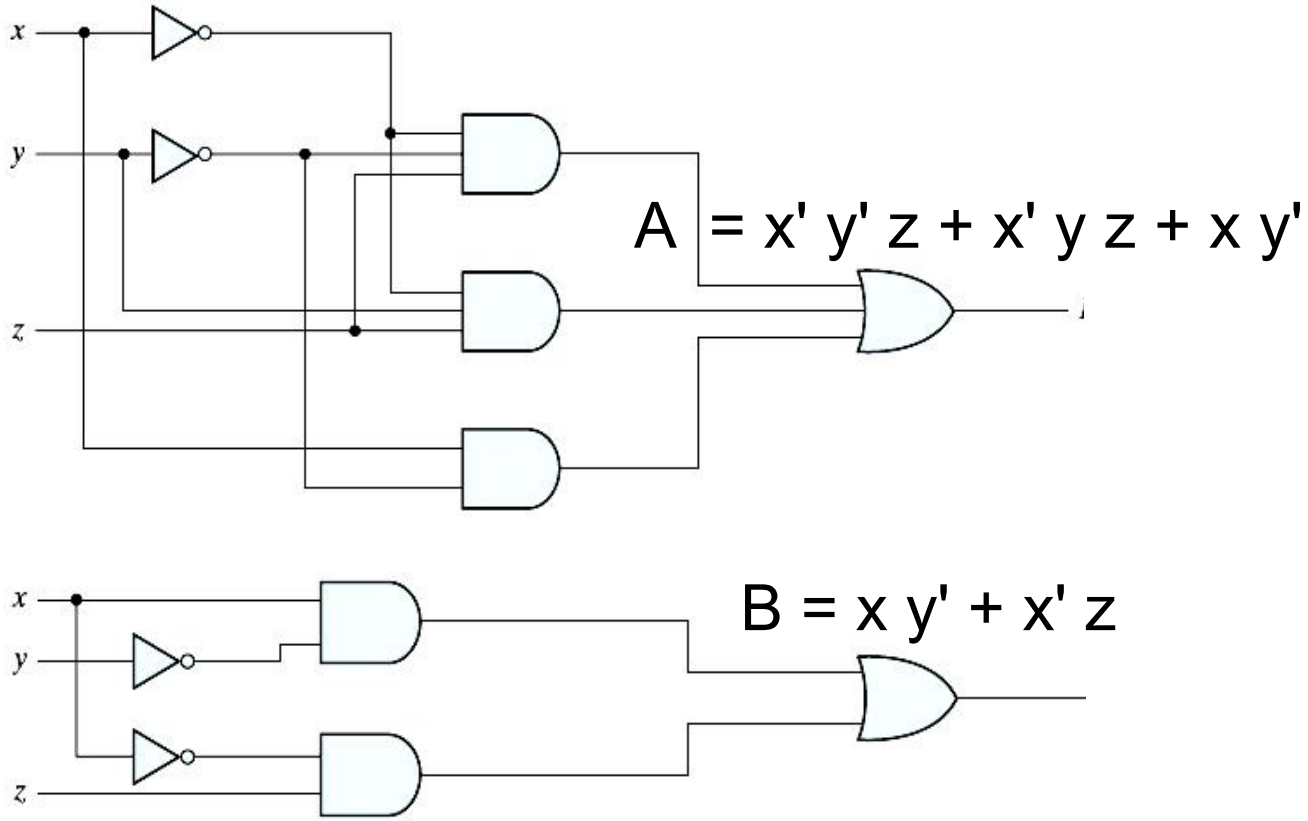



```
always  begin
   #10  B = ~B;
end
```

Recall: Boolean Functions

- The truth table of 2n entries

x	y	z	A	B
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	0	0
1	1	1	0	0



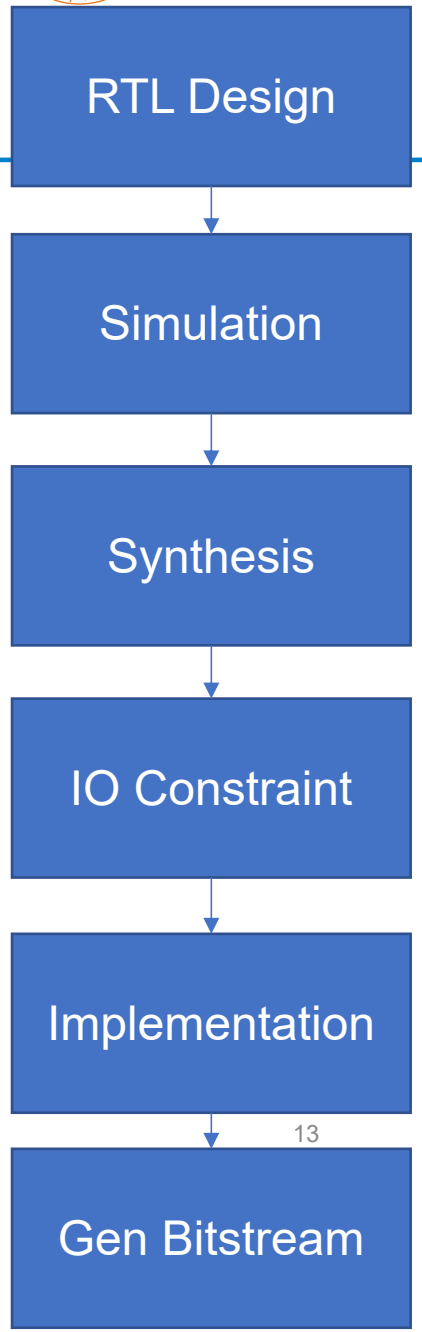
- Note: Two Boolean expressions may specify the same function: A = B
- B is more economical

Practice1: Verilog Design Flow

- 1. Create a Verilog file, save as **lab2_hw.v.** Design a circuit using Verilog that implements:

  A = x' y' z + x' y z + x y'

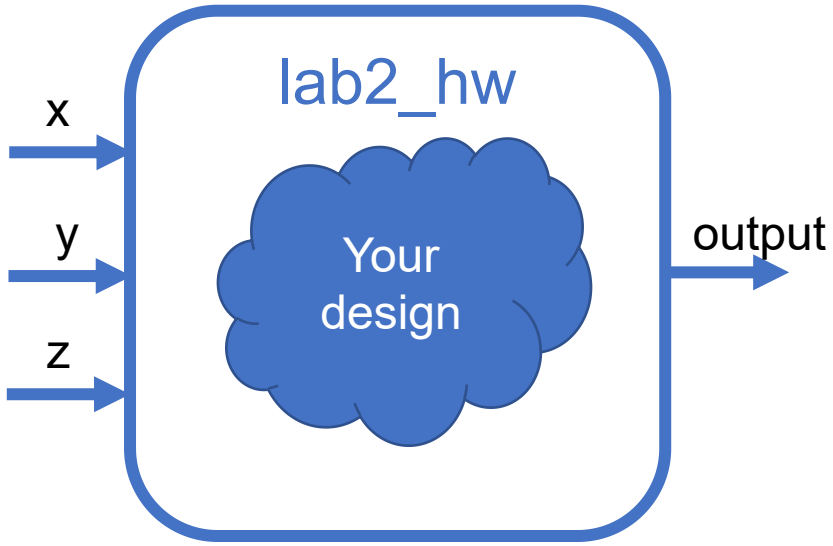
- 2. Write a simulation file, save as **lab2_hw_tb.v.** Write your testbench and Run the "**Simulation**" to check waveform.
- 3. Run "**RTL Analysis**" to check gate level schematic
- 4. Run "**Synthesis**"
- 5. Connect IO pins in "**Constraint wizard**"
- 6. Run "**Implementation**" and "**Generate Bitstream**"
- 7. Program your FPGA with bitstream for on board and test
- 8. **Call TAs** to show your schematic, waveform and on-board result for **Lab Attendance**



baiyh@sustech.edu.cn

Tips1: structure design

- 1. Create a Verilog file, save as lab2_hw.v
- Design a circuit using Verilog that implements:
  - A = x' y' z + x' y z + x y'

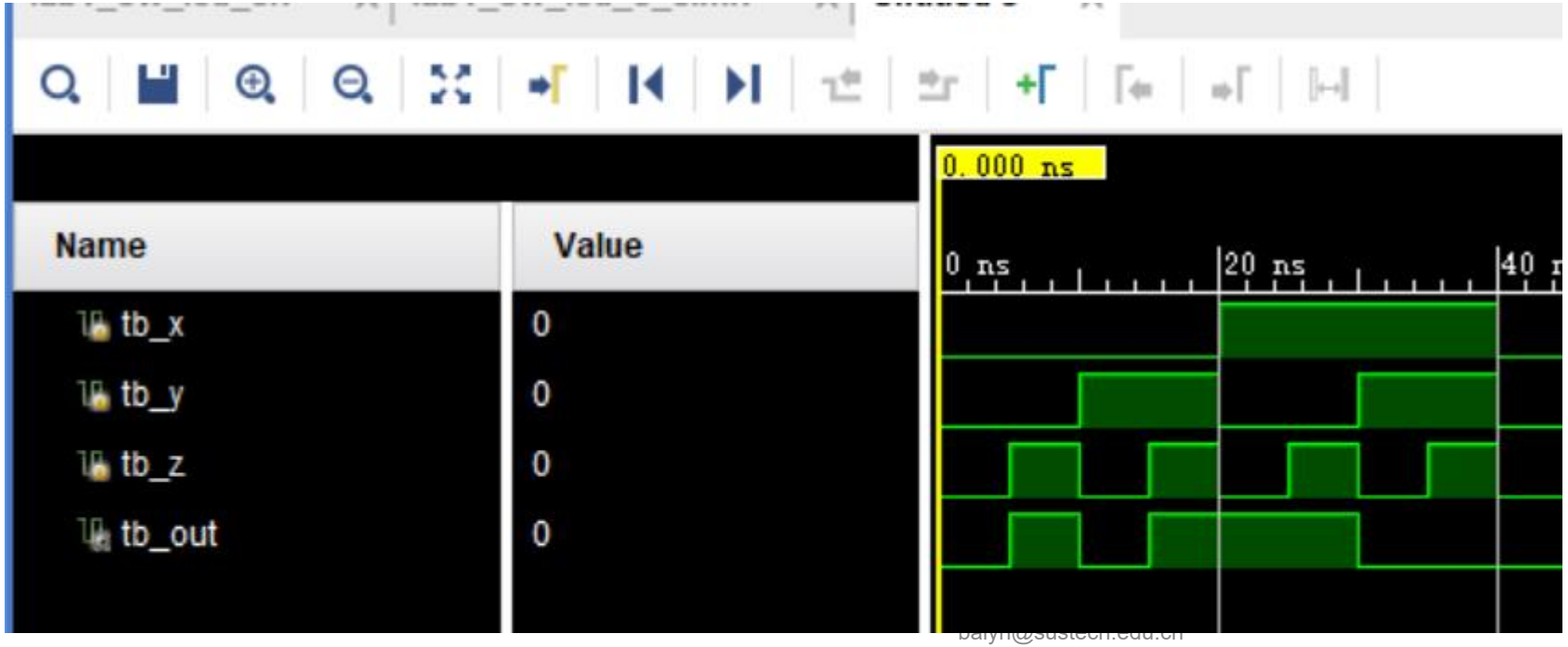



```
module lab2_hw(
    input x,
    input y,
    input z,
    output out);
    wire notx, noty, notz;
    wire out1, out2, out3;

    not nx(notx, x);
    // Complete your code here
    or u4(out, out1, out2, out3);

endmodule
```

Tip2: simulation

- Create a simulation file, save as lab2_hw_tb.v
- Instantiate your top module as dut(design for test), add stimulis, Compare waveform with truth table.

x	y	z	A	B
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	0	0
1	1	1	0	0

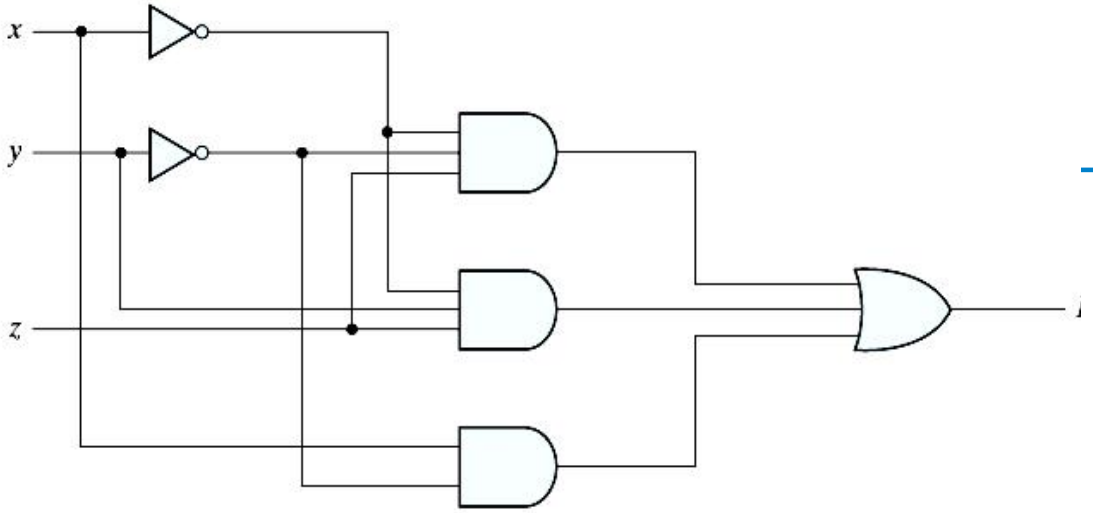


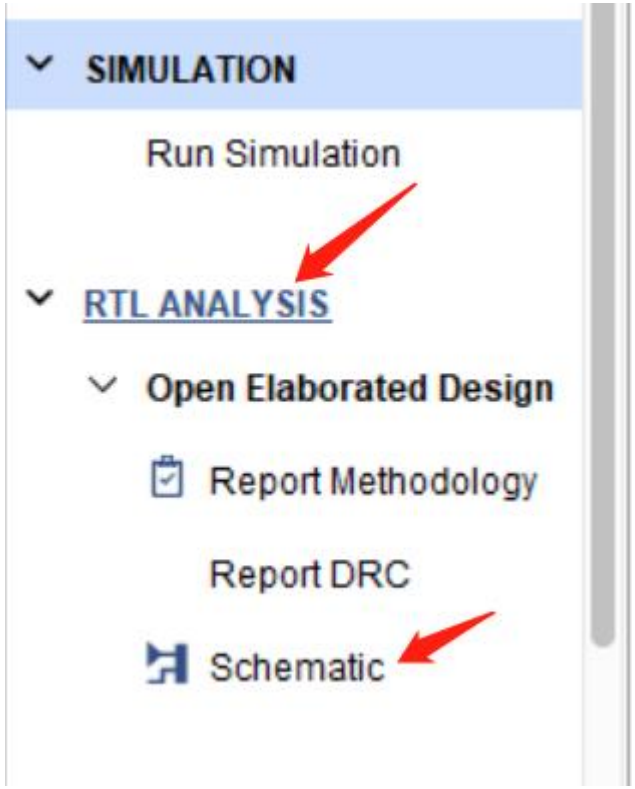

```
module lab2_hw_tb( );
//connect to input
reg tb_x;
reg tb_y;
reg tb_z;

//connect to output
wire tb_out;
//instantiate the unit
lab2_hw dut(
.x(tb_x),
.y(tb_y),
.z(tb_z),
.out(tb_out)
);

initial begin...

always begin...
```

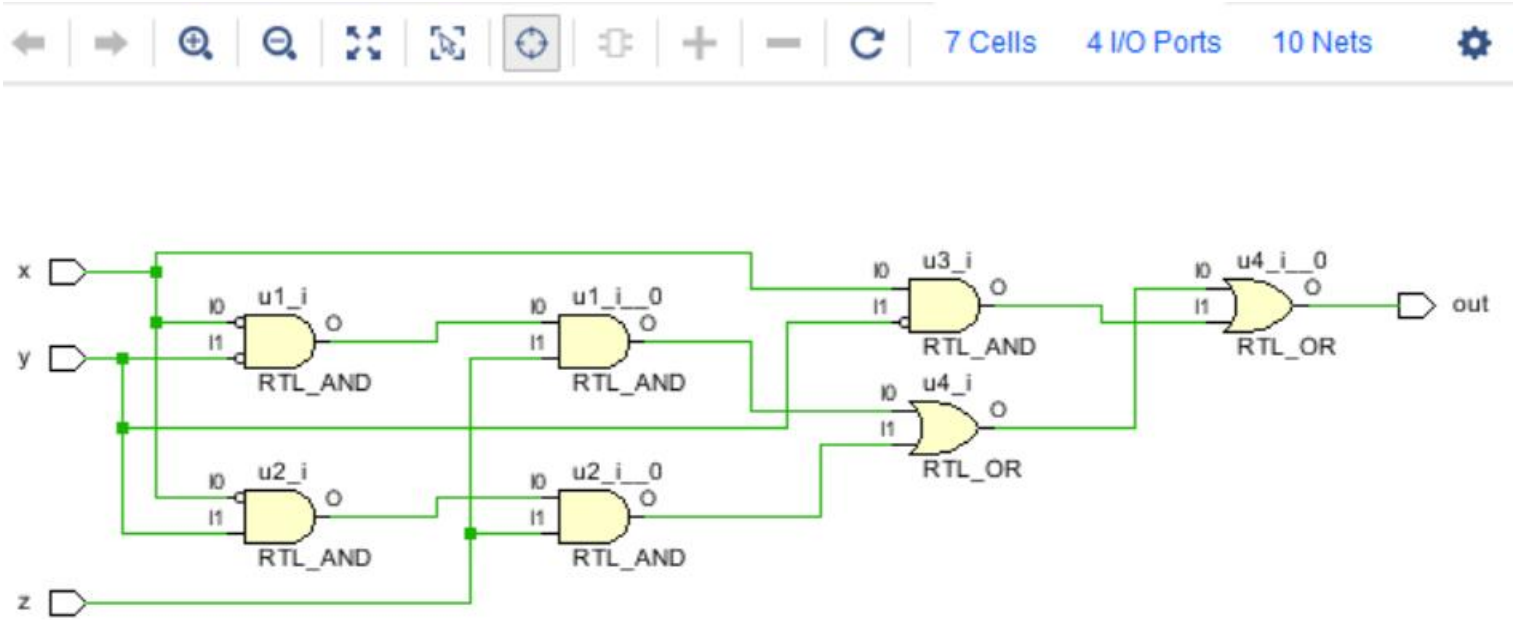
baiyh@sustech.edu.cn

Tip3: RTL Analysis

- Generate pre-synthesis schematic and compare it with the reference gate level design.





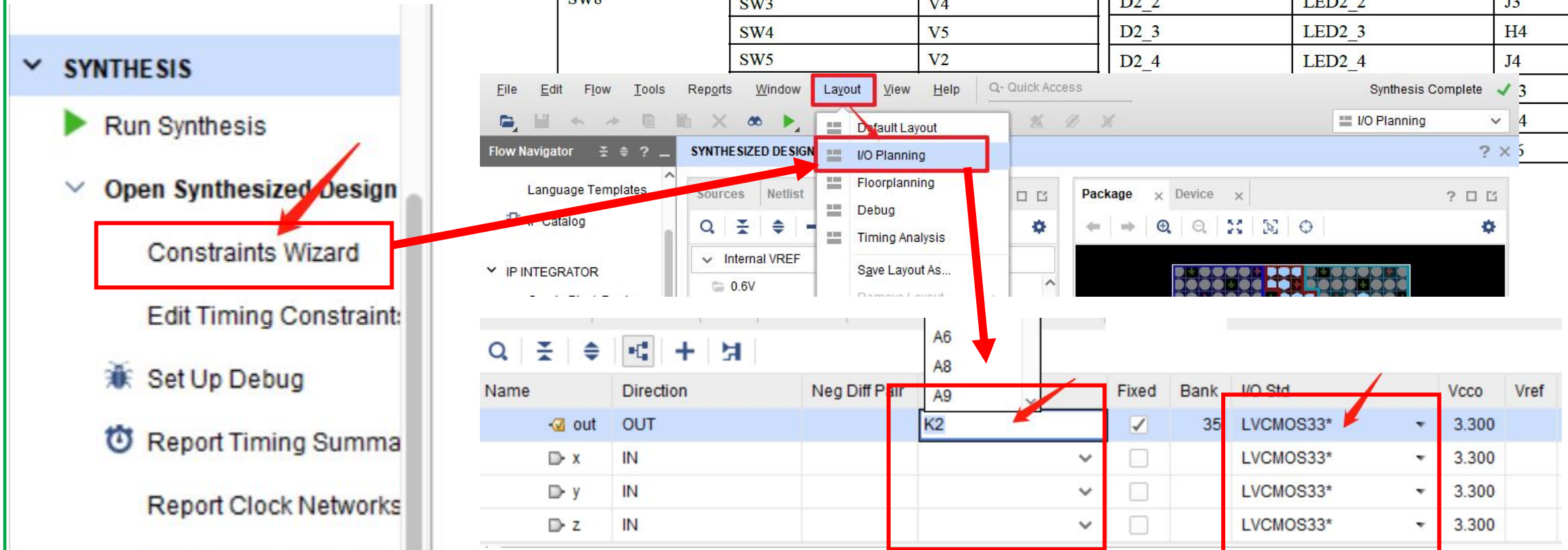


Tip4: Constraint

- Connect the I/O ports to the package pin and set their I/O Std.

名称	原理图标号	FPGA IO PIN
SW0	SW_0	R1
SW1	SW_1	N4
SW2	SW_2	M4
SW3	SW_3	R2
SW4	SW_4	P2
SW5	SW_5	P3
SW6	SW_6	P4
SW7	SW_7	P5
SW8	SW0	T5
	SW1	T3
	SW2	R3
	SW3	V4
	SW4	V5
	SW5	V2

名称	原理图标号	FPGA IO PIN
D1_0	LED1_0	K3
D1_1	LED1_1	M1
D1_2	LED1_2	L1
D1_3	LED1_3	K6
D1_4	LED1_4	J5
D1_5	LED1_5	H5
D1_6	LED1_6	H6
D1_7	LED1_7	K1
D2_0	LED2_0	K2
D2_1	LED2_1	J2
D2_2	LED2_2	J3
D2_3	LED2_3	H4
D2_4	LED2_4	J4



Practice2 (optional)

- 1. Change your design and implement:

  B = x y' + x' z

- 2. Check simulation waveform, RTL Analysis schematic and on board test, is this circuit has same functionality with the previous one? Which one is more economical?

baiyh@sustech.edu.cn