DIGITAL DESIGN

LAB4　PARAMETER OF VERILOG & PACKAGE CUSTOMIZABLE IP CORE & USE IT

2021 FALL TERM

LAB4

- Parameter in Verilog
- Package Customizable IP CORE
- Use The Customizable IP CORE to Do The Design

PARAMETER

- **Parameter** is used to improve the readability and maintainability of code.
  - Used to define constants, such as delay and width variables, which is equivalent to defining an identifier representing a constant, also known as a symbolic constant.
  - They're run-time constants, and can be overridden when instantiate the module.
  - Syntax for parameter definitions :

    parameter *param_name1* = value/expression,
    param_name2 = value/expression, …
    - *Use comma to separate different parameter definitions.*
- The definition of the parameter is partial and **valid only in the current module.**
- The parameter definition can use the previously defined integer and real parameter.
- *All the following illustration base on the IP wrapper of NOR gate*

PARAMETER DEFINITION(1)

```
module norgate
#(parameter Port_Num = 2,WIDTH = 1) (
    input [(WIDTH - 1) : 0] a,    input [(WIDTH - 1) : 0] b,
    input [(WIDTH - 1) : 0] c,    input [(WIDTH - 1) : 0] d,
    input [(WIDTH - 1) : 0] e,    input [(WIDTH - 1) : 0] f,
    input [(WIDTH - 1) : 0] g,    input [(WIDTH - 1) : 0] h,
    output [(WIDTH - 1) : 0] q
    );
    assign q = ~(a | b | c | d | e | f | g | h);
endmodule
```

PARAMETER DEFINITION(2)

No #

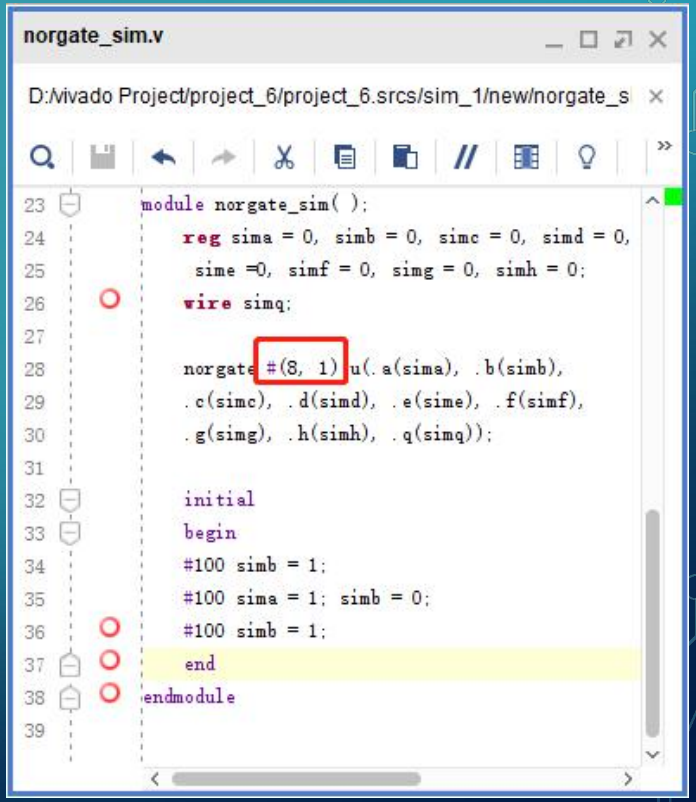
```
module norgate(a, b, c, d, e, f, g, h, q);
    parameter Port_Num = 2, WIDTH = 1;
    input[WIDTH -1 : 0] a, b, c, d, e, f, g, h;
    output[WIDTH -1 : 0]q;
    assign q = ~ (a | b | c | d | e | f | g | h);
endmodule
```

PARAMETER OVERRIDE

- The value of the parameter can be overridden when instantiate the module designed in the design source.

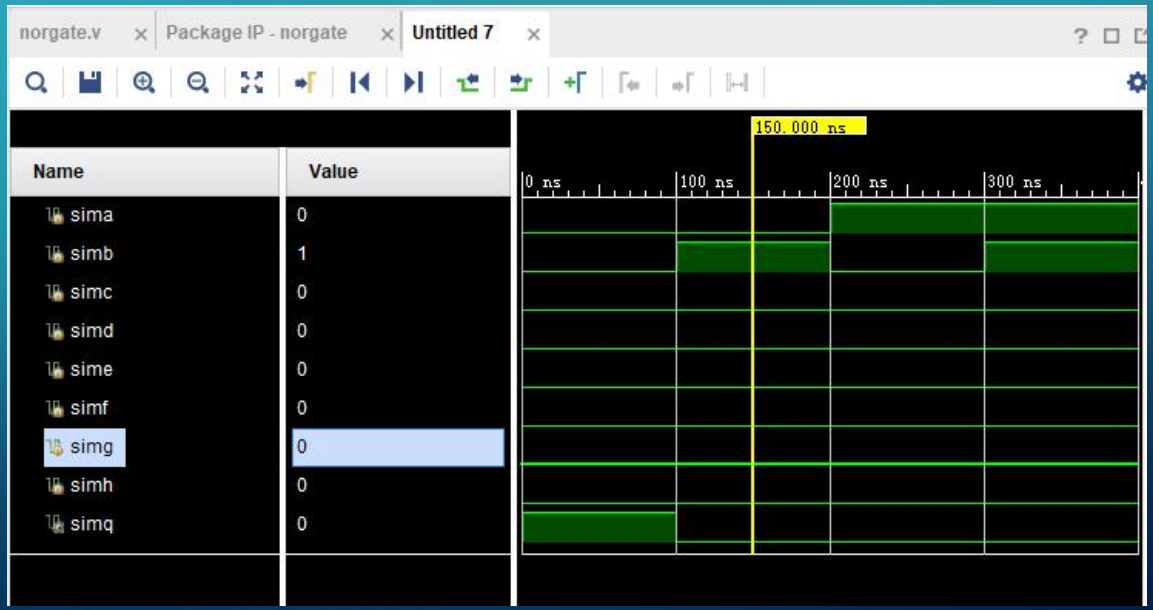
when instantiate the module andgate,

we customize the Port_Num as 8 and WIDTH as 1.



- Using simulation to verify the function of the module in the design file.

Once the simulation result is correct we can start IP packaging.



IP CORE

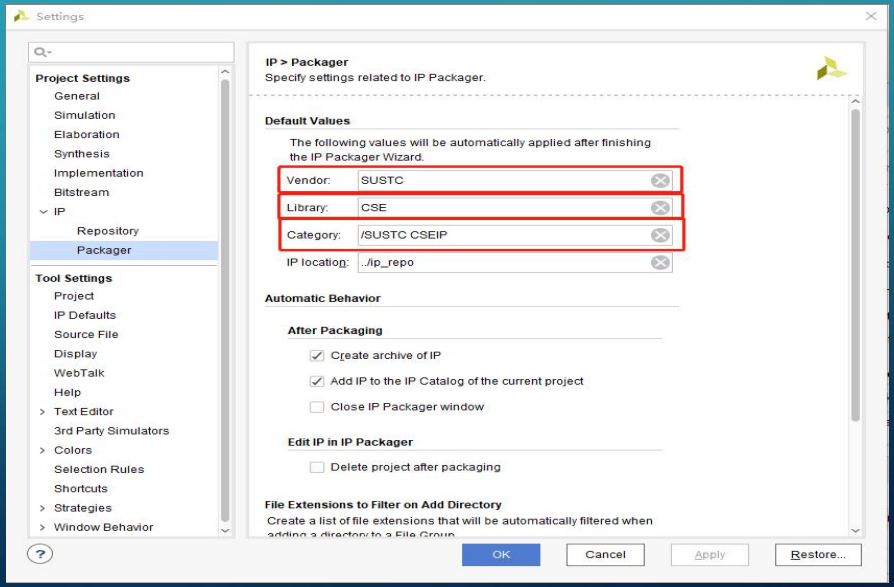
- **IP core**, or **IP block** is a reusable unit of logic, cell, or integrated circuit (commonly called a "chip") layout design that is the intellectual property of one party. IP cores may be licensed to another party or can be owned and used by a single party alone. The term is derived from the licensing of the patent and/or source code copyright that exist in the design. IP cores can be used as building blocks within application-specific integrated circuit (ASIC) designs or field-programmable gate array (FPGA) logic designs.
- IP cores (software)are typically offered as synthesizable RTL. Synthesizable cores are delivered in a hardware description language such as Verilog or VHSIC hardware description language (VHDL).

----- from wiki

- A IP created by using vivado

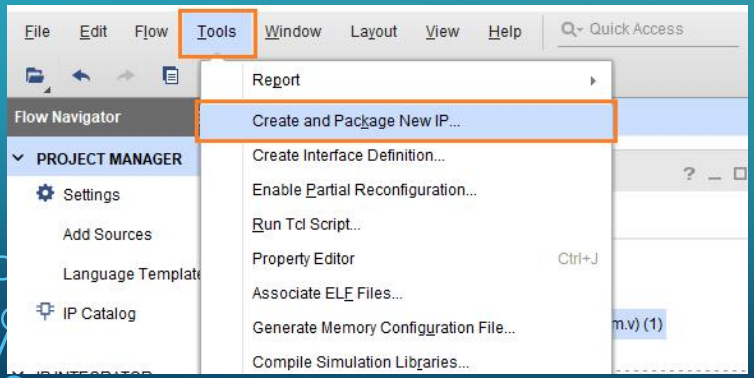
PACKAGE CUSTOMIZABLE IP CORE(1)

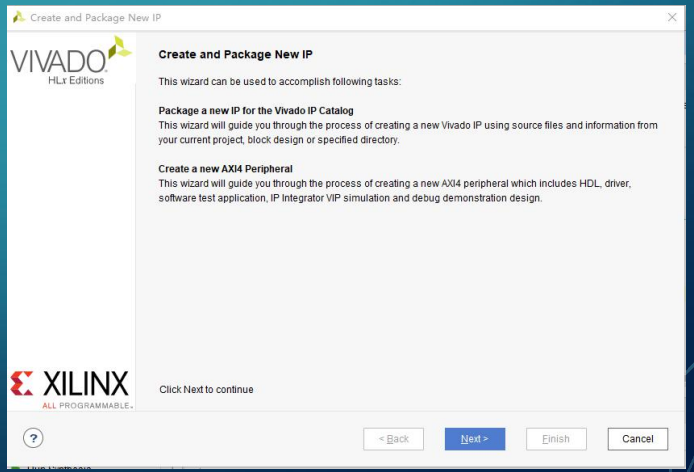
- Click the Flow Navigator—> Project Manager—> Setting—>IP—>Packager
  - Modify the vendor to SUSTC, and Library to CSE. These two properties will be part of the IP file name.
  - Modify the category to SUSTC CSEIP. This property will be displayed as category name.



PACKAGE CUSTOMIZABLE IP CORE(2)

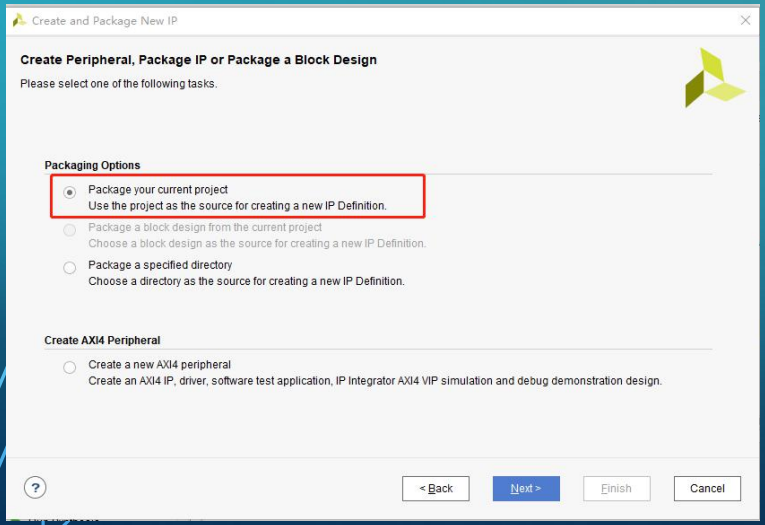
- Click the Tools —>Create and Package New IP…
- Follow the wizard

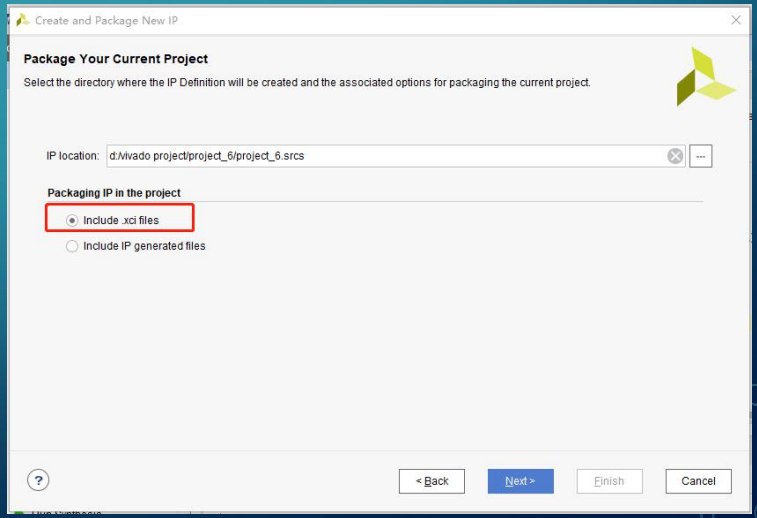




PACKAGE CUSTOMIZABLE IP CORE(3)

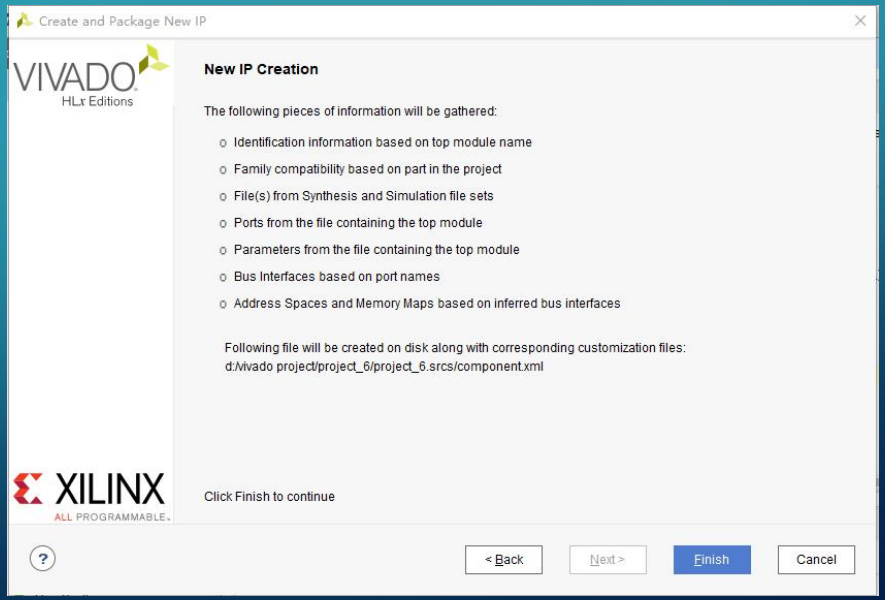
- Choose *package your current project*
  - choose *'include .xci file'*





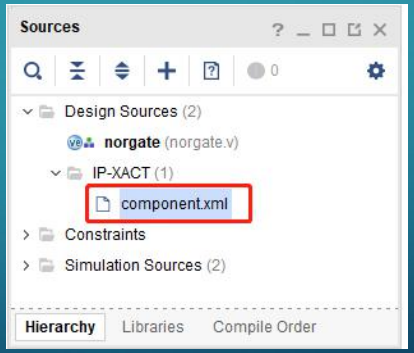
PACKAGE CUSTOMIZABLE IP CORE(4)

- Click finish here



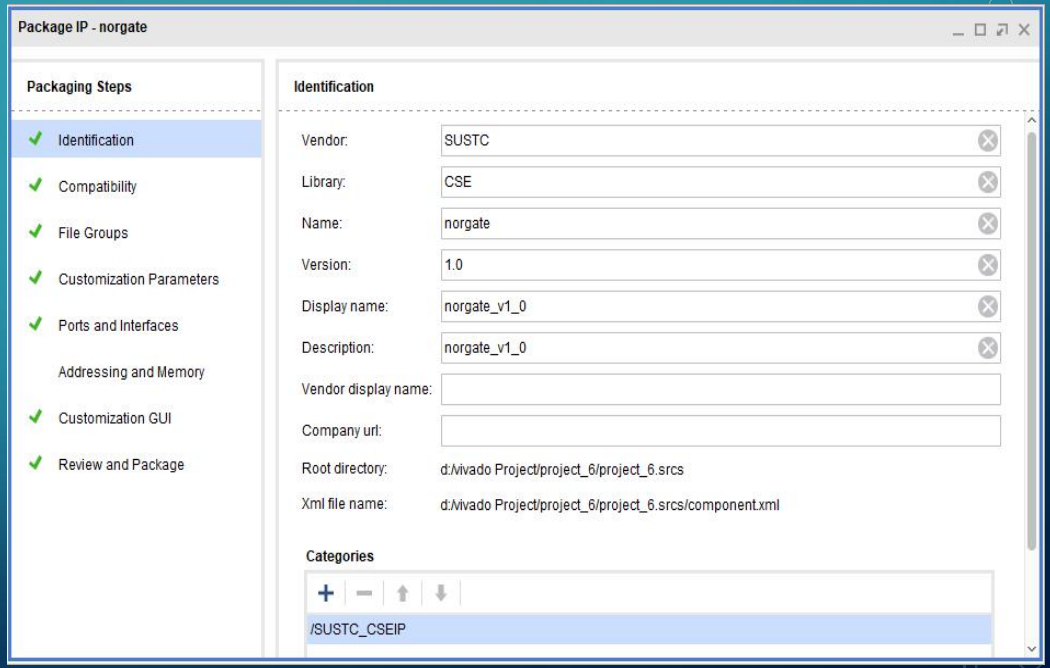
PACKAGE CUSTOMIZABLE IP CORE(5)

- A file named as component.xml will be added to your design sources automatically



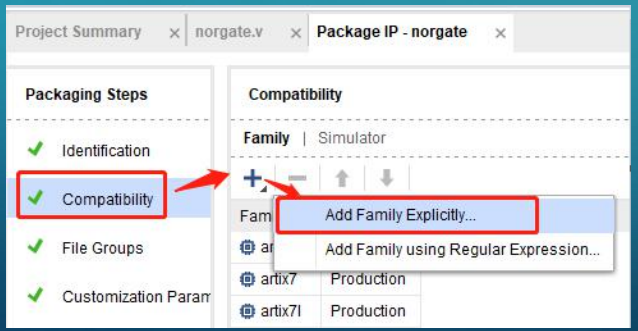
CUSTOMIZATION IP CORE(1)-IDENTIFICATION

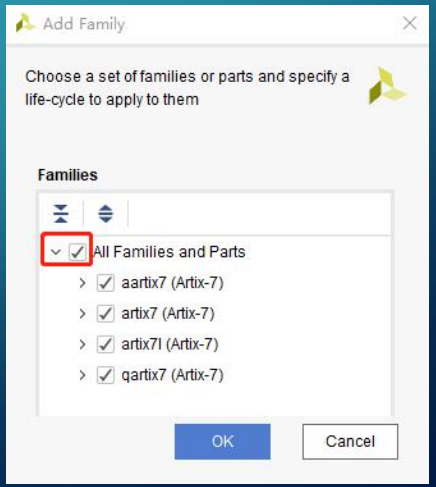
- If you want to modify those properties, you can do it. Here we just keep it as it is.



CUSTOMIZATION IP CORE(2)-COMPATIBILITY(1)

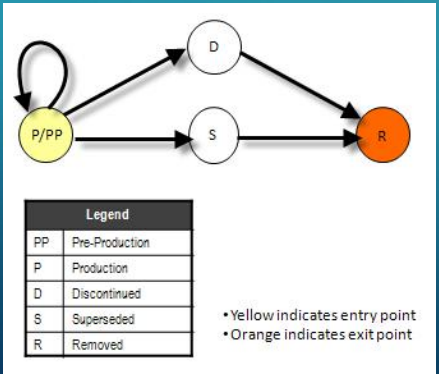
- If you want the IP be valid with other FPGA Family, we can add them all.

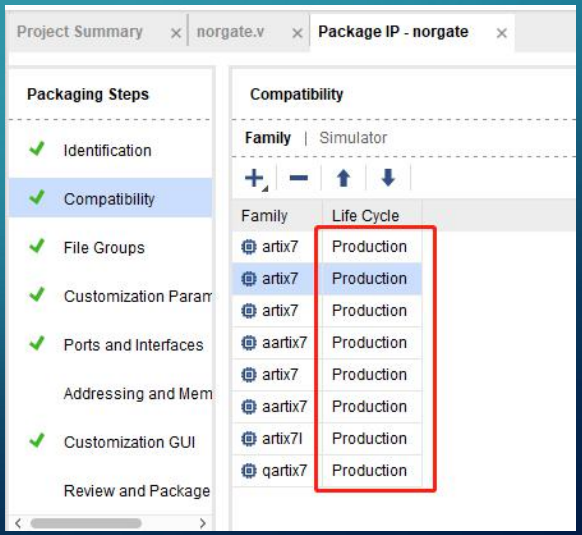




CUSTOMIZATION IP CORE(2)-COMPATIBILITY(2)

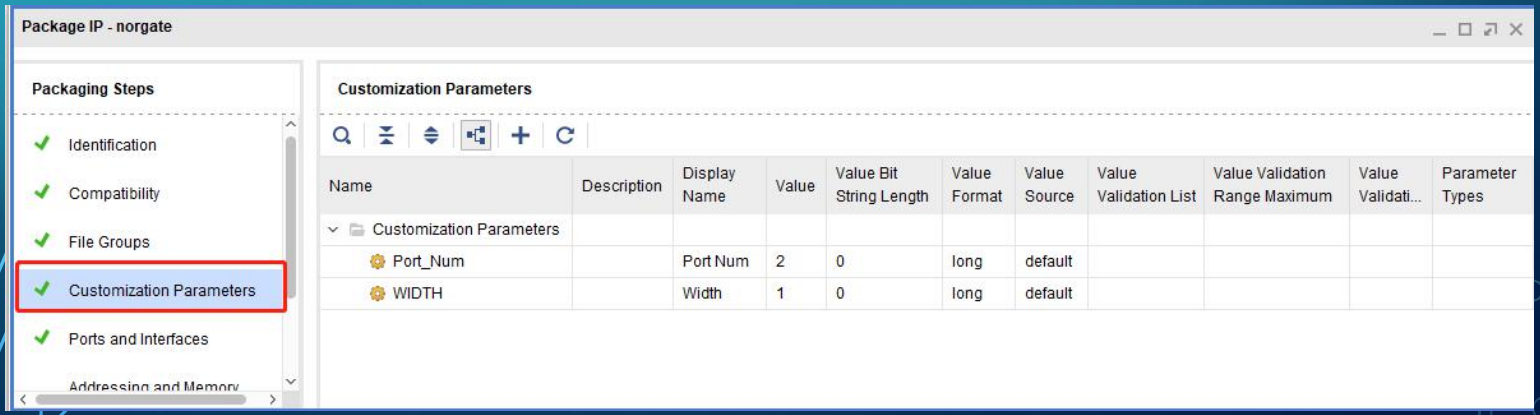
- As for the *life circle* , we choose '*Production*'.
- A Production IP core is one which is provided for general public release, and has been verified using production speed files.





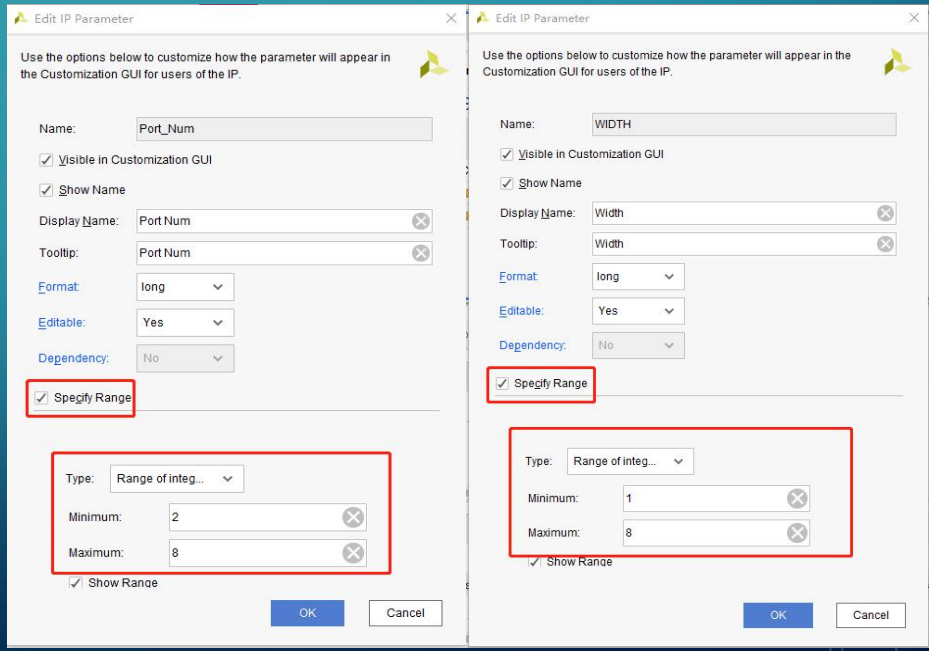
CUSTOMIZATION IP CORE(3)- CUSTOMIZATION PARAMETERS(1)

- Here we can customize the defined parameters, double-click the parameter name in the table, you can edite the relevant information.



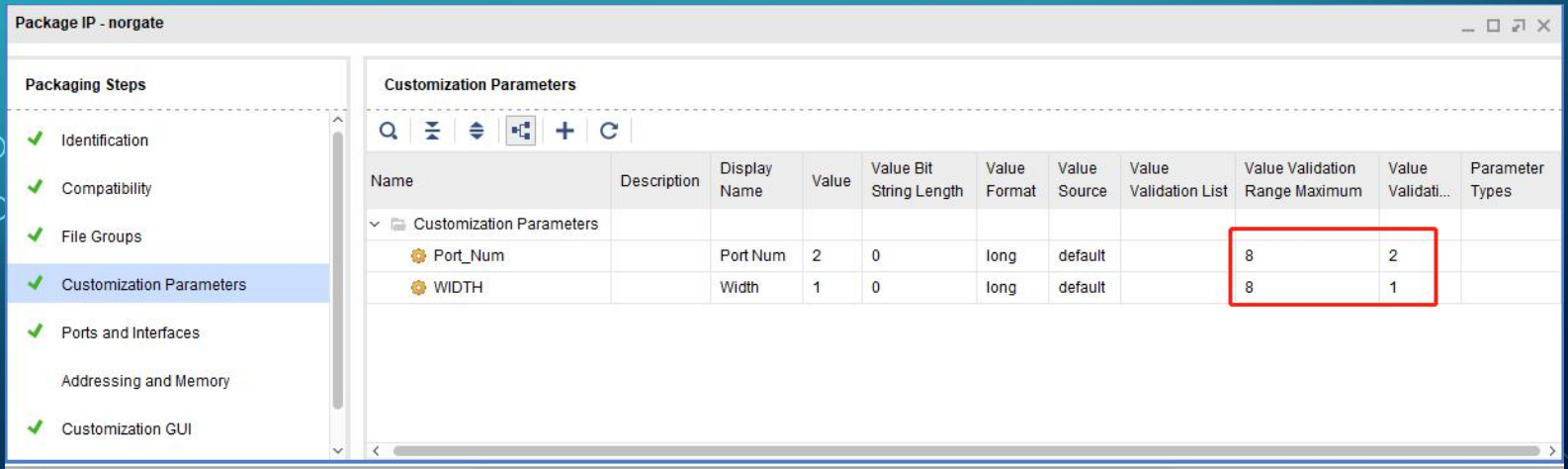
CUSTOMIZATION IP CORE(3)- CUSTOMIZATION PARAMETERS(2)

- For a NOR gate, there are at least two inputs; here eight inputs has been defined.
  - for the value of parameter 'Port_Num', the minimum is 2, and maximum is 8.
- As for the parameter WIDTH, the minimum is 1, and maximum is 8.



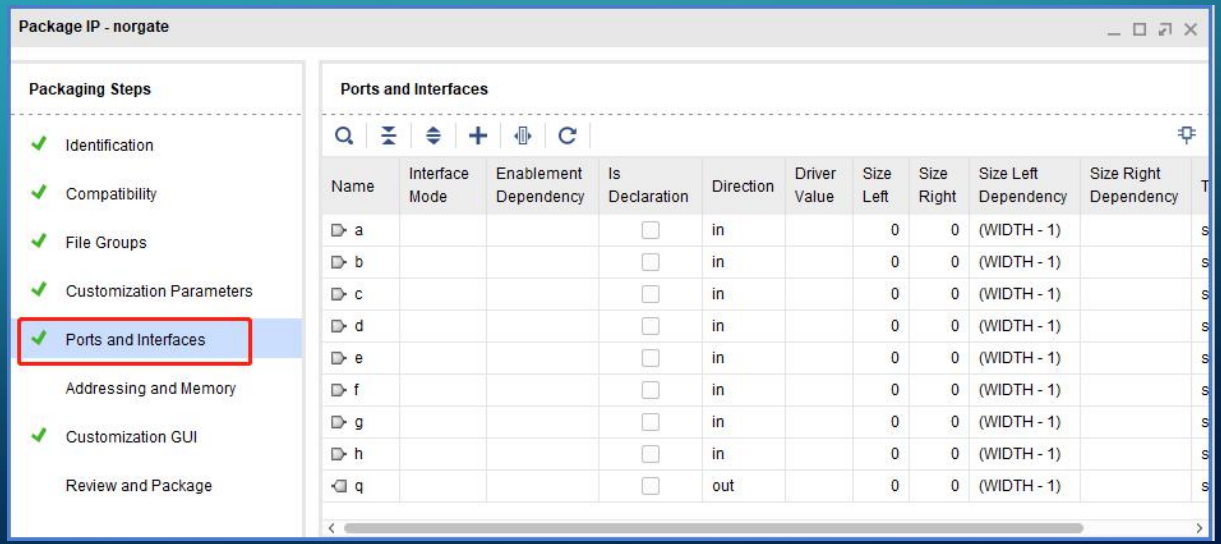
CUSTOMIZATION IP CORE(3)- CUSTOMIZATION PARAMETERS(3)

- After the setting , we can see the following changes which are marked by red box.



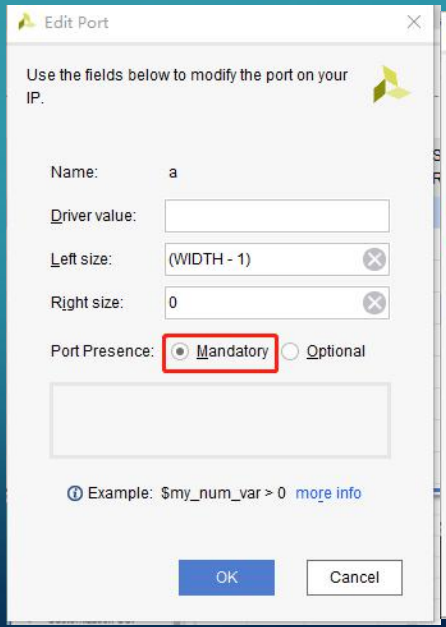
CUSTOMIZATION IP CORE(4)- PORTS AND INTERFACES(1)

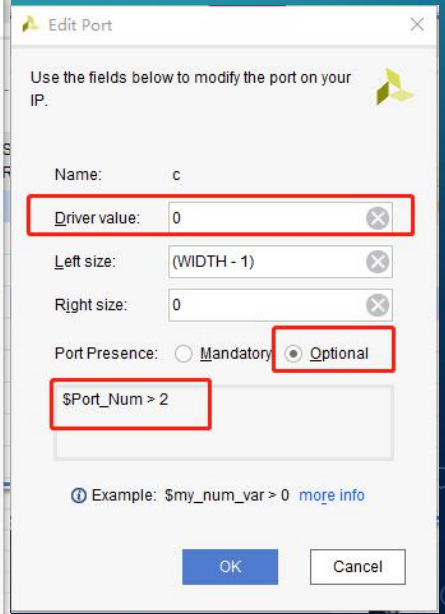
- Here we set properties of ports. Double-click the port name in the table, you can edit the relevant information.



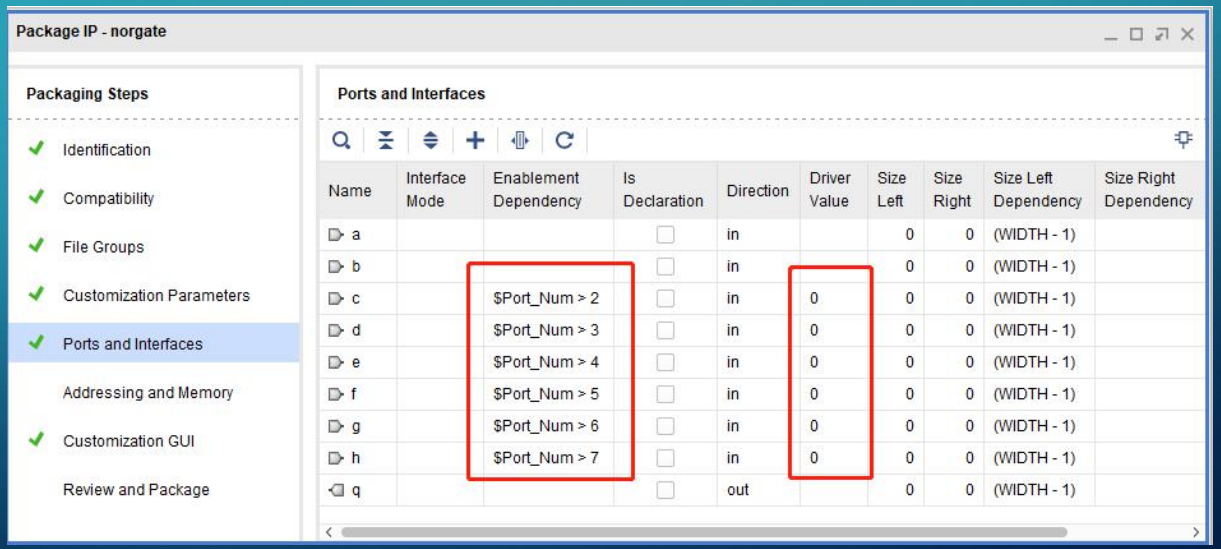
CUSTOMIZATION IP CORE(4)- PORTS AND INTERFACES(2)

- There are at least two input ports, so the port named ‘a’ and ‘b’ are Mandatory while others are Optional.
  - In this case, whether the ports named as c ~h is enabled is determined by the value of parameter Port_Num.
- In a Nor gate , if the input port is disabled, the ‘ Driver value ’of that port should be 0.





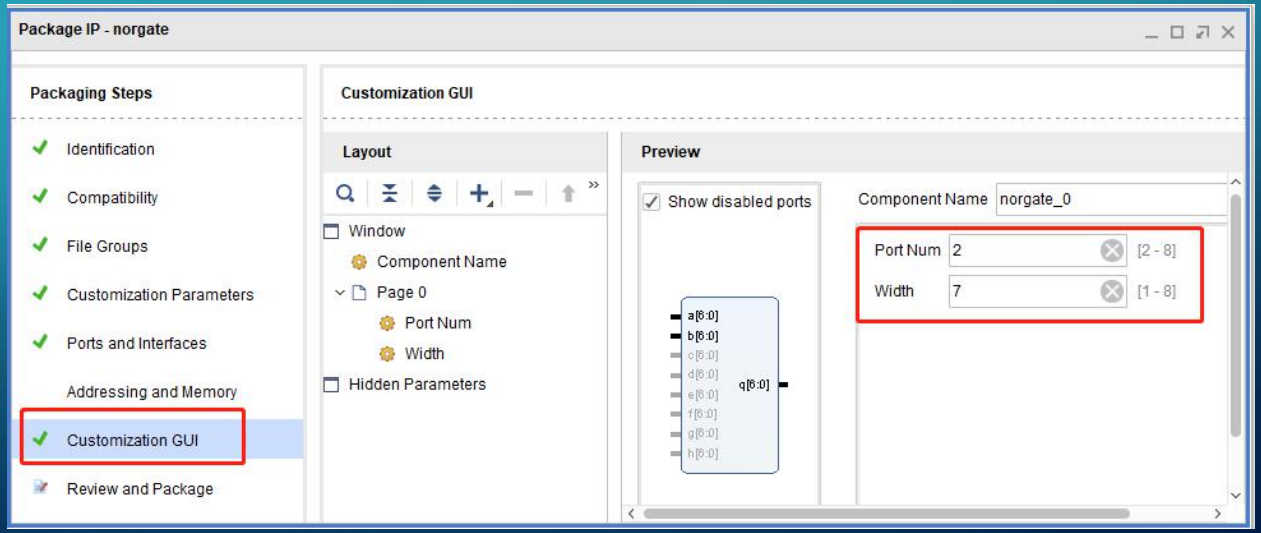
CUSTOMIZATION IP CORE(4)- PORTS AND INTERFACES(3)



Name	Interface Mode	Enablement Dependency	Is Declaration	Direction	Driver Value	Size Left	Size Right	Size Left Dependency	Size Right Dependency
a				in		0	0	(WIDTH - 1)	
b				in		0	0	(WIDTH - 1)	
c		$Port_Num > 2		in	0	0	0	(WIDTH - 1)	
d		$Port_Num > 3		in	0	0	0	(WIDTH - 1)	
e		$Port_Num > 4		in	0	0	0	(WIDTH - 1)	
f		$Port_Num > 5		in	0	0	0	(WIDTH - 1)	
g		$Port_Num > 6		in	0	0	0	(WIDTH - 1)	
h		$Port_Num > 7		in	0	0	0	(WIDTH - 1)	
q				out		0	0	(WIDTH - 1)	

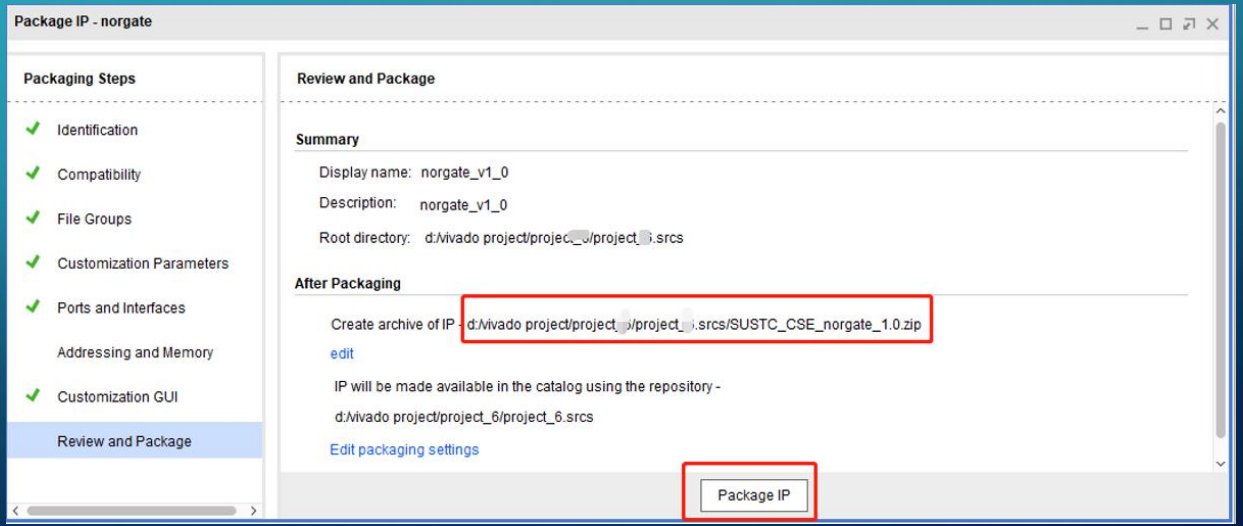
CUSTOMIZATION IP CORE(5)- CUSTOMIZATION GUI

- Here we can verify the effect of parameter changes on IP encapsulation.



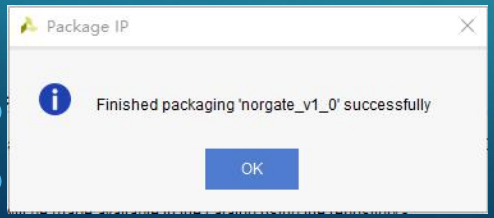
CUSTOMIZATION IP CORE(6)- REVIEW AND PACKAGE

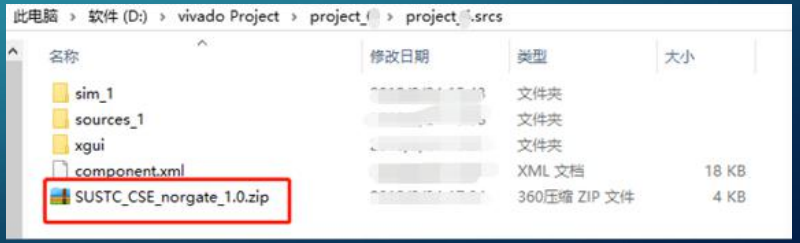
- Remember the file path where the IP core will be created. Click *Package IP*.



PACKAGE CUSTOMIZABLE IP CORE

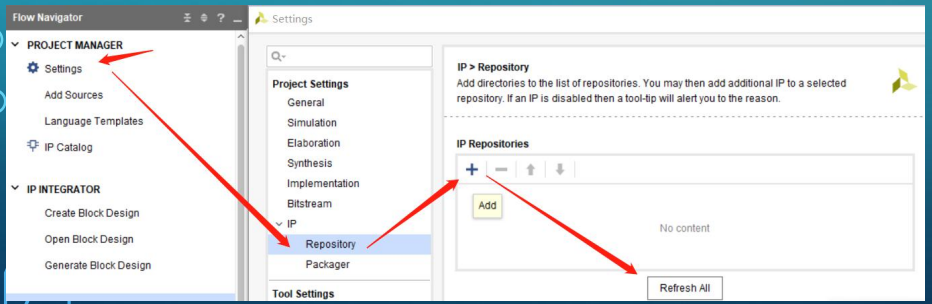
- If the packaging is success, You can find your IP core exactly under the file path showed before.
- Then you can put the IP core to your IPCore file, and decompress it, so you can used it in your following design.





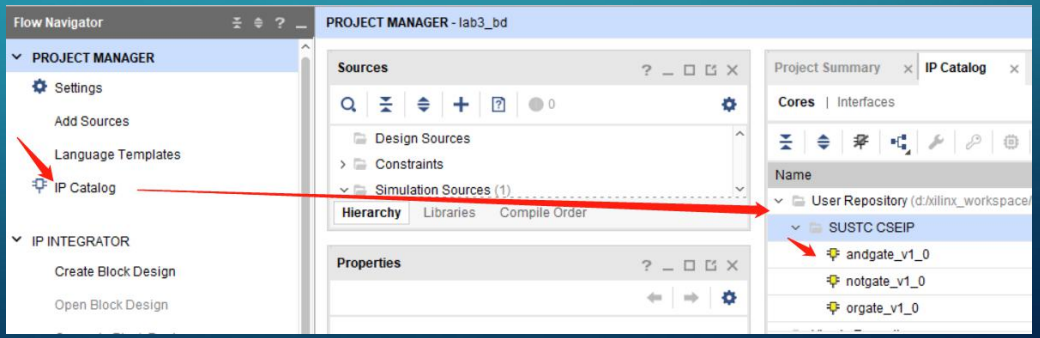
USING IP(1) - ADDING IP INTO PROJECT(1)

- Prepare for adding IP into current project:
  - 1.make you project to find the IP repositories
    - Flow navigator ->settings ->IP ->Repository (add directories to the list of repositories)
    - IP repositories is a place where stored some unzipped IPs



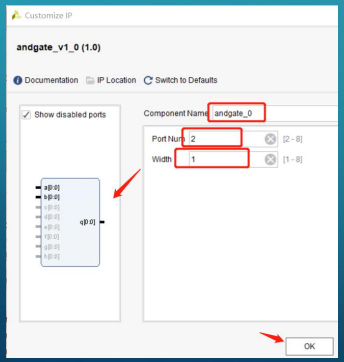
USING IP(1) - ADDING IP INTO PROJECT(2)

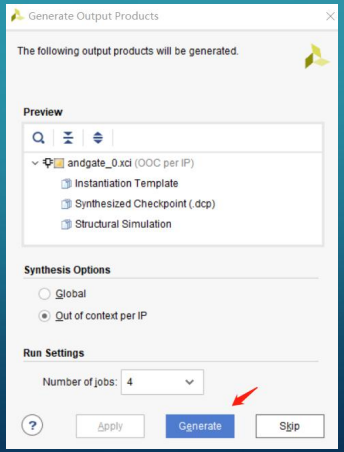
- Prepare for adding IP into current project :
  - 2.find ip and add it to your project
    - Flow navigator ->PROJECT MANGER ->IP Catalog , find the IP and double click

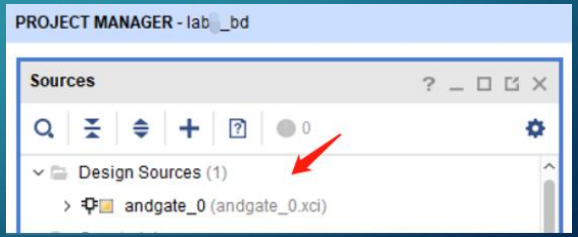


USING IP(1)- ADDING IP INTO PROJECT(3)

- Prepare for adding IP into current project :
  - 3. customize the IP , generate it so that it will be add into the source of the project ,which means the IP can be used in the project.

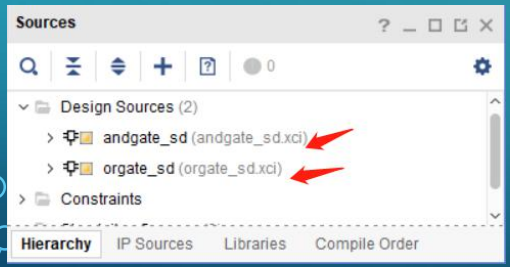


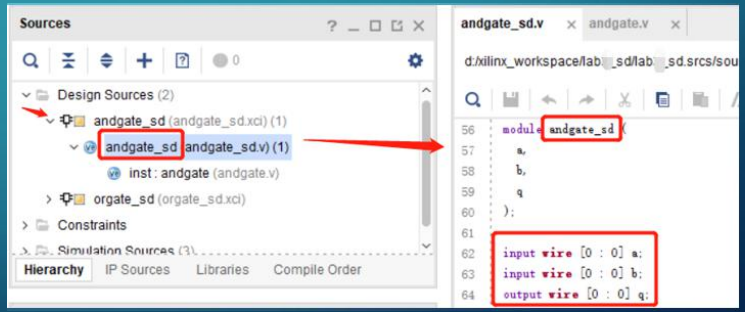




USING IP(2)
STRUCTURED DESIGN BASED ON IP(1)

- If you have added the IPs into the Design Sources of the project, you can find them, remember the module's name and ports' name:



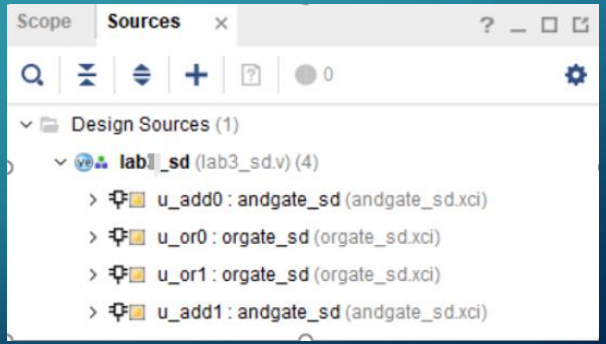


USING IP(2)

STRUCTURED DESIGN BASED ON IP(2)

1. Create a Verilog design file (Add sources -> choose Design source type)
2. Edit : instance IP , build module with IP instance.

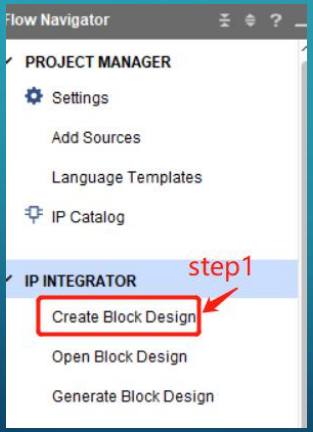
```
module lab3_sd(
    input x,
    input y,
    output q1,
    output q2,
    output q3
    );

  // q1 = x
    assign q1 = x;

  // q2 = x | (x & y )
    wire temp0;
    andgate_sd u_add0( .a(x), .b(y), .q(temp0) );
    orgate_sd u_or0(.a(temp0), .b(x), .q(q2) );

    // q3 = x & (x | y)
    wire temp1;
    orgate_sd u_or1( .a(x), .b(y), .q(temp1) );
    andgate_sd u_add1( .a(temp1), .b(x), .q(q3) );
endmodule
```

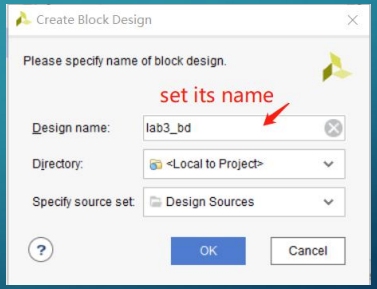


USING IP(3) - BLOCK DESIGN(1)

step 1: create a block design , then set its name

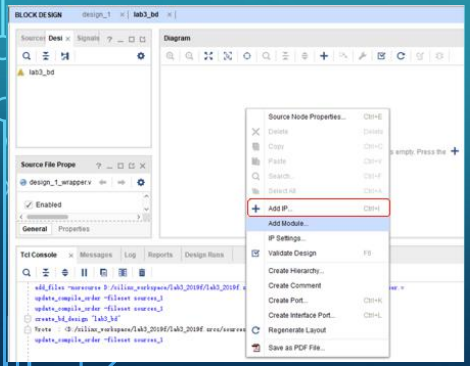
NOTIC: BLOCK DESIGN could be based on Module or IP core or both.

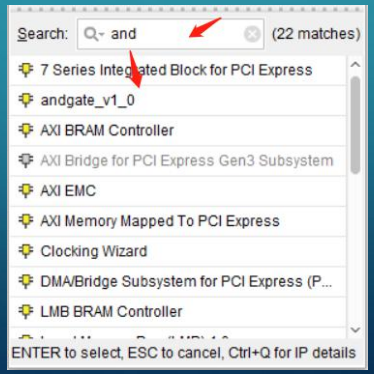


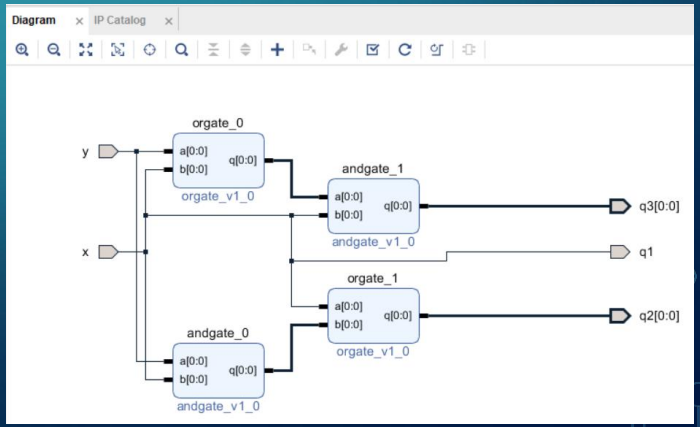


USING IP(3) - BLOCK DESIGN(2)

1. create a block design, set its name. //same with step 1 of block design on module.
2. Instance the IP, place it : Find the IP , re-customize it if needed . Then place it in the Diagram.
3. Instance the port, place it:
   Right click blank place ,choose create port (to set the name , direction and width of the port)
4. Connect the ports and IP instances
5. Check if it is ok or not







PRACTICES

- 1. Design a NAND gate, and package it to an IP Core named as SUSTC_CSE_nandgate.

Requirments: the maximum port is 8;

the minimum port is 2;

the width of the ports should range from 1 to 8.

- 2. Use the SUSTC_CSE_nandgate, SUSTC_CSE_norgate, SUSTC_CSE_notgate and SUSTC_CSE_andgate to verify the following equation. it should be realized both by structured design and block design.

$$(x+y)'(xy)' = x'y'$$

create testbench, do simulation to verify function of the design, generate bitstream file, test on the board.

TIPS : BLOCK DESIGN BASED ON MODULE

Build block design based on module is almost same as using IP core, such as create block design, creat ports, connect ports with module. The only differece is 'add module'.
NOTIC : right click blank in the ‘Diagram’ window, choose ‘Add Module’ to add a module into the block