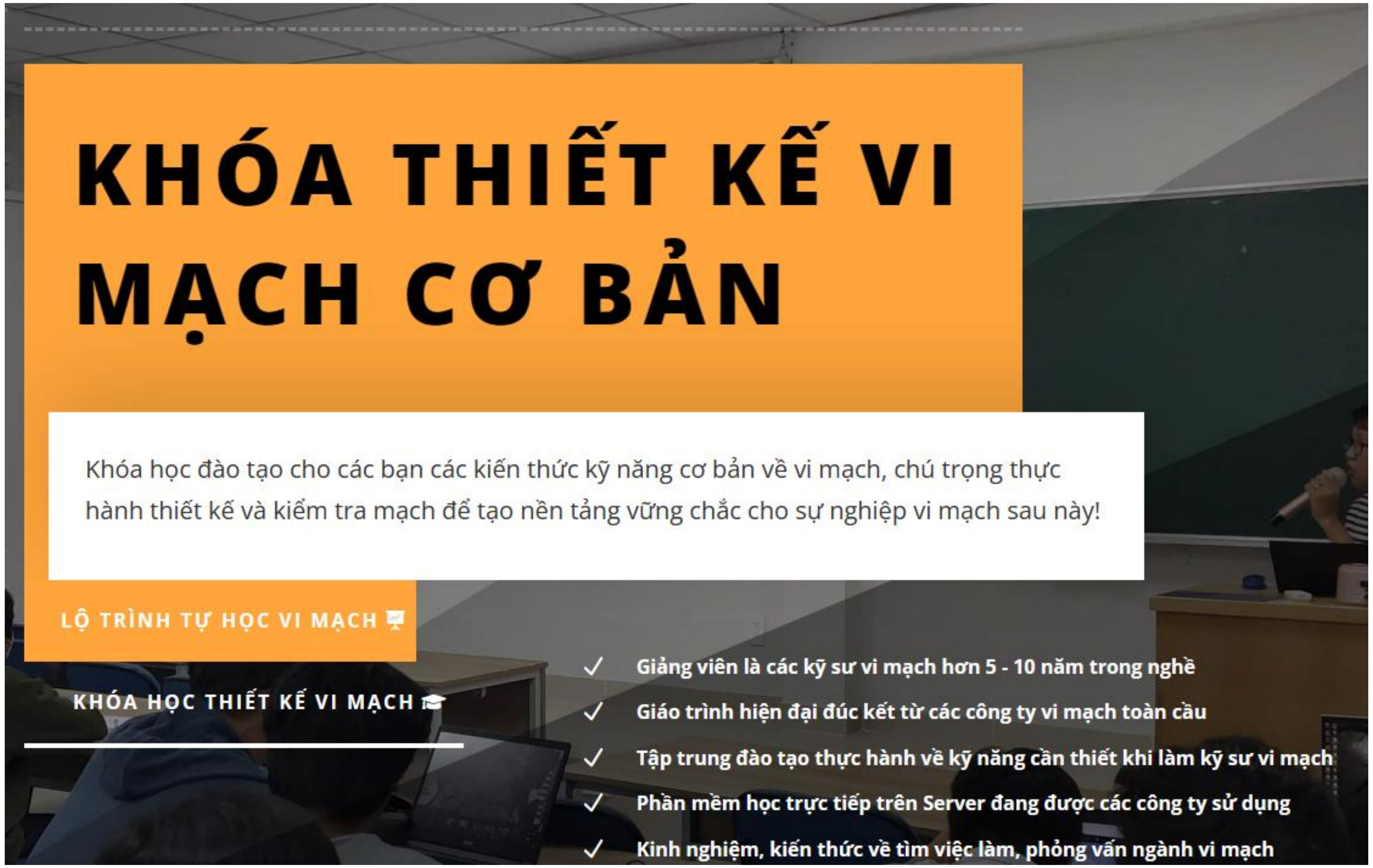
IC OVERVIEW
RTL DESIGN AND VERIFICATION

COURSE INTRODUCTION

[Khóa Học Thiết Kế Vi Mạch Cơ Bản - Trung Tâm Đào Tạo Thiết Kế Vi Mạch ICTC]



KHÓA THIẾT KẾ VI MẠCH CƠ BẢN

Khóa học đào tạo cho các bạn các kiến thức kỹ năng cơ bản về vi mạch, chú trọng thực hành thiết kế và kiểm tra mạch để tạo nền tảng vững chắc cho sự nghiệp vi mạch sau này!

- ✓ Giảng viên là các kỹ sư vi mạch hơn 5 - 10 năm trong nghề
- ✓ Giáo trình hiện đại đúc kết từ các công ty vi mạch toàn cầu
- ✓ Tập trung đào tạo thực hành về kỹ năng cần thiết khi làm kỹ sư vi mạch
- ✓ Phần mềm học trực tiếp trên Server đang được các công ty sử dụng
- ✓ Kinh nghiệm, kiến thức về tìm việc làm, phỏng vấn ngành vi mạch

COURSE INTRODUCTION

SUMMARY

HOMEWORK

QUESTION

SELF-LEARNING

Session 7: Verilog Fundamental – Part 2 Module

1. Verilog module
2. Module instance

VERILOG MODULE

A module is a fundamental building block, which can be simulated and synthesized.
It can represent anything from a simple logic gate to a complex system.
The module contains the code that describes the hardware's behavior and structure.

```
module module_name ( port_name, port_name, …);

   module_port declaration

   data_type declaration

   logic description

endmodule
```

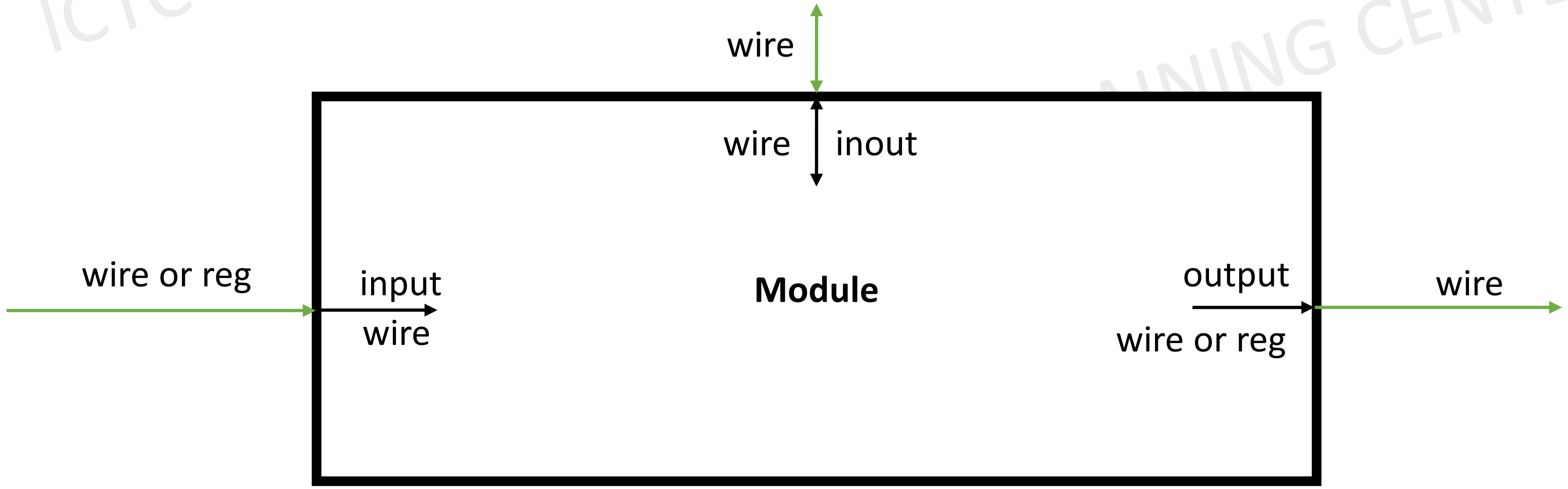
```
module and_gate( a, b, c);  //module and port list
   input a;
   input b;                  //module_port declaration
   output c;

   wire a;
   wire b;                   //data type declaration
   wire c;

   assign c = a & b;         //logic description

endmodule
```

VERILOG MODULE

module port declaration

A module port defines the interface between module and other external modules.
Ports are used to pass the signals in and out of a module.
There are 3 types of ports:

- input: receive signal into module. It should be net type.
- output: send signal out of the module. It can be net or reg type.
- inout (bidirectional port): can both receive and send signal. It should be net type.



VERILOG MODULE

module port declaration

Ports can be declared as single-bit or vector.

- **Syntax**: <**port_type**> <**port_size**> <**port_name**>;
- **Example**:
  - input write_en; //1 bit input port named "write_en"
  - input [31:0] data; //32 bit vector data input port named "data"
  - input [10:2] address; //9 bit vector part-select input port named "address"
  - output [3:2] protect; //2 bit vector part-select of output port named "protect"

Note: The vector part-select port should not be used to avoid the confusion. But it is perfectly fine with that declaration.

VERILOG MODULE

module port declaration

Port and data type can be declared together:

```
input wire write_en;
input wire [31:0] wdata;
input wire [15:2] address;
output reg ack;
output wire [31:0] rdata;
```

It’s illegal to declare the data type again for those ports as below:

```
input wire write_en;
input wire [31:0] wdata;
…
wire write_en;
wire [31:0] wdata;
```

This is illegal. The tool will report compile error.

VERILOG MODULE

module port declaration

Port and data type can be declared in port list:

```
module and_gate( a, b, c);
   input a;
   input b;
   output c;

   wire a;
   wire b;
   wire c;

   assign c = a & b;

endmodule
```

```
module and_gate
(
   input wire a,
   input wire b,
   output wire c
);
   assign c = a & b;

endmodule
```

Similar to the above slide, it's illegal to declare again the ports or data types in the body of the module if using this method.

VERILOG MODULE

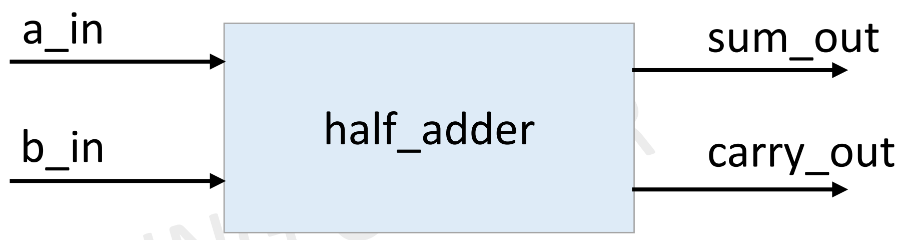
HALF ADDER

Practice: design half_adder module based on below truth table

Half-Adder is used to perform the binary addition of two binary numbers. It is called a "half-adder" because it can add two bits but does not account for any carry input from a previous stage.

1. Create 07_ss7 folder under your home directory
2. Copy /ictc/student-data/share/teacher/07_ss7/half_adder/ to your folder
3. Create the half_adder.v under rtl folder and writing your code
4. Go to sim and compile, run to check the result.

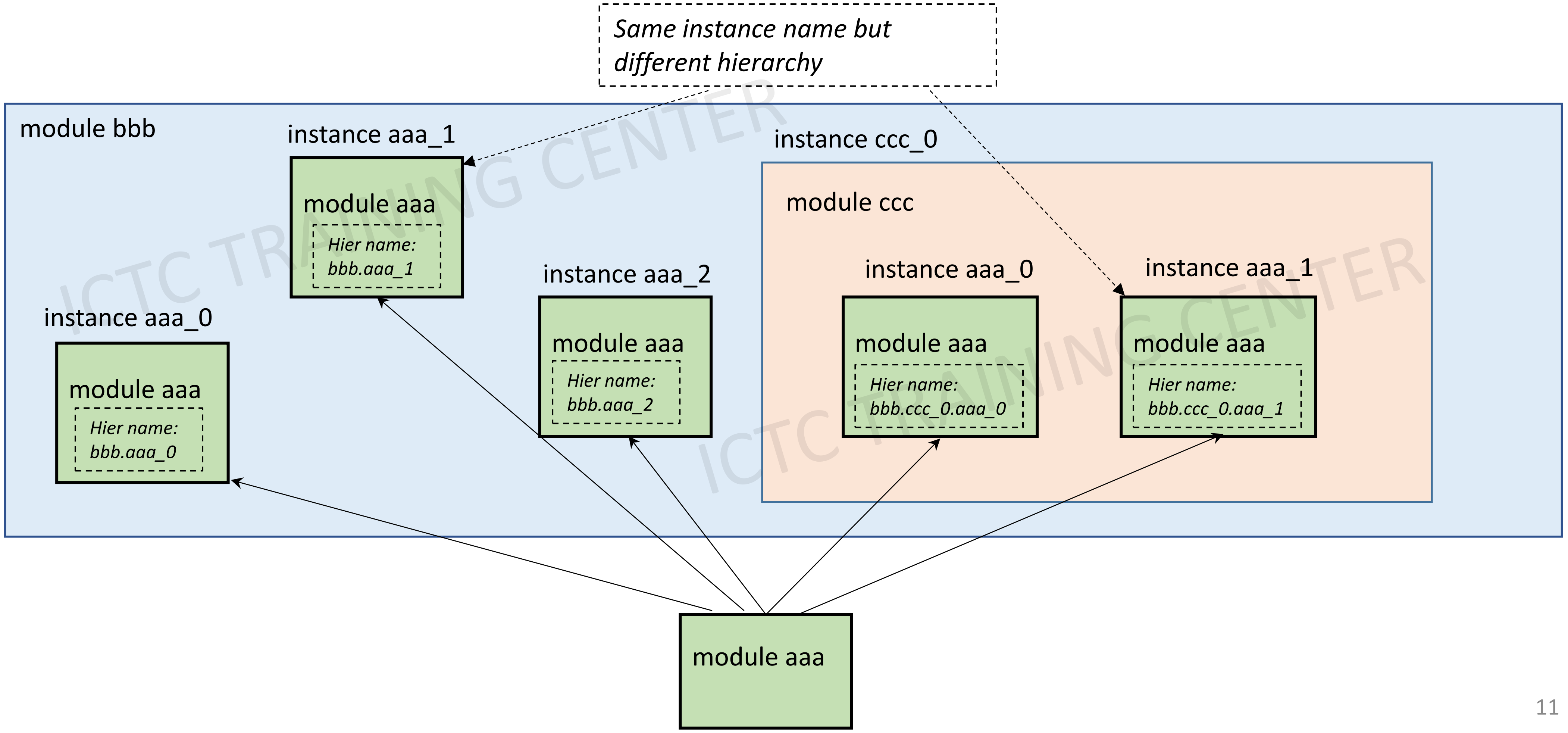
a_in	b_in	sum_out	carry_out
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



VERILOG MODULE

module hierarchy

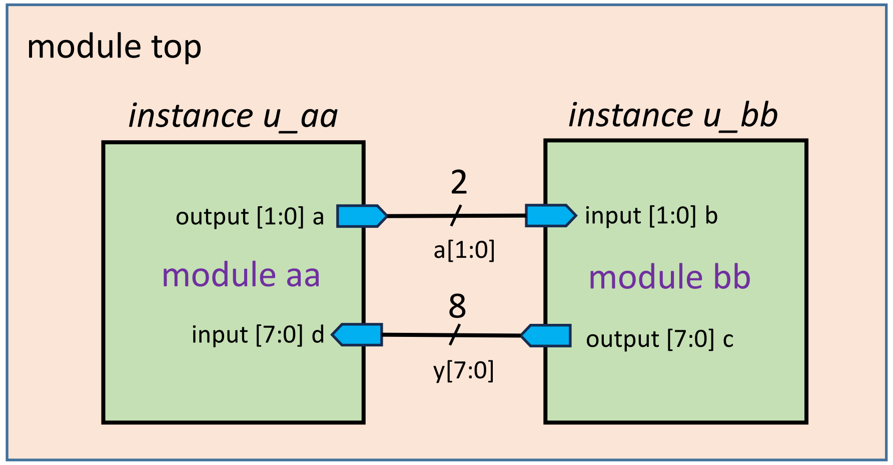
- Module can be instatiated in another module to generate hierarchical structure.
- Module can be instantiated as much as possible as long as their instance name inside the module are different.
- Hierarchy name of each instance is unique in the whole structure.



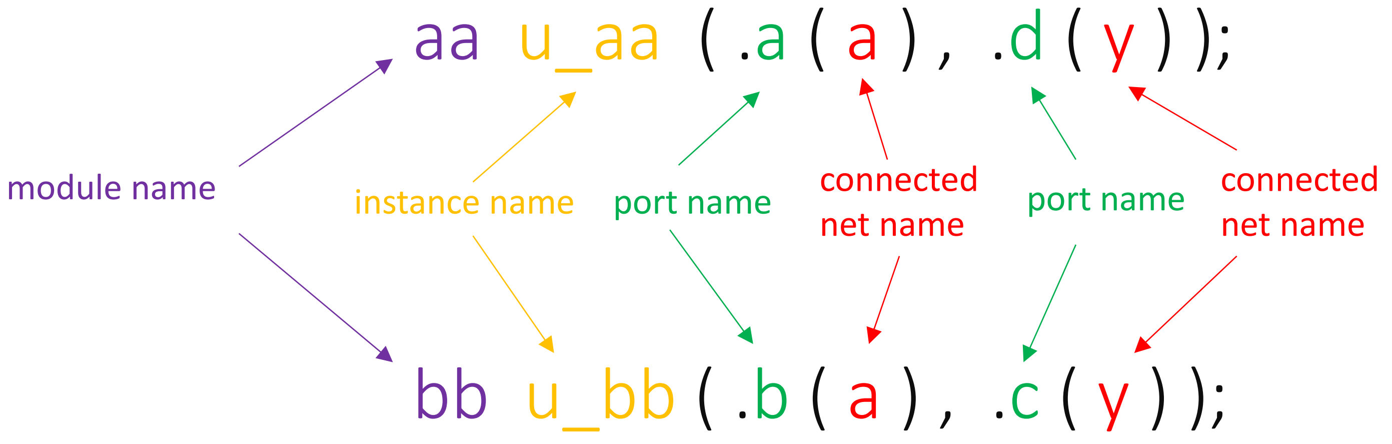
VERILOG MODULE

port connection

- When connecting directly port-to-port, input port must be connected to output port of connected module.
- The ports connected to each other can have different names.
- Need a wire (net type) to connect between 2 ports, wire can have name same as the port or can be different.



```
wire [1:0] a;  //wire connect port a & b
wire [7:0] y;  //wire connect port c & d
```



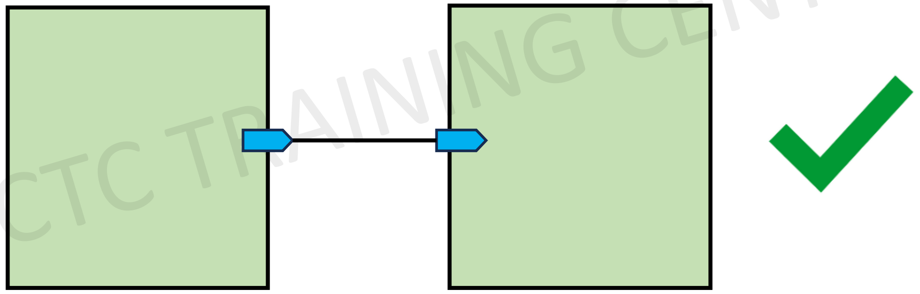
```
aa  u_aa  ( .a ( a ) ,  .d ( y ) );
bb  u_bb ( .b ( a ) ,  .c ( y ) );
```

VERILOG MODULE

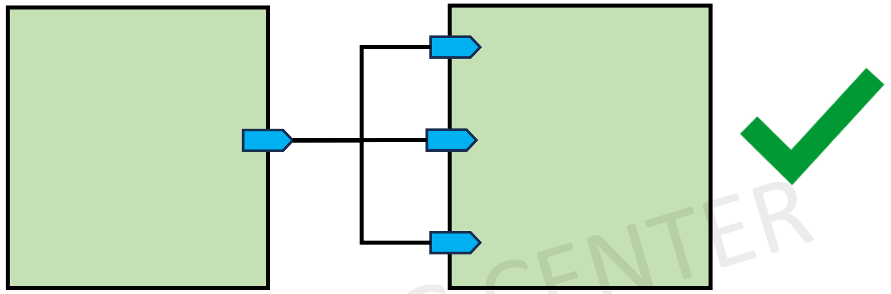
port connection

Note:

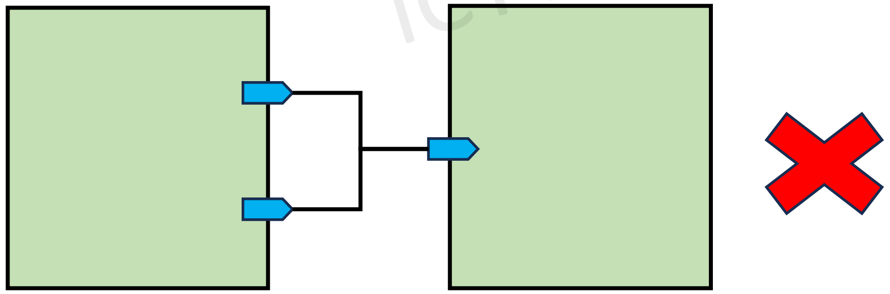
- 1 output port can be connected to more than 1 input port (fan-out).
- 1 input port can NOT be connected from more than 1 output port (multi driver).



Single output to single input



Single output to multiple inputs



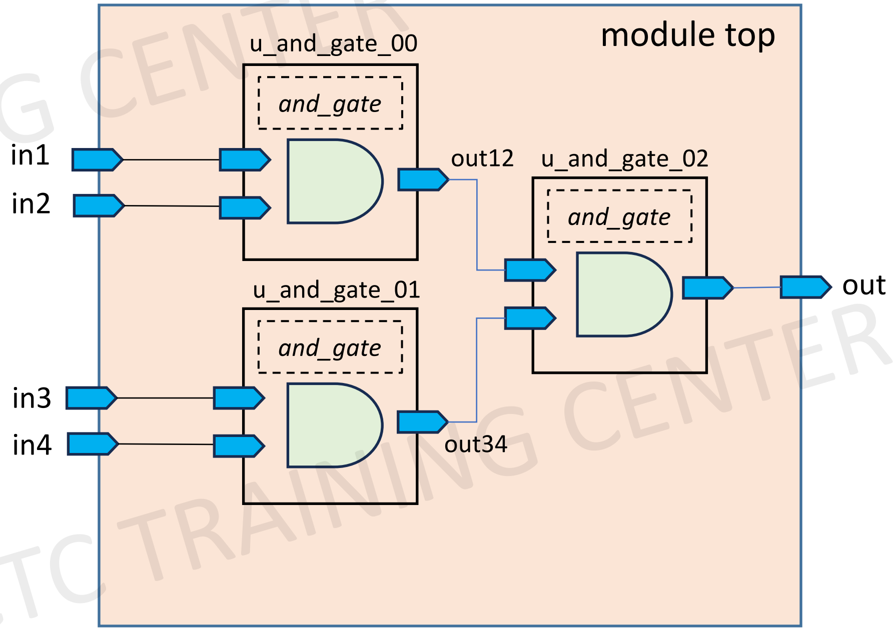
Multiple output to single input

VERILOG MODULE

practice

Practice: complete the hierarchy of and_gate below based on the and_gate module on previous example

```
module and_gate
(
   input wire a,
   input wire b,
   output wire c
);
   assign c = a & b;

endmodule
```



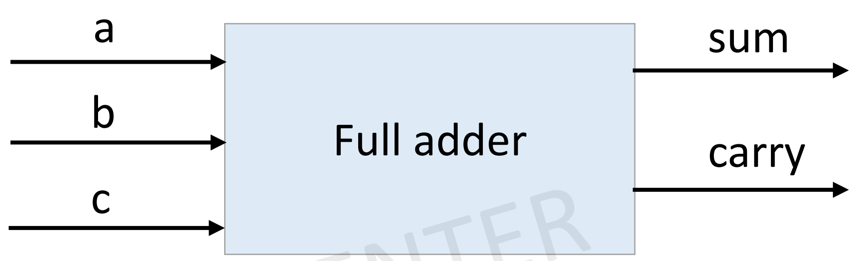
mod_name inst_name (.port_name (connected_net_name) , ...);

VERILOG MODULE

FULL ADDER from HALF ADDER

Full-Adder is used to perform the binary addition of three binary numbers.

A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



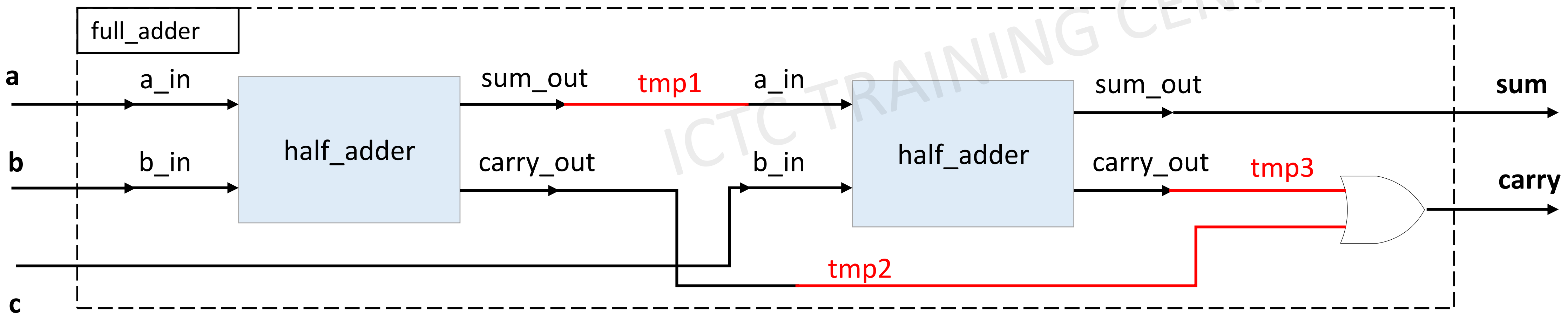
VERILOG MODULE

FULL ADDER

Practice: Design Full Adder

Full-Adder can be made from 2 half-adder as below.

1. Go to 07_ss7 folder under your home directory
2. Copy /ictc/student-data/share/teacher/07_ss7/full_adder/ to your folder
3. Create the full_adder.v under rtl folder and writing your code. The port name need to be same as the diagram.
4. Go to sim and compile, run to check the result.



SESSION 7

SUMMARY

SUMMARY:

- ❑ A module is a fundamental building block, which can be simulated and synthesized .
- ❑ Module can be instatiated in another module to generate hierarchical structure.

HOMEWORK

Homework1: Design a 2-bit full adder module to perform 2-bit binary addition based on 1 bit full-adder

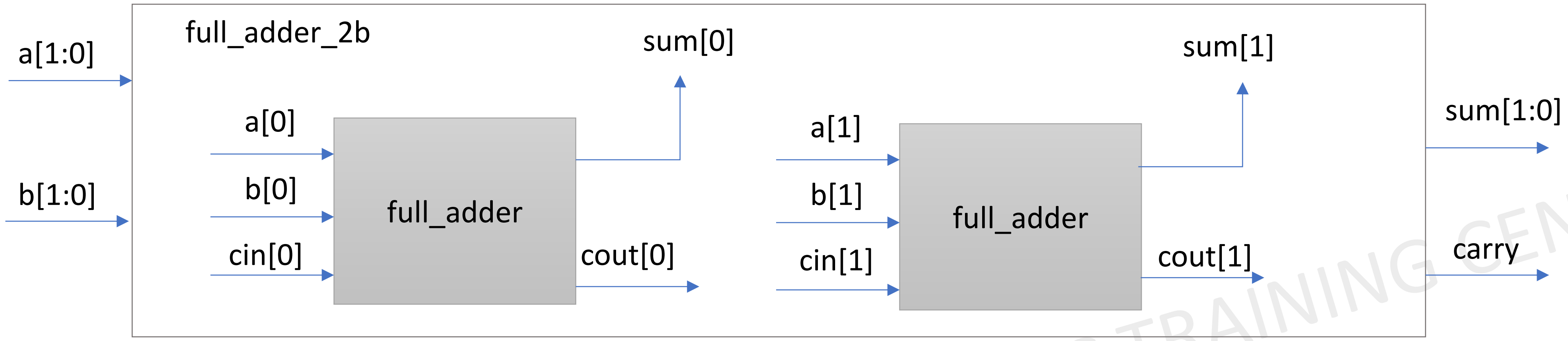
1. Go to 07_ss7 folder under your home directory
2. Copy /ictc/student-data/share/teacher/07_ss7/full_adder_2b/ to your folder
3. Create the full_adder_2b.v under rtl folder and writing your code
4. Go to sim and compile, run to check the result.

```
0  0  0  ← carry
   0  0  ← a[1:0]
   0  1  ← b[1:0]
  -------
   0  1  ← sum[1:0]
```

```
0  1  0  ← carry
   0  1  ← a[1:0]
   0  1  ← b[1:0]
  -------
   1  0  ← sum[1:0]
```

```
1  1  0  ← carry
   0  1  ← a[1:0]
   1  1  ← b[1:0]
  -------
   0  0  ← sum[1:0]
```

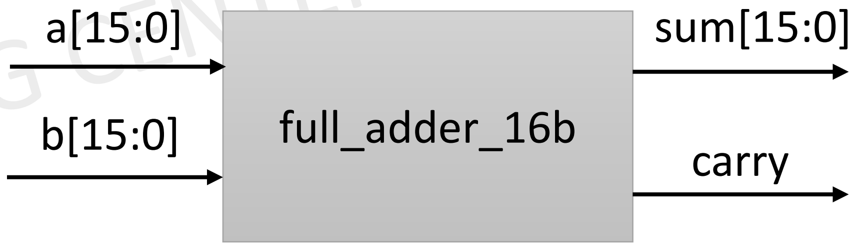
```
1  1  0  ← carry
   1  1  ← a[1:0]
   1  1  ← b[1:0]
  -------
   1  0  ← sum[1:0]
```



HOMEWORK

Homework2(*): Design 16-bit full adder module based on full-adder on previous practice

1. Go to 07_ss7 folder under your home directory
2. Copy /ictc/student-data/share/teacher/07_ss7/full_adder_16b/ to your folder
3. Create the full_adder_16b.v under rtl folder and writing your code
4. Go to sim and compile, run to check the result.



Extra homework: This homework is not counted for ranking. This is just for your practice.
Prove that full_adder can be generated from 2 half-adder (as the diagram in previous practice) by logic equivalent. This homework is j