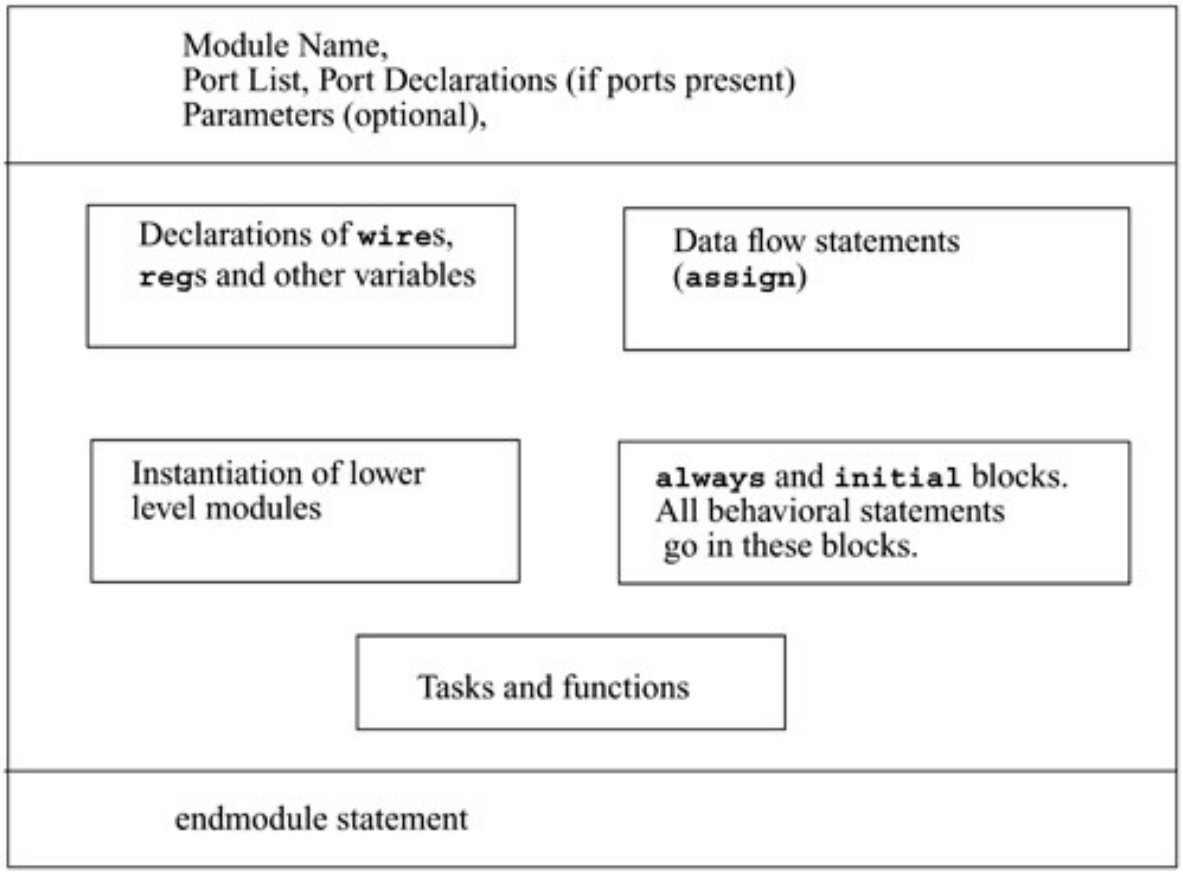
Concept of a ‘module’

- A module is a basic building block in Verilog and can specify a system of any complexity.
- The module definition is the same for a system of any complexity.
- Provides functionality through its port interface.
- It can be an element or a collection of lower-level design (macro or leaf cells or primitive cells) blocks.

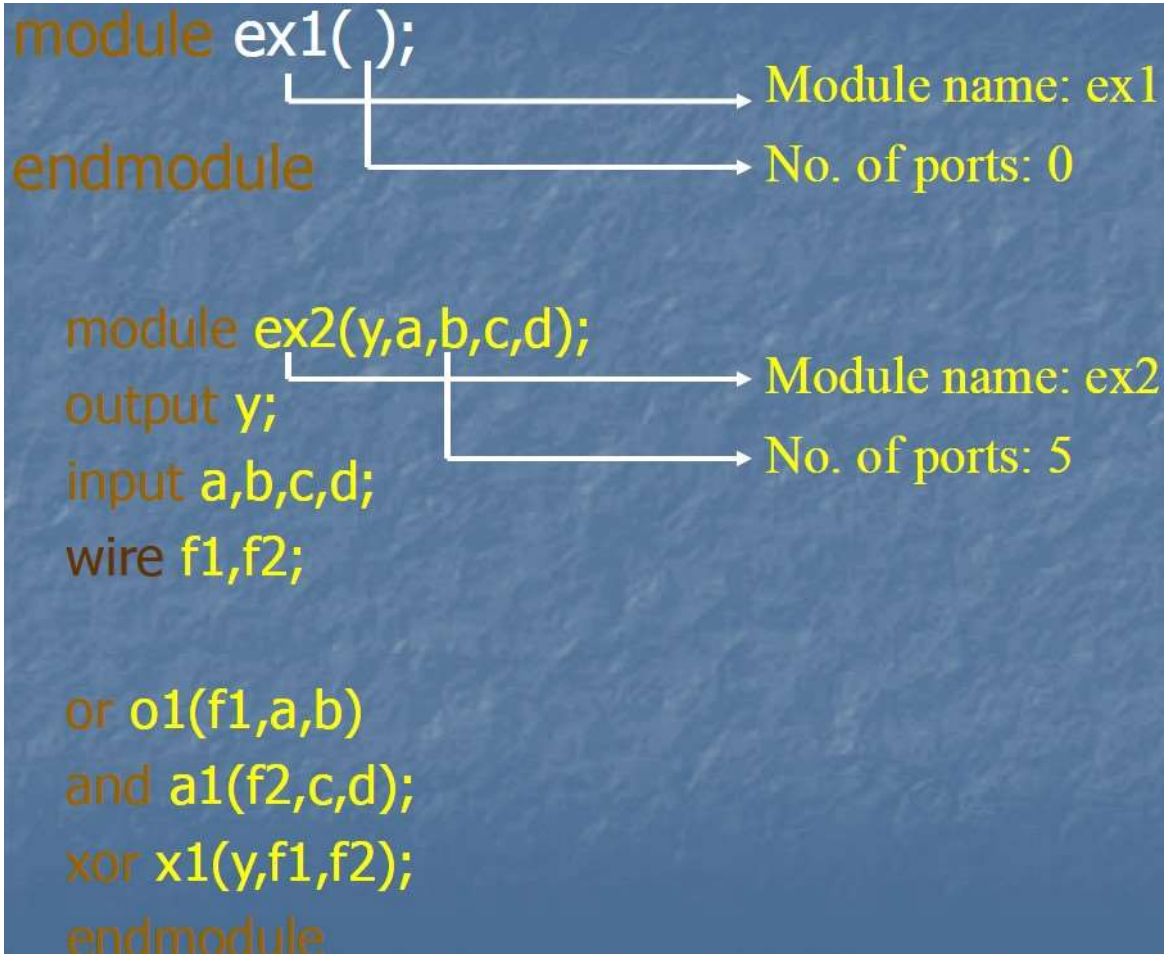
Components of a Verilog Module



How to declare a module ?

- A module in Verilog is declared using the keyword module and a corresponding keyword endmodule must appear at the end of the module.
- Each module must have a module name, which acts as an identifier.
- A module can have an optional port list which describes the input, output & inout terminals of the module.

Module declaration - Examples



A module communicates with the other modules or the external environment through its ports. So the functionality of any module is reflected through its ports by hiding the internals of the module, which describes the functionality for that module.

Nesting of modules

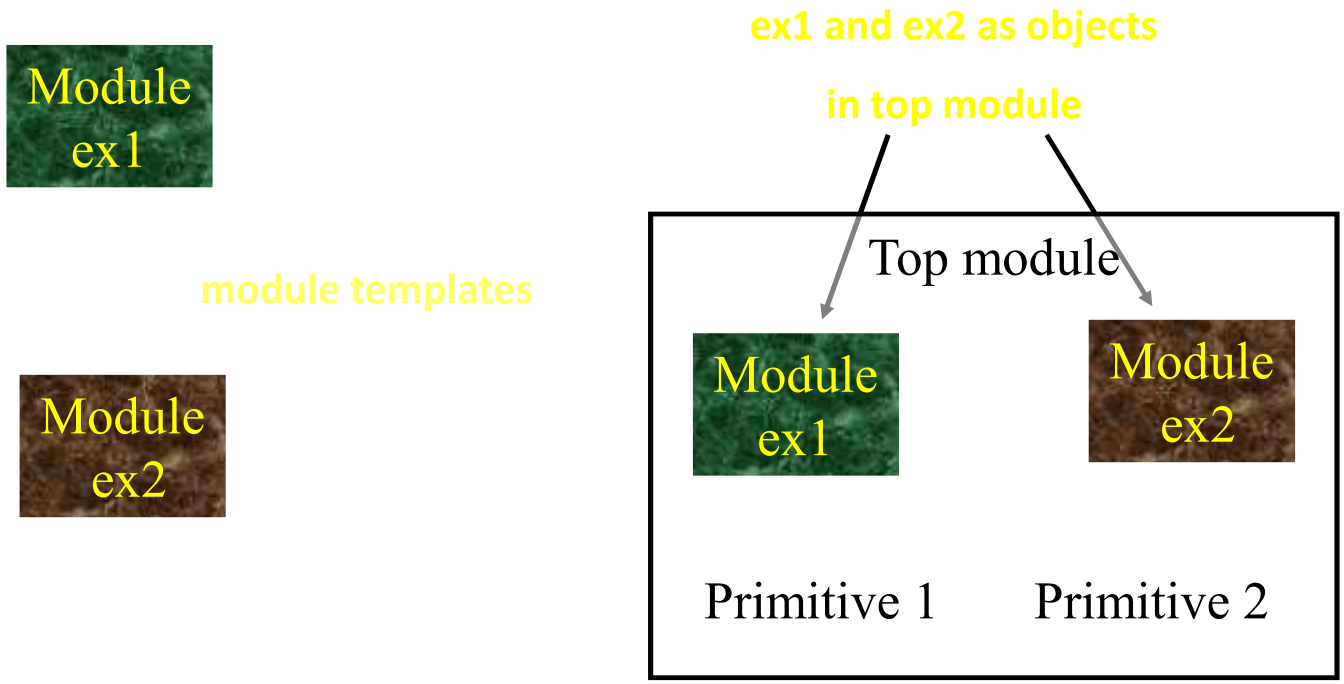
- In Verilog nesting of modules is not permitted i.e., one module definition cannot contain another module definition within the module and endmodule statements.

```
module counter(q, clk, reset);
output [3:0]q;
input clk, reset;

    module T_FF(q, clock, reset)   // Illegal

        endmodule
endmodule
```

Module instances

- A module provides a template from which one can create actual objects.
- Each object has its own name, variables, parameters and I/O interface.
- The process of creating objects from a module template is called instantiation, and the objects are called instances.
- Primitives and modules can be instantiated.

Module instantiation



Ports

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	bidirectional

Port Declarations

```
module fulladd4(sum, c_out, a, b, c_in);
//Begin port declarations section
output [3:0] sum;
output c_cout;
 input [3:0] a, b;
 input c_in; //End port declarations section ...
<module internals> ...
endmodule
```

Data types

- Net Data type
- Registers Data type

Data Type

A variable belongs to one of two data types

Net:

- Must be continuously driven
- Used to model connection between continuous assignments and Instantiations.

Register:

- Retains the last value assigned to it.
- Often used to represent storage element.

Net Data Type

- Nets represent connections / physical wires between hardware elements.
- Nets will not store / hold any value.
- Different net types supported for synthesis:
  - wire , wor, wand, tri , supply0, supply1
  - wire and tri are equivalent ; when there are multiple drivers, driving them, the output of the drivers are shorted together.
  - wor / wand inserts an OR/AND gate at the connection.
  - supply0 / supply1 model power supply connections.
- Default Size : 1-bit / scalar
- Default Value : z

Wire declaration examples

- wire a; // signal ‘a’ declared as wire
- wire out; // signal ‘out’ declared as wire

  Ex: assign out = a | b; or o1(out, a, b);

- wire a, b; // signals ‘a’ & ‘b’ declared as wires
- wire d = 1’b0; /*net ‘b’ is fixed to logic value ‘0’ at declaration*/

Registers

- ✓In Verilog registers represent data storage elements.
- ✓Used to model hardware memory elements / registers.
- ✓Registers can hold / store a value.
- ✓Declared by the keyword **reg , integer**
- ✓Default Size : 1-bit / scalar
- ✓Default Value : x

Vectors

- Nets or register data types can be declared as vectors (more no. of bits).
- If bit width is not specified then the default value is 1-bit (scalar).

```
wire a;      //  default  scalar net value
wire [7:0] bus;      // 8-bit bus
wire [31:0] busA, busB, busC;   //32- bit bus
reg clock; // scalar register(default)
reg [0:40] virtual_addr;  //virtual address 41 bits
```

Addressing Vectors

- **wire** [15:0]busA;

busA[9]; // bit # 9 or 10th bit of vector busA from LSB

- wire [0:15]busB;

busB[9]; // bit # 9 or 7th bit of vector busB from LSB

- reg [31:0]cnt_out;

cnt_out[14:7]; // group of 8 bits of a vector register

cnt_out[7:14]; // is illegal addressing

Reference

1. Samir Palnitkar,”Verilog HDL: A Guide to Digital Design and Synthesis” Prentice Hall, Second Edition,2003