NATIONAL UNIVERSITY OF HO CHI MINH CITY
UNIVERSITY OF INFORMATION TECHNOLOGY
FACULTY OF COMPUTER ENGINEERING

LECTURE

Subject: **VERILOG**

Hardware Description Language

Chapter3: Modules and Hierarchical structure

Lecturer: Lam Duc Khai

Agenda

Agenda

3. Chapter 3: Module and Hierarchical structure

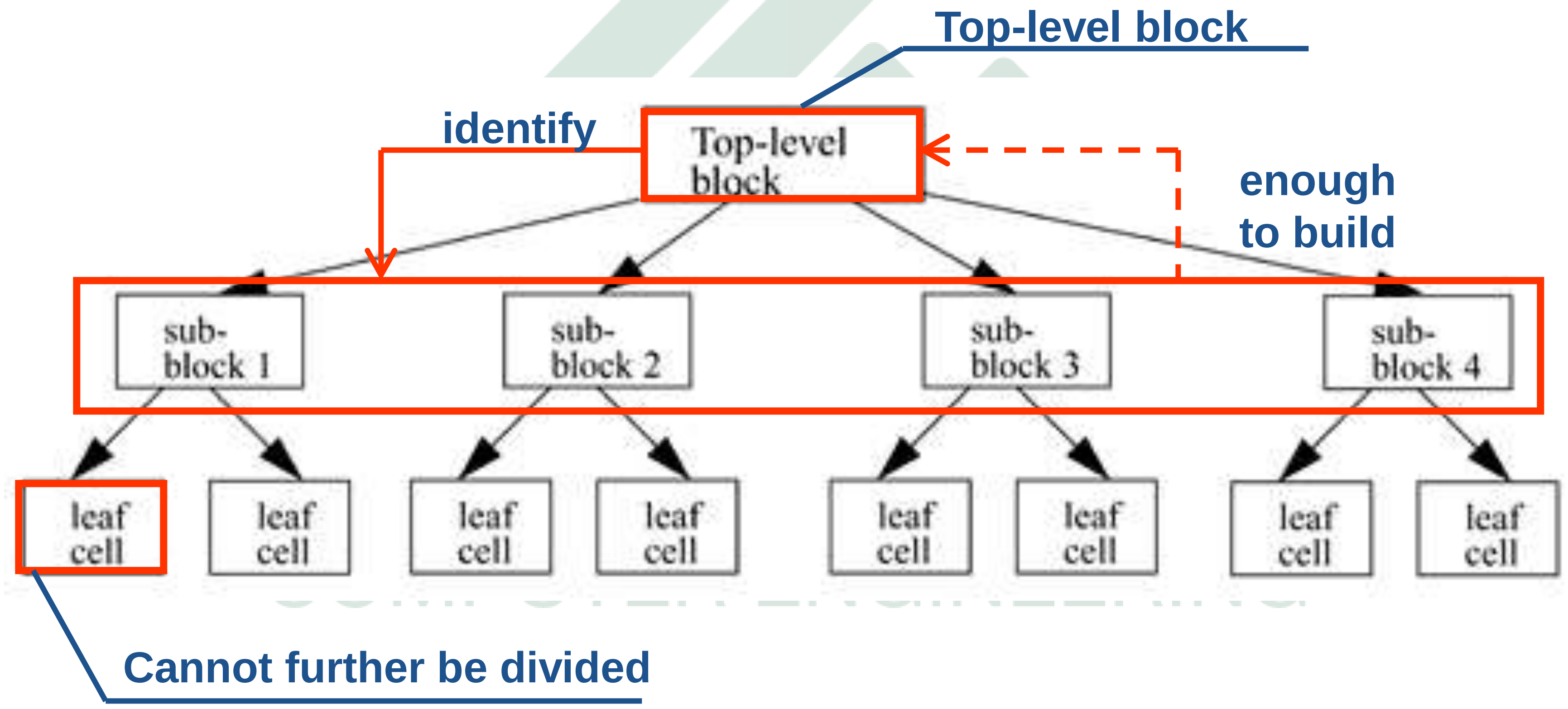
- Hierarchical structure
- Modules
- Instances

Hierarchical structure

- The Verilog HDL supports a hierarchical hardware description structure by allowing modules to be embedded within other modules. Higher level modules create instances of lower level modules and communicate with them through input, output, and bidirectional ports. These module input/output (I/O) ports can be scalar or vector.

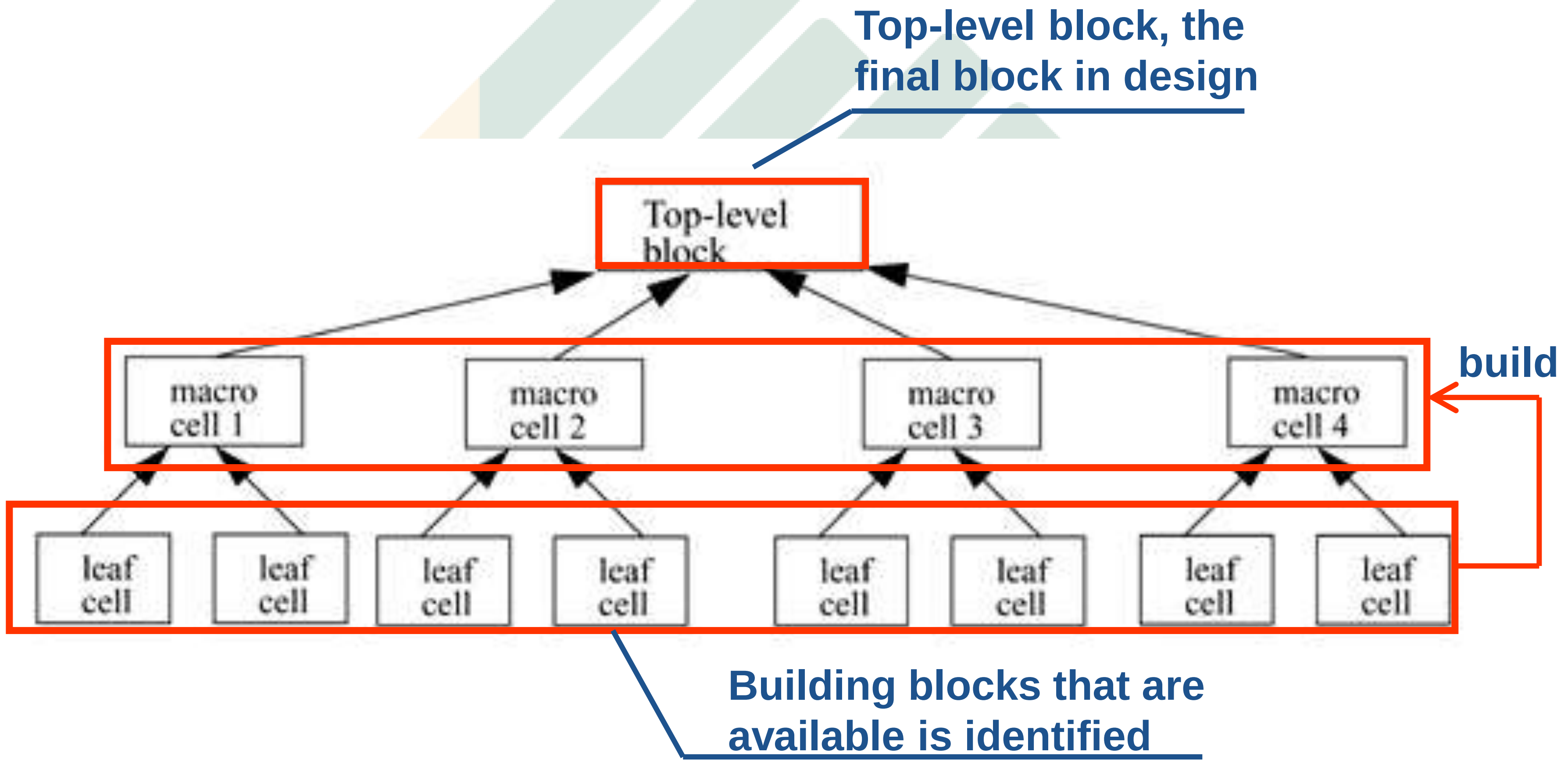
Hierarchical structure (Cont'd)

- Top-down design methodology



Hierarchical structure (Cont'd)

➢ **Bottom-up design methodology**



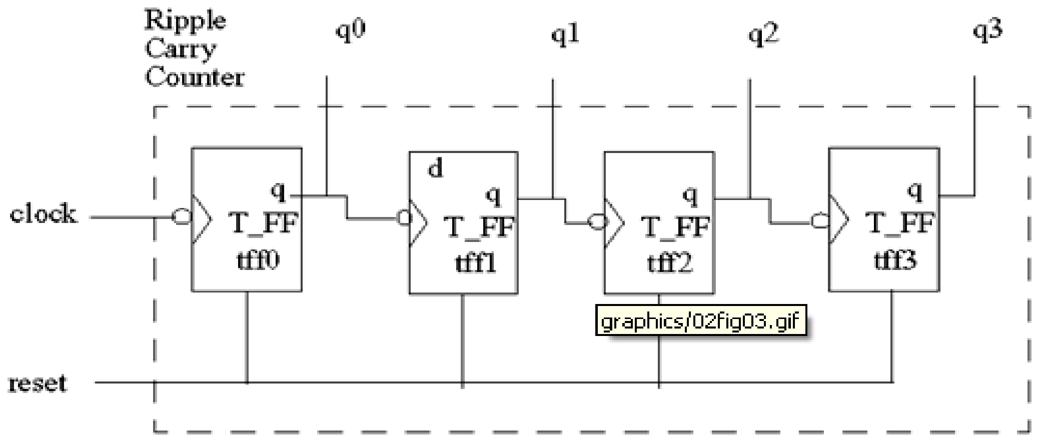
Hierarchical structure (Cont’d)

➢Design Methodologies

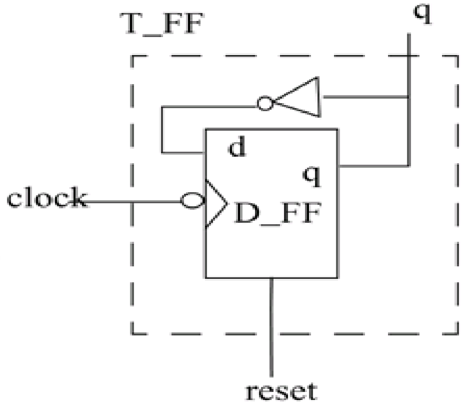
- A **combination** of top-down and bottom-up flows is typically used
  - Design architects define the specifications of the top-level block
  - Logic designers break up the functionality into blocks and sub-blocks.
  - At the same time, circuit designers are designing optimized circuits for leaf-level cells. They build higher-level cells by using these leaf cells.
  - The flow meets at an intermediate point

Hierarchical structure (Cont'd)

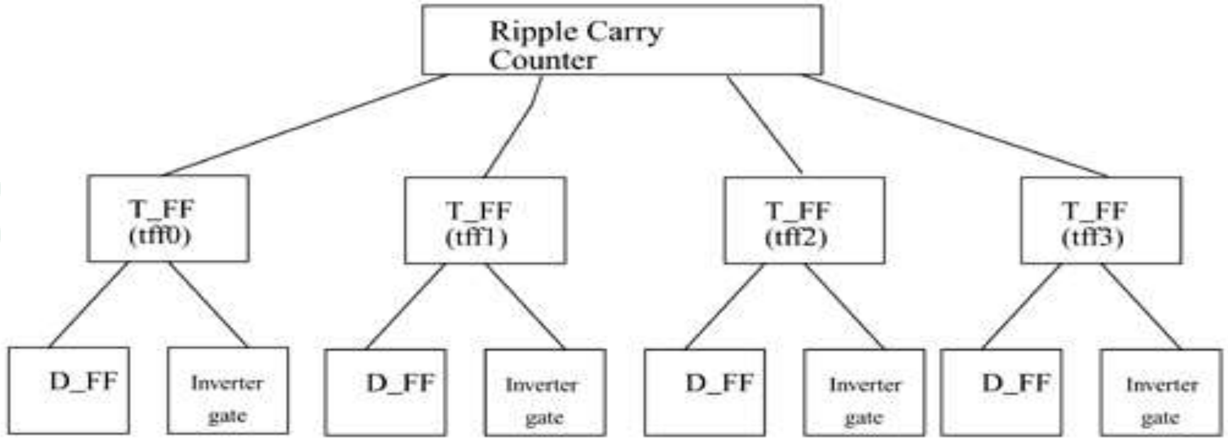
➢Example: 4-bit Ripple Carry Counter



Ripple Carry Counter



T-flipflop



Design Hierarchy

Modules

- A **module** is the basic building block in Verilog
  - Can be an element or a collection of lower-level design blocks
  - Provide functionality for higher-level block through its port interface
  - Hide internal implementation
  - Is used at many places in the design
  - Allows designers modify module internals without effecting the rest of design

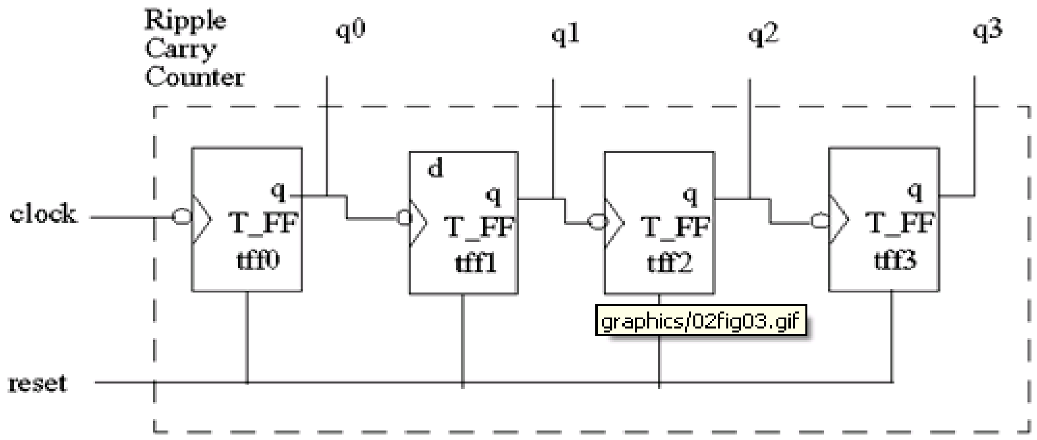
Modules (Cont’d)

➢Typical Module Components Diagram

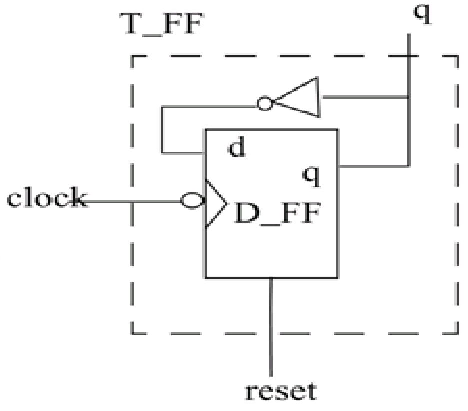
Module name, Port list (optional, if there are ports)
Port declarations Parameter list
Declaration of variables (wires, reg, integer etc.)
Instantiation of inner (lower-level) modules
Structural statements (i.e., assign and gates)
Procedural blocks (i.e., always and initial blocks)
Tasks and functions
endmodule declaration

Modules (Cont'd)

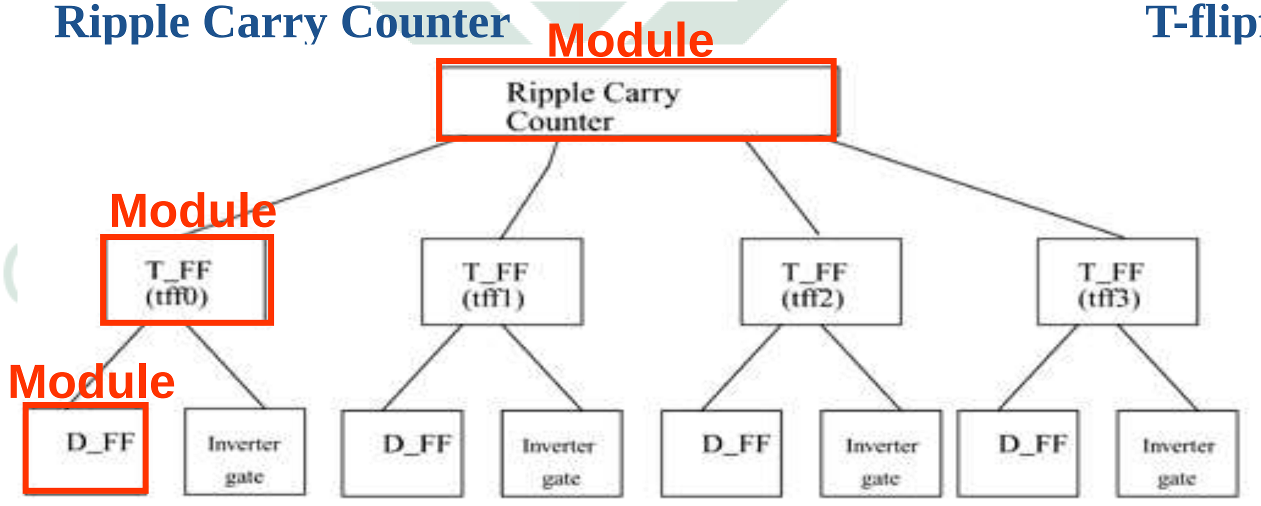
➢Example: 4-bit Ripple Carry Counter



Ripple Carry Counter



T-flipflop



Design Hierarchy

Modules (Cont'd)

➢ Module description

module *module name* **(** *port name, port name,…***);**

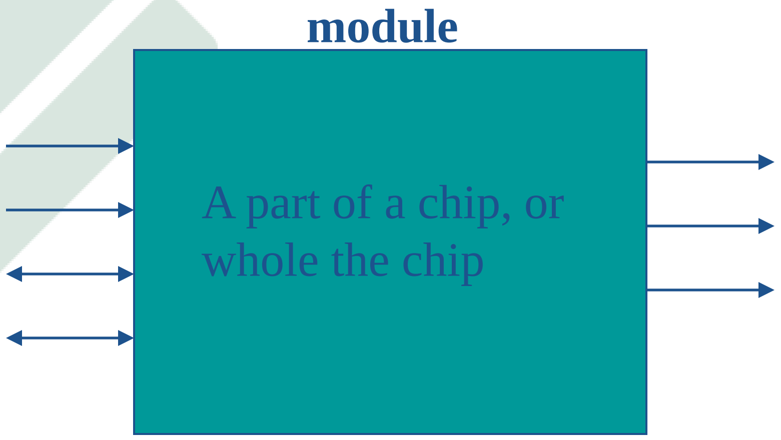
module_port declaration

data type declaration

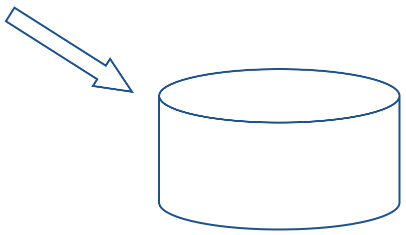
logic description part

endmodule

A module definition



The file name for RTL source must be "*module name.v*"



Modules (Cont'd)

➢ Module_port declaration

module *module name* **(** *port name, port name,…***);**

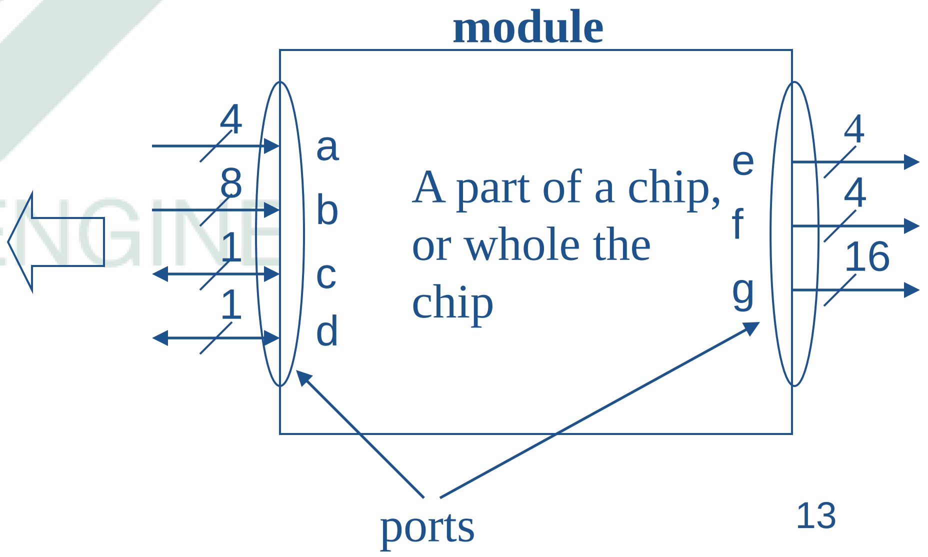
module_port declaration ⟵ Declare whether the ports are input and/or output

input <port_size> *port name, port name*, …;

output <port_size> *port name, port name*, …;

inout <port_size> *port name, port name*, …;

```
module data_conv ( a, b, …);
input [3:0] a;
input [7:0] b;
output [3:0] e, f;
output [15:0] g;
inout c, d;
```

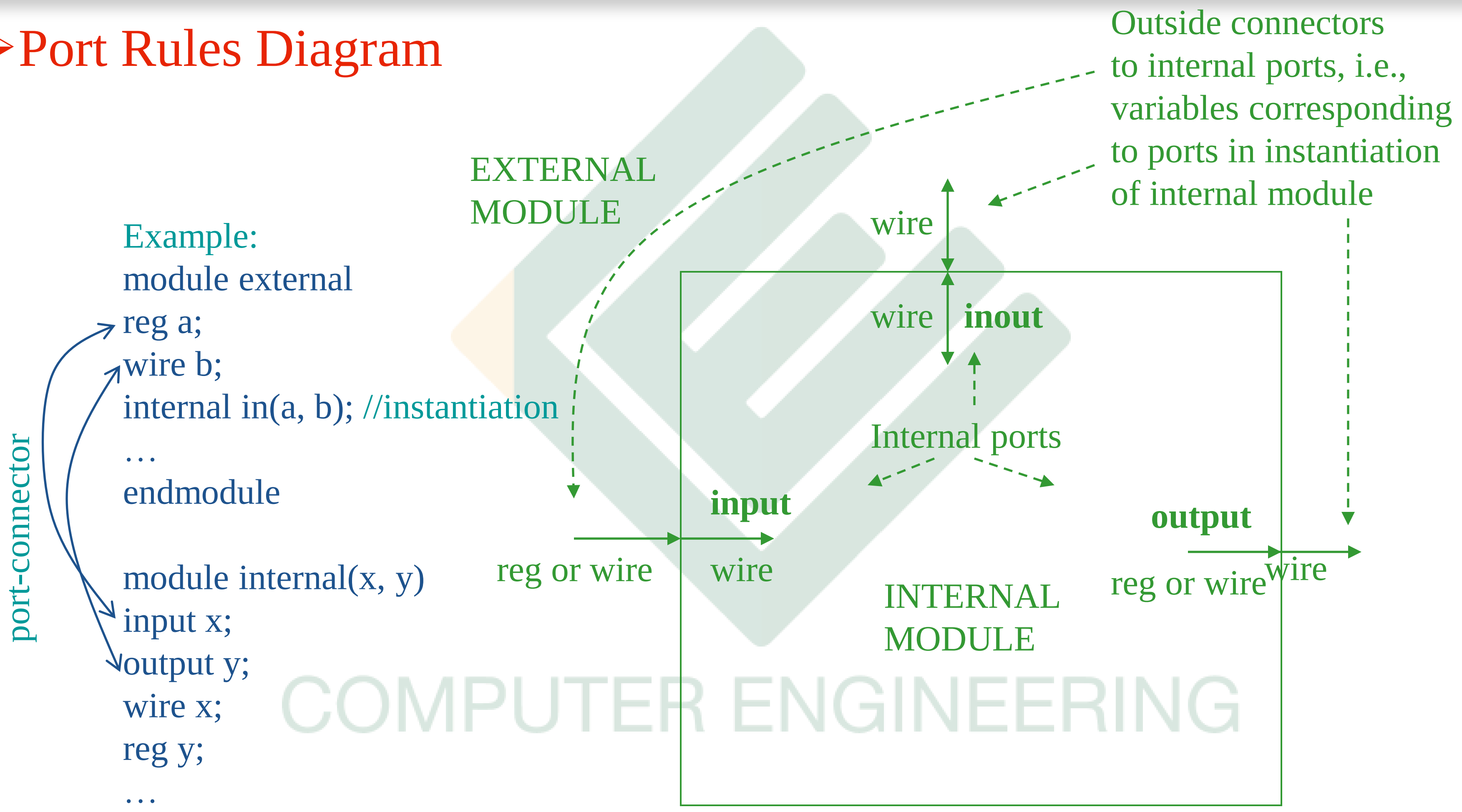


Modules (Cont'd)

➢Port Rules Diagram

Example:
module external
reg a;
wire b;
internal in(a, b); //instantiation
…
endmodule

module internal(x, y)
input x;
output y;
wire x;
reg y;
…
endmodule



port-connector
EXTERNAL MODULE
Outside connectors to internal ports, i.e., variables corresponding to ports in instantiation of internal module
wire
wire
inout
Internal ports
input
reg or wire
wire
output
reg or wire
wire
INTERNAL MODULE

General rule (with few exceptions) Ports in all modules except for the stimulus module should be wire. Stimulus module has registers to set data for internal modules and wire ports only to read data from internal modules.

Modules (Cont'd)

➢ Data type declaration

module *module name* **(** *port name, port name,…***);**

module_port declaration

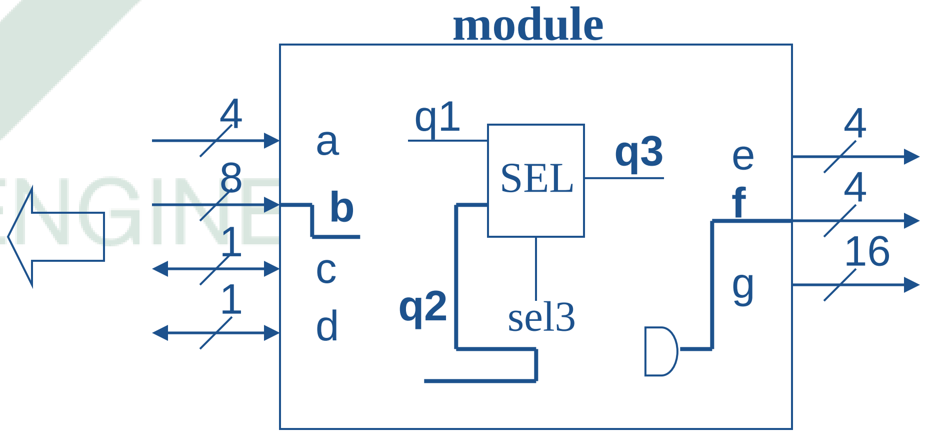
Data type declaration ← Declare characteristics of variables

for net data type

wire <size> *variable name, variable name, …;*

wire <size> *variable name, variable name, …;*

```
wire [3:0] a;
wire [7:0] b;
wire c, d;
wire [3:0] f;
wire [7:0] q1, q2, q3, q4;
wire sel3, …;
….
```



Modules (Cont’d)

➢ Data type declaration (Cont’d)

module *module name* **(** *port name, port name,…* **);**

module_port declaration

Data type declaration

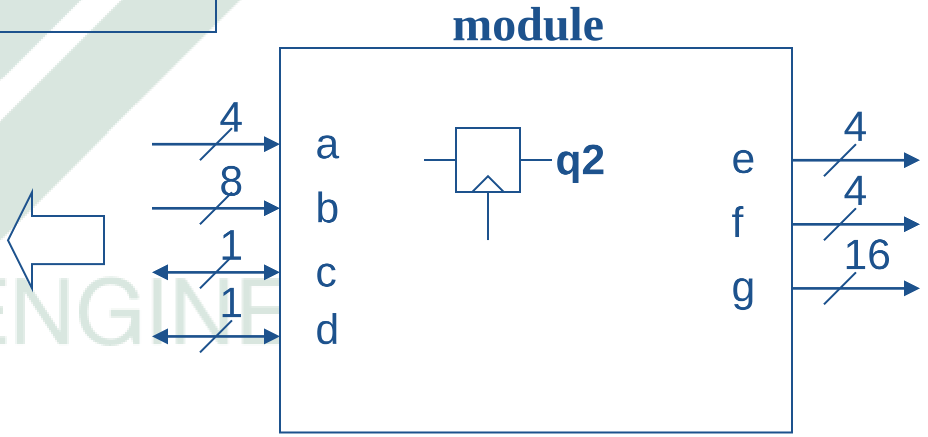
for register data type

reg <size> *variable name, variable name, …;*

reg <size> *variable name, variable name, …;*

Define signals which are output of FF, registers, and other memory elements as register type variable.

```
wire [3:0] e;
wire [15:0] g;
wire q2;
….
```



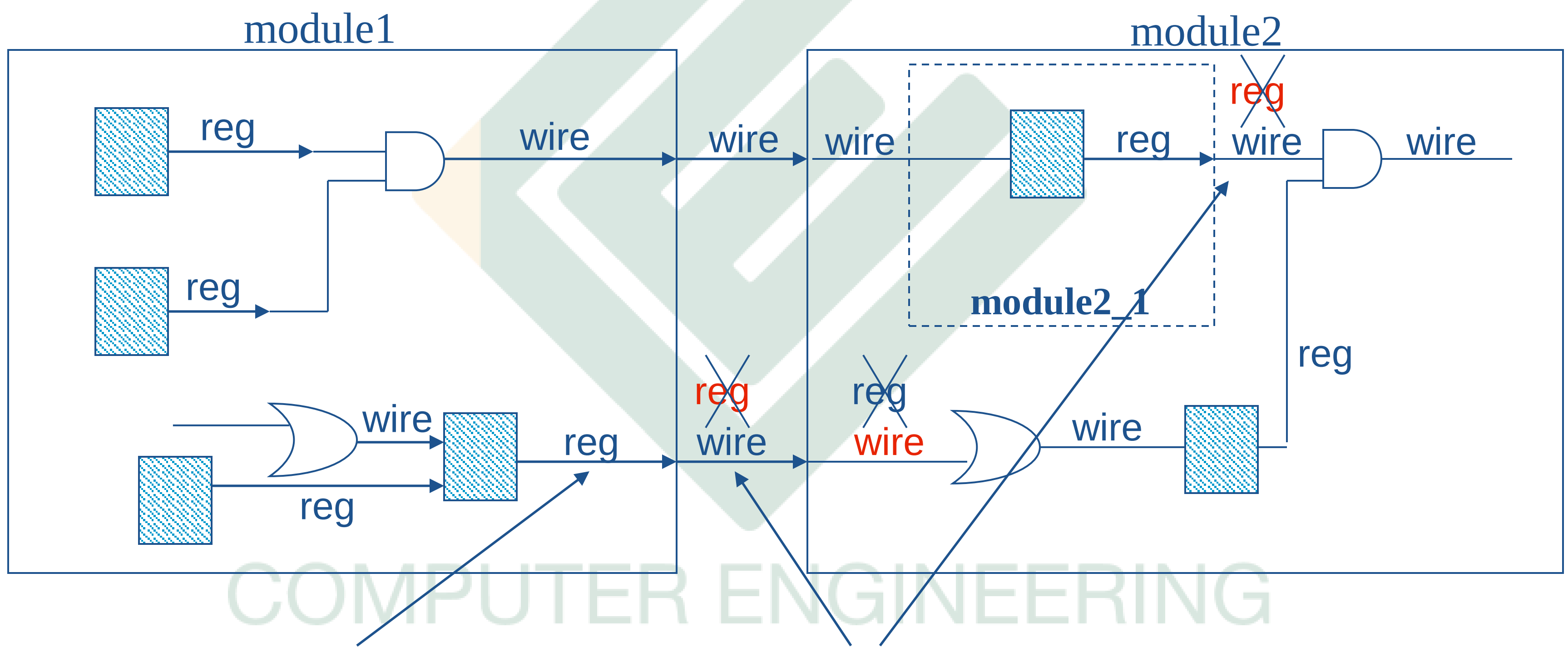
Note:

Output does not have to be declared as register data type

Input (inout) must not be declared as register data type

Modules (Cont'd)

➢Example: Mistakes and correct on register and net data type



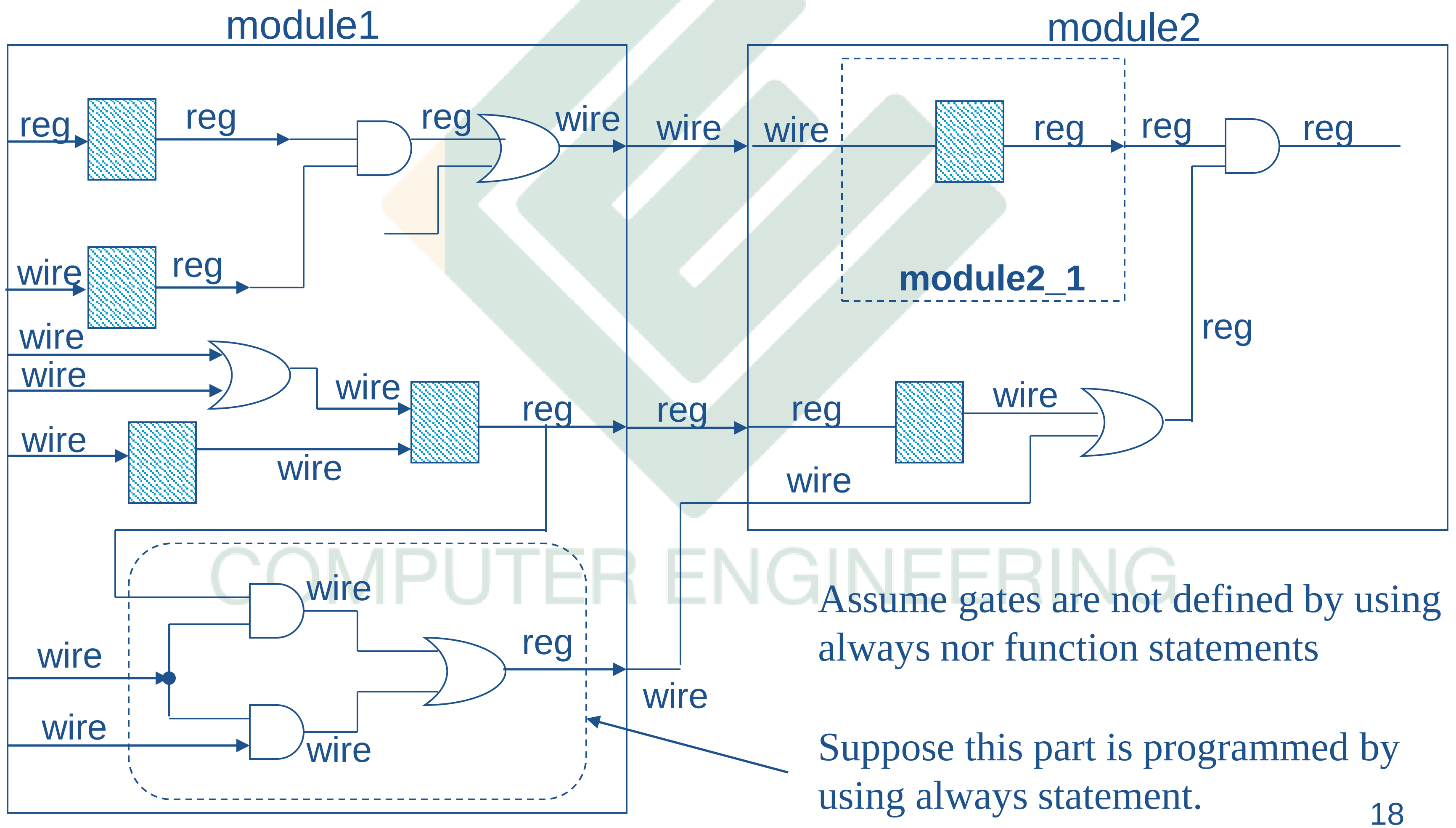
The output of memory element must be defined as *reg*

They must be defined as *wire* at these points.

▨ : memory element

Modules (Cont'd)

➢Example: Mistakes and correct on register and net data type (Cont'd)



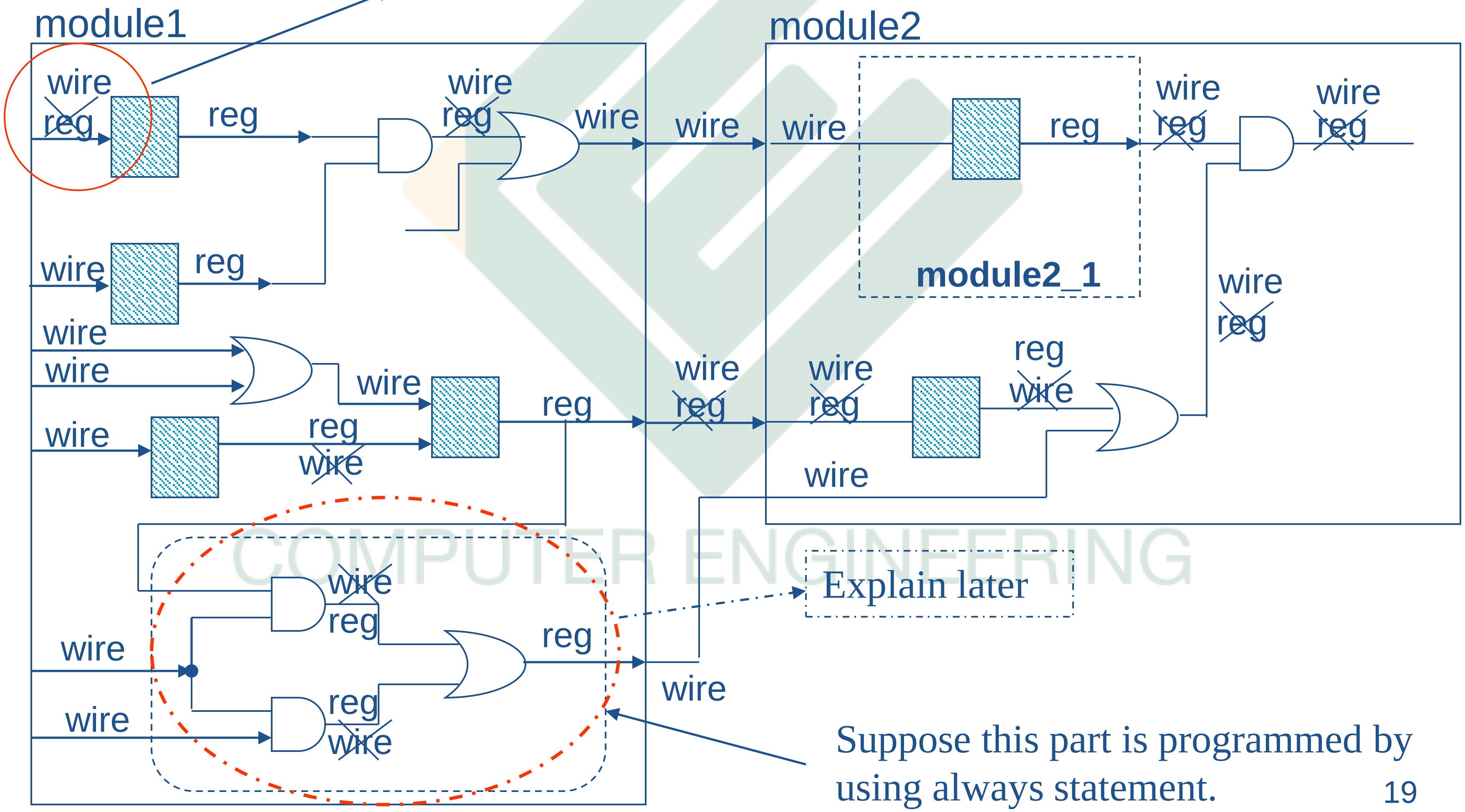
Assume gates are not defined by using always nor function statements

Suppose this part is programmed by using always statement.

Modules (Cont'd)

➢Example: Mistakes and correct on register and net data type (Cont'd)

Note: *reg* data type cannot be declared as an input !!!



Explain later

Suppose this part is programmed by using always statement.

Modules (Cont’d)

➢ Logic description part (Cont’d)

module *module name* **(** *port name, port name,…***);**

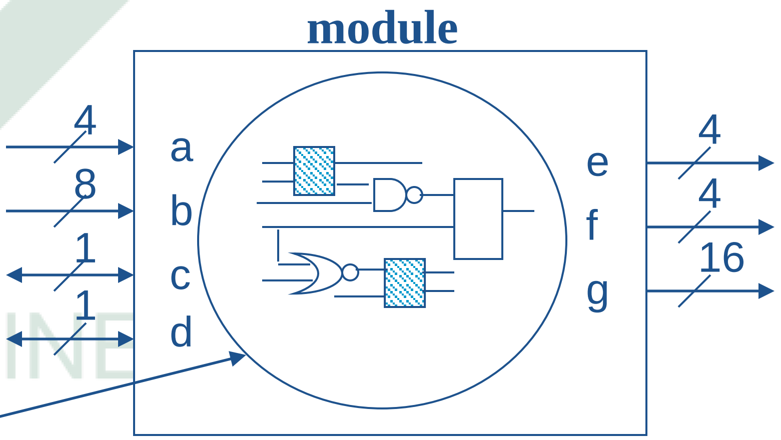
module_port declaration

Data type declaration

Logic description part ← The main part of logic is written here.

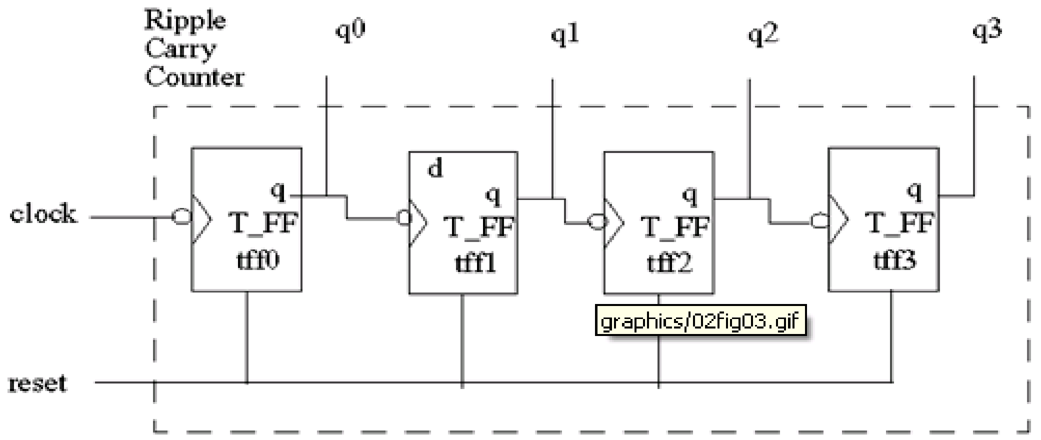
endmodule

Logic is coded in this part using various operator including connections to lower level blocks.

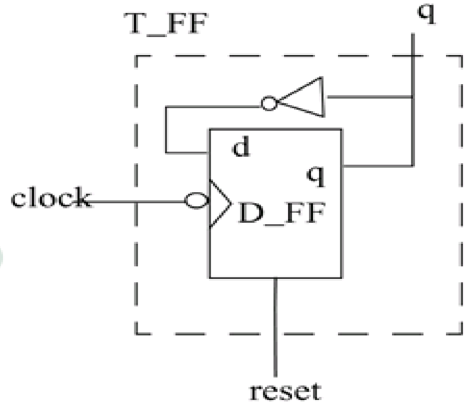


Instances

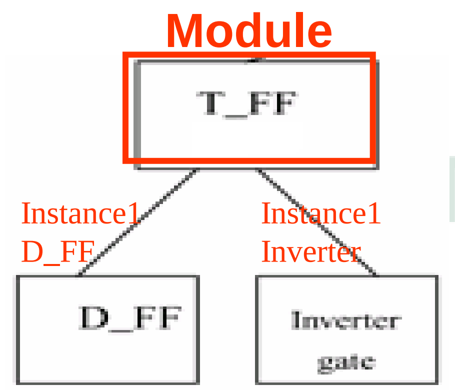
➢Example: 4-bit Ripple Carry Counter

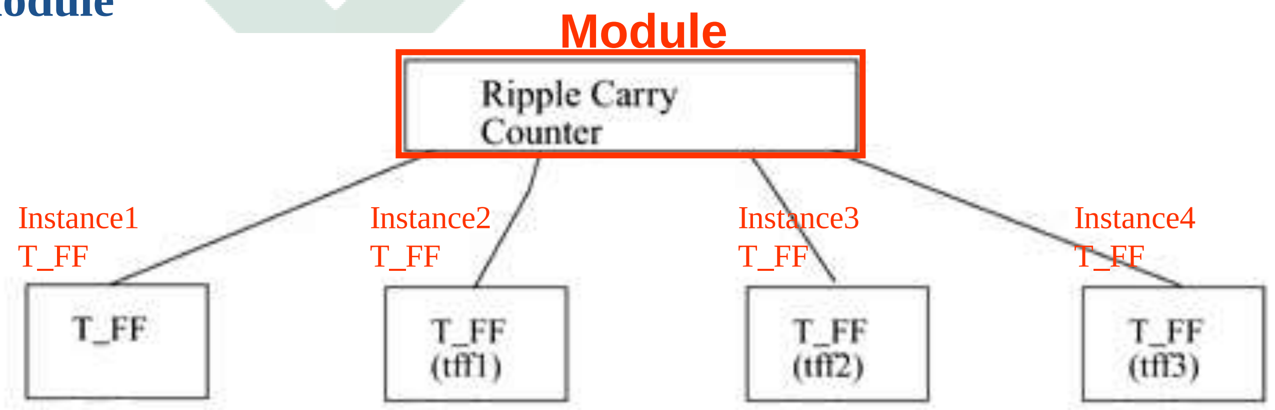


Ripple Carry Counter module



T-flipflop module





Instances (Cont’d)

➢ **Connecting module instance ports by ordered list**

The port expressions listed for the module instance shall be in the same order as the ports listed in the module declaration.

```
module topmod;
wire [4:0] v;
wire a,b,c,w;
modB b1 (v[0], v[3], w, v[4]);
endmodule

module modB (wa, wb, c, d);
inout wa, wb;
input c, d;
tranif1 g1 (wa, wb, cinvert);
not #(2, 6) n1 (cinvert, int);
and #(6, 5) g2 (int, c, d);
endmodule
```

Instances (Cont’d)

➢Connecting module instance ports by name

Connections are made by name, the order in which they appear is irrelevant.

```
module topmod;
wire [4:0] v;
wire a,b,c,w;
modB b1 (.wb(v[3]),.wa(v[0]),.d(v[4]),.c(w));
endmodule

module modB(wa, wb, c, d);
inout wa, wb;
input c, d;
tranif1 g1(wa, wb, cinvert);
not #(6, 2) n1(cinvert, int);
and #(5, 6) g2(int, c, d);
endmodule
```

END