שפת תכנון חומרה
Verilog - ורילוג

Verilog – Modules and Ports

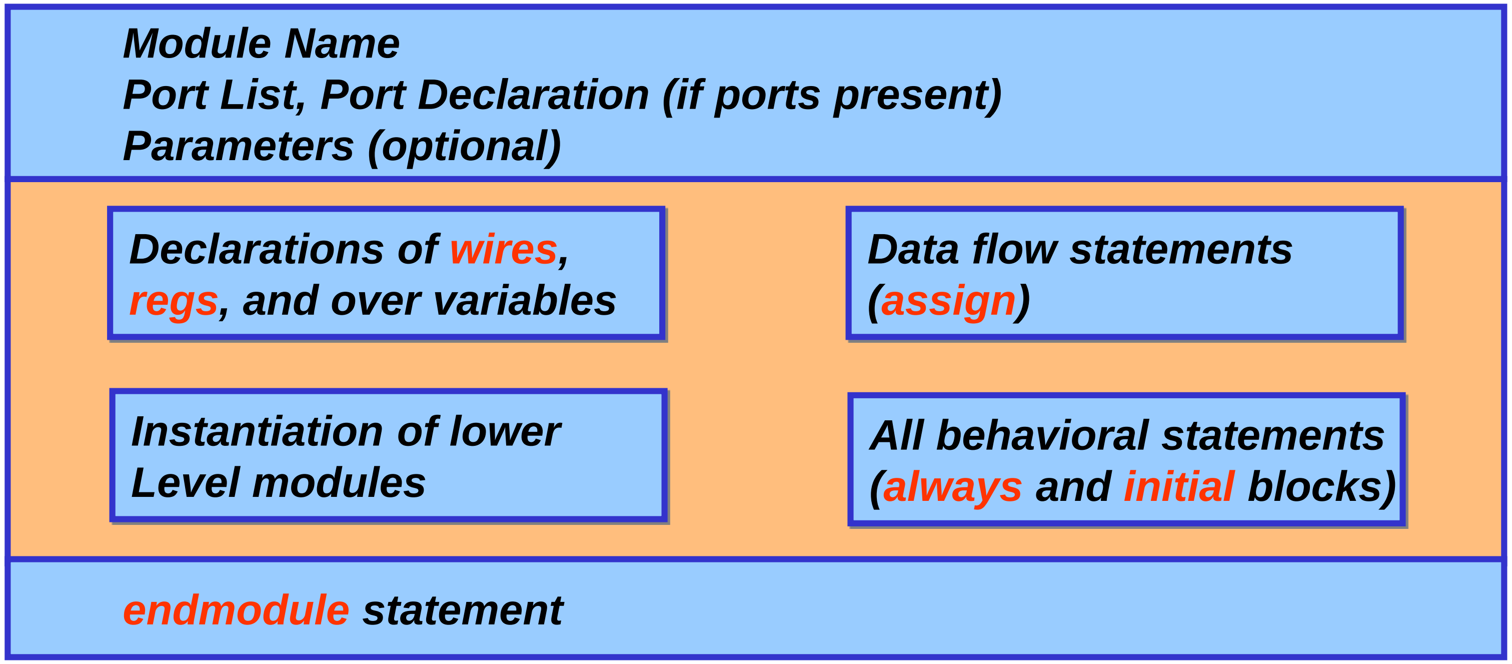
Dr. Avihai Aharon

Objectives

- Identify the components of a Verilog module definition: module names, port lists, parameters, variable declarations, dataflow statements, behavioral statements, instantiation of other modules, and tasks or functions
- Understand how to define the port list for a module and declare it in Verilog
- Describe the port connection rules in a module instantiation
- Understand how to connect ports to external signals, by ordered list, and by name
- Explain hierarchical name referencing of Verilog identifiers

Modules in Verilog: Structure

- **A module in Verilog consists of distinct parts**



Module Name
Port List, Port Declaration (if ports present)
Parameters (optional)
Declarations of wires, regs, and over variables
Data flow statements (assign)
Instantiation of lower Level modules
All behavioral statements (always and initial blocks)
endmodule statement

- **Multiple modules can be defined in any order in a single file**

Modules in Verilog: Components

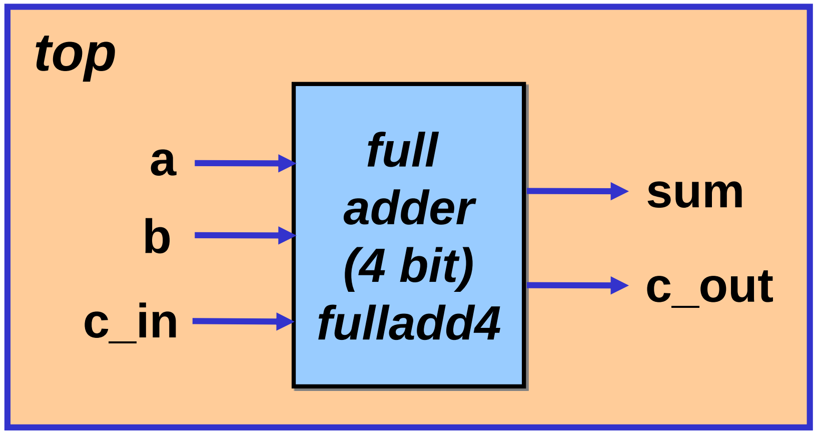
- A module definition always begin with the keyword module
- The module name, port list, port declarations, and optional parameters must come first in a module definition
- Port list and port declarations are present only if the module has any port to interact with the external environment
- The five components within a module are – variable declaration, dataflow statements, instantiation of lower modules, behavioral blocks, and tasks or functions – can be in any order and at any place of module definition (as per design needs)
- The endmodule statement must always come last in a module definition

Ports: Definition

- Ports provide the interface by which a module can communicate with its environment
- The environment can interact with the module only through its ports
  - ✓ This provides a very powerful flexibility to the designer
- The internals of the module are not visible to the environment
  - ✓ The internals of the module can be changed without affecting the environment as long as the interface is not modified
- Ports are also referred to as terminals
- A module definition contains an optional list of ports
  - ✓ If the module does not exchange any signals with the environment, there are no ports in the list

Ports: List of Ports

- A module definition contains an optional *port list*



```
module fulladd4(sum, c_out, a, b, c_in);        // Module with a list of ports
module top;                                     // No list of ports, top-level module in
                                                // simulation
```

Ports: Declaration

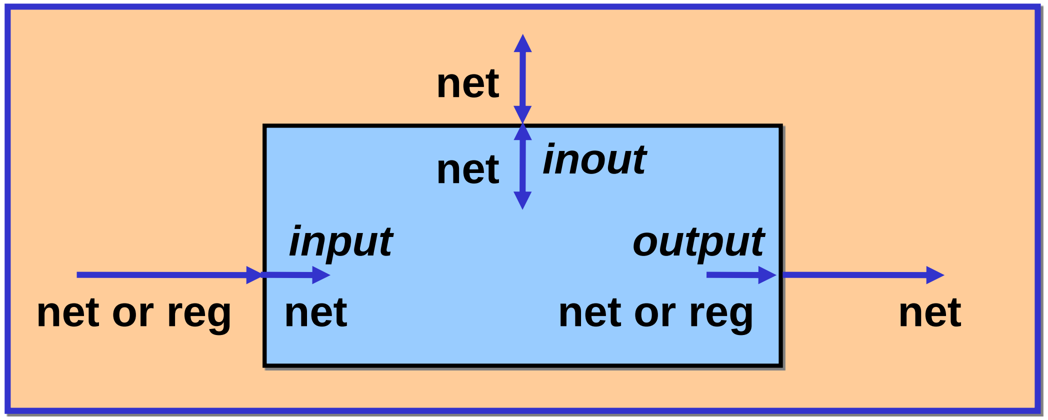
- All ports in the list of ports must be declared in the module. Ports can be declared as follows:

Verilog Keyword	*Type of Port*
input	Input port
output	Output port
inout	Bi-directional port

- **All port declarations are implicitly declared as wire in Verilog**
  - ✓ **If port intended to be a wire, it is sufficient to declare it as output, input, or inout**
  - ✓ **Input or inout ports are normally declared as wires**
  - ✓ **If output ports hold their value, they must be declared as reg**

Ports: Connection Rules

- One can visualize a port as consisting of two units, one unit is internal to the module another is external to the module
  - ✓ The internal and external units are connected
  - ✓ There are rules governing port connections when modules are instantiated within over modules (summarized in Figure)
  - ✓ The Verilog simulator complains if any port connection rules are violated



Ports: Connection Rules (cont.)

- Inputs
  - ✓ Internally, input ports must always be of the type net
  - ✓ Externally, the inputs can be connected to reg or wire variable
- Outputs
  - ✓ Internally, output ports can be of the type reg or net
  - ✓ Externally, output ports must always be connected to a net
- Inouts
  - ✓ Internally, inout ports must always be of the type net
  - ✓ Externally, inout ports must always be connected to a net
- It is legal to connect internal and external items of different sizes when making inter-module port connection (not recommended)
- Verilog allows ports to remain unconnected (not recommended)

Ports: Connecting to External Signals

- There are two methods of making connections between signals specified in the module instantiation and the ports in a module definition
- Connecting by ordered list
  - ✓ The signals to be connected must appear in the module in the same order as the ports in the port list in the module definition
- Connecting by name (recommended)
  - ✓ Verilog provides the capability to connect external signals to ports by the port names, rather than by position
  - ✓ The port connections can be specified in any order as long as the port name in the module definition correctly matches the external signal
  - ✓ Unconnected ports can be dropped in port connection by name
- The two methods may be mixed!!!

Ports: Connecting to External Signals (by order)

- Connecting by order

```
module fulladd4(sum, c_out, a, b, c_in);

// Begin port declaration section
output [3:0] sum;
output c_out;
input [3:0] a, b;
input c_in;
// End port declaration section
…
<module internals>
…
endmodule
```

```
module top;

// Declare connection variables
wire [3:0] SUM;
wire C_OUT;
reg [3:0] A, B;
reg C_IN;
// Instantiate fulladd4, call it fa_ordered
// Signals are connected to ports
// by order
fulladd4 fa_ordered (SUM, C_OUT, A,
                     B, C_IN);
…
<stimulus>
…
endmodule
```

Ports: Connecting to External Signals (by name)

- Connecting by order & name (recommended)

```
module fulladd4(sum, c_out, a, b, c_in);

// Begin port declaration section
output [3:0] sum;
output c_out;
input [3:0] a, b;
input c_in;
// End port declaration section
…
<module internals>
…
endmodule
```

```
module top;
// Declare connection variables
wire [3:0] SUM;
wire C_OUT;
reg [3:0] A, B;
reg C_IN;
// Instantiate fulladd4,call it
    fa_byname
// Signals are connected to ports
// by name
fulladd4 fa_byname (.sum(SUM),
                    .c_out(C_OUT),
                         .a(A), .b(B),
                         .c_in(C_IN));
…
<stimulus>
…
endmodule
```

Ports: Hierarchical Names

- Verilog supports a hierarchical design methodology
- Every module instance, signal or variable is defined with an identifier
- A particular identifier has a unique place in the design hierarchy
- Hierarchical name referencing allow us to denote every identifier in the design hierarchy with a unique name. Any identifier can be addressed from any place in the design by simply specifying the complete hierarchical name of that identifier
- A hierarchical name is a list of identifiers separated by dots (".") for each level of hierarchy

```
top.fa_byname.c_in          // Full hierarchical name: port c_in in module fulladd4
top.C_IN;                              // Full hierarchical name: port C_IN in
    module top
```

Modules & Ports Summary

- Module definitions contain various components. Keywords module and endmodule are mandatory. Other components – port list, port declarations, variable and signal declarations, dataflow statements, behavioral blocks, lower-level module instantiations, and task or functions – are optional and can be added as needed

- Ports provide the module with a means to communicate with other modules or its environment. A module can have a port list. Ports in the port list must be declared as input, output, or inout. When instantiating a module, port connection rules are enforced by the Verilog simulator

- Ports can be connected by name or by ordered list

- Each identifier in the design has a unique hierarchical name. Hierarchical names allow us to address any identifier in the design from any over level of hierarchy in the design