Module #06 : Verilog HDL Modules

6.1: Module Declaration:

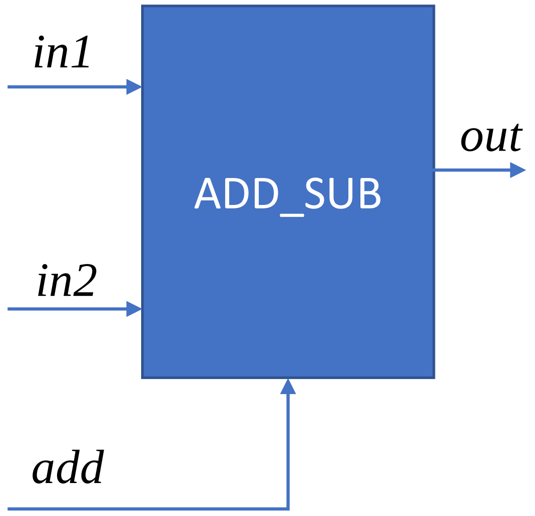
- A module is the principal design entity in Verilog. The first line of a module declaration specifies the name and port list (arguments).

Syntax:

```
module module_name (port_list);
  input [msb:lsb] input_port_list;
  output [msb:lsb] output_port_list;
  inout [msb:lsb] inout_port_list;
   ... statements ...
endmodule
```

Example 6.1:

```
module add_sub(add, in1, in2, out);
input add; // defaults to wire
input [7:0] in1, in2; wire in1, in2;
output [7:0] out; reg out;
      ... statements ...
endmodule
```



Module #06 : Verilog HDL Modules

6.2: Module Instantiations:

- Module declarations are designs from which one creates actual objects (instantiations).
- Modules are instantiated inside other modules, and each instantiation creates a unique object from the actual design. The exception is the top-level module which is its own instantiation
- The instantiated module's ports must be matched to those defined in the actual module design.
- Modules may not be instantiated inside procedural blocks.(always & initial block)
- This is specified as –

1) by name, using a dot(.) " .design_port_name (name_of_wire_connected_to_port)".

Example 6.2:

```
// Passing arguments by using port names a, b, c
module byname (a, b, c);
. . .
// Module instantiation in a top level module
byname bynameinstance(c.z, a.x, b.y)
```

Module #06 : Verilog HDL Modules

2) by position, placing the ports in exactly the same positions in the port lists of both the design and the instance

Example 6.3:

```
// Passing arguments(signals) by position.
module byposition (a, b, c);
. . .
// Module instantiation in a top level module
byposition bypositioninstance(x, y ,z);
```

Module #06 : Verilog HDL Modules

6.3: Parameterized Modules:

- You can build modules that are parameterized and specify the value of the parameter at each instantiation of the modules

Syntax:

module_name #(1st_parameter_values, 2nd_parm_value, ...) instance_name(port_connection_list);

Example 6.4:

```
 //Module Definition
module shift_n (it, ot); // used in module test_shift.
input [7:0] it; output [7:0] ot;
parameter n = 2;‘ // default value of n is 2
assign ot = (it << n); // it shifted left n times
endmodule
// Module instantiation in a top level module
wire [7:0] in1, ot1, ot2, ot3;
shift_n shft2(in1, ot1), // shift by 2; default
shift_n #(3) shft3(in1, ot2); // shift by 3; override parameter 2.
shift_n #(5) shft5(in1, ot3); // shift by 5; override parameter 2.
```

Module #06 : Verilog HDL Modules

- Macros do string substitutions and do many of the jobs parameters do. They are good for global parameters because they do not have to be passed through modules

Syntax:

define macro_name value;

Example 6.5:

```
// USING MACROS LIKE PARAMETERS
`define M 8 // Word width
module top
wire [`M -1:0] x, y, z; // equivalent to wire [8-1:0] x,y,z;
```

Module #06 : Verilog HDL Modules

6.4: Continuous Assignment:

- The continuous assignment is used to assign a value onto a wire in a module. It is the normal assignment outside of always or initial blocks (Module #07).
- Continuous assignment is done with an explicit assign statement or by assigning a value to a wire during its declaration.
- Note that continuous assignment statements are concurrent and are continuously executed during simulation.
- The order of assign statements does not matter. Any change in any of the right-hand-side inputs will immediately change a left-hand-side output.

Syntax:

wire wire_variable = value;
assign wire_variable = expression;

Example 6.6:

wire [1:0] a = 2'b01; // assigned on declaration
assign b = c & d; // using assign statement
assign d = x | y;
/ The order of the assign statements*
*does not matter. */*