Port Connection Rules

- Named association
- Positional association

Coding Styles

- A module cannot be declared within another module
- A module can instantiate other modules
- A module instantiation must have a module identifier (instance name) except built in primitives,gate and switch primitives and user defined primitives

Named association at the top level modules to avoid confusion

Structural Modelling

- Structural Modelling at gate level

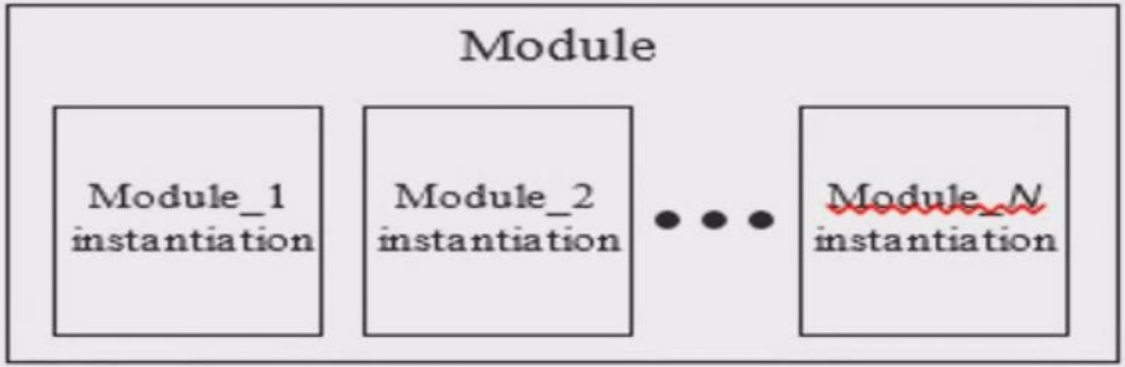
The Half adder instantiates two gate primitives

The Full adder instantiates two half modules and one gate primitive

The Four bit adder is constructed by four full adders instances

You

Verilog program structure



Each module declaration includes a list of interface signals that can be used to connect to other modules or to the outside world.

You

Module Declaration

module module-name (module interface list);
[list-of-interface-ports]
...
[port-declarations]
...
[functional-specification-of-module]
...
endmodule

The items enclosed in square brackets are optional. The list-of-interface- ports normally has the following form:

type-of-port list-of-interface-signals

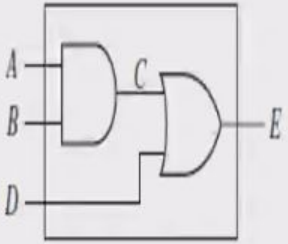
{; type-of-port list-of-interface-signals};

- The curly brackets indicate zero or more repetitions of the enclosed clause. Type-of-port indicates the direction of information; whether information is flowing into the port or out of it.
- Input port signals are of keyword **input**, output port signals are of keyword **output**, and bidirectional signals are of keyword **ino** list-of- ports can be combined with the module interface list.

You

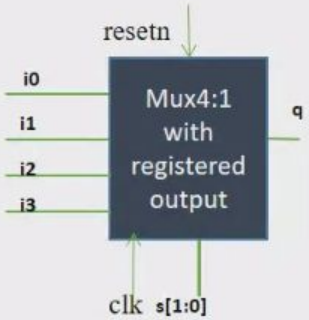
Verilog Modules

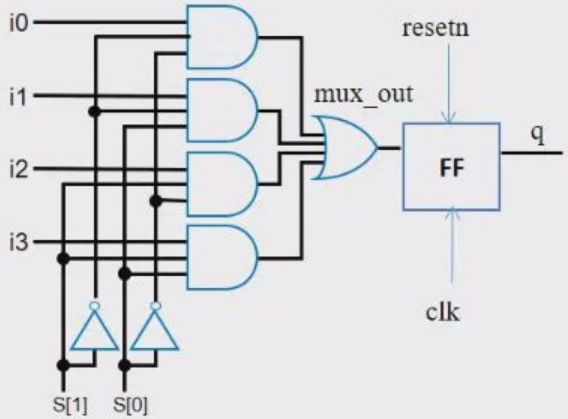
- The general structure of a Verilog code is a module description.
- A module is a basic building block that declares the input and output signals and specifies the internal operation of the module
- The **module** declara- tion has the name two_gates and specifies the inputs and outputs. *A*, *B*, and *D* are input signals, and *E* is an output signal.
- The signal *C* is declared within the module as a **wire** since it is an internal signal.
- The two concurrent statements that describe the gates are placed and the module ends with **endmodule**. All the input and output signals are listed in the module statement without specifying whether they are input or output.




```
module two_gates (A, B, D, E);
output E;

input A, B, D;

wire C;

  assign C = A && B; // concurrent
  assign E = C || D; // statements
endmodule
```

Multiplexer Design





Sel[1:0]	mux_out
00	i0
01	i1
10	i2
11	i3

- A multiplexer of 2^n inputs has n select lines, are used to select which input line to send to the output.
- There is only one output in the multiplexer
- Simple 4: 1 Multiplexer with a registered output
- Based on the select input Sel[1:0], mux_out shall be any one of i0, i1, i2, i3 as shown in truth

You

module <module Name> (ports);

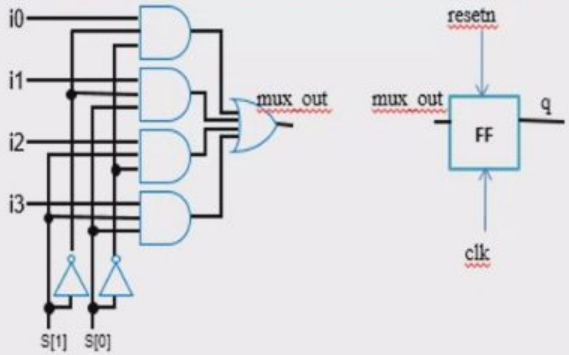
Inputs, outputs, Inout ports; parameters

Assign statements (concurrent)

Initial statements (procedural)

Always statements (procedural)

endmodule




```
// Multiplexer in Data flow model
module mux_ex ( i0, i1, i2,i3,clk, resetn, sel,q);
 input clk, resetn;
 input  i0, i1,i2,i3;
 input [1:0] sel;
 output q;

 reg  q_out;
 wire mux_out;

 // concurrent assignments
 assign mux_out = sel[1] ? ( sel[0] ? i3: i2) : ( sel[0] ? i1: i0);


always @(posedge clk or resetn)
   begin
    if (!resetn)
     q_out <= 0;
    else
     q_out <= mux_out;
   end
 assign q = q_out;

endmodule
```

ports

Module

Module I/Os

Wire, reg declarations

Concurrent assignments

Procedural block

Sensitivity list

Sequential statements

End module