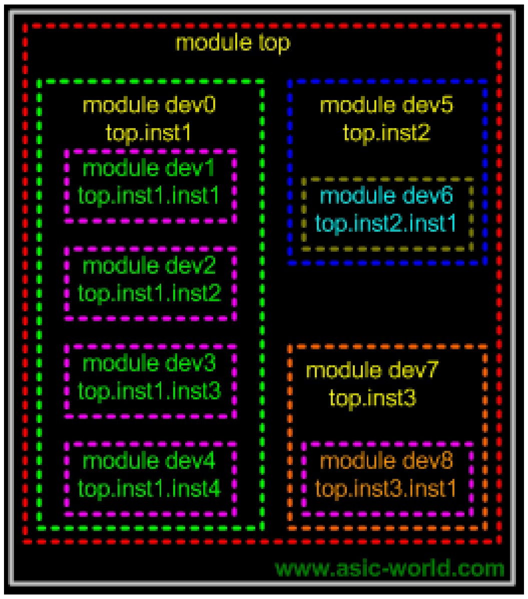
Verilog HDL Syntax And Semantics Part-II

Feb-9-2014

Modules

- Modules are the building blocks of Verilog designs
- You create the design hierarchy by instantiating modules in other modules.
- You instance a module when you use that module in another, higher-level module.



Ports

- Ports allow communication between a module and its environment.
- All but the top-level modules in a hierarchy have ports.
- Ports can be associated by order or by name.

You declare ports to be input, output or inout. The port declaration syntax is :
input [range_val:range_var] list_of_identifiers;
output [range_val:range_var] list_of_identifiers;
inout [range_val:range_var] list_of_identifiers;

NOTE : As a good coding practice, there should be only one port identifier per line, as shown below

Examples : Port Declaration

```
input                clk      ; // clock input
input  [15:0]        data_in  ; // 16 bit data input bus
output [7:0]          count    ; // 8 bit counter output
inout                data_bi  ; // Bi-Directional data bus
```

You could download file port_declare.v here

Examples : A complete Example in Verilog

```
module addbit (
a      , // first input
b      , // Second input
ci     , // Carry input
sum    , // sum output
co       // carry output
);
//Input declaration
input a;
input b;
input ci;
//Ouput declaration
output sum;
output co;
//Port Data types
wire  a;
wire  b;
wire  ci;
wire  sum;
wire  co;
//Code starts here
assign {co,sum} = a + b + ci;

endmodule // End of Module addbit
```

You could download file addbit.v here

Modules connected by port order (implicit)

Here order should match correctly. Normally it's not a good idea to connect ports implicitly. It could cause problem in debug (for example: locating the port which is causing a compile error), when any port is added or deleted.

```
//-----------------------------------------------------
// This is simple adder Program
// Design Name : adder_implicit
// File Name   : adder_implicit.v
// Function    : This program shows how implicit
//              port connection are done
// Coder      : Deepak Kumar Tala
//-----------------------------------------------------
module adder_implicit (
result        , // Output of the adder
carry         , // Carry output of adder
r1            , // first input
r2            , // second input
ci              // carry input
);

// Input Port Declarations
input    [3:0]   r1         ;
input    [3:0]   r2         ;
input            ci         ;

// Output Port Declarations
output   [3:0]  result      ;
output          carry       ;

// Port Wires
wire     [3:0]    r1        ;
wire     [3:0]    r2        ;
wire              ci        ;
wire     [3:0]    result    ;
wire              carry     ;

// Internal variables
wire              c1        ;
wire              c2        ;
wire              c3        ;

// Code Starts Here
addbit u0 (
r1[0]           ,
r2[0]           ,
ci              ,
result[0]       ,
c1
);

addbit u1 (
r1[1]          ,
r2[1]          ,
c1             ,
result[1]      ,
c2
);

addbit u2 (
r1[2]          ,
r2[2]          ,
c2             ,
result[2]      ,
c3
);

addbit u3 (
r1[3]          ,
r2[3]          ,
c3            ,
result[3]     ,
carry
);

endmodule // End Of Module adder
```

You could download file adder_implicit.v here

Modules connected by name

Here the name should match with the leaf module, the order is not important.

```
//-----------------------------------------------------
// This is simple adder Program
// Design Name : adder_explicit
// File Name   : adder_explicit.v
// Function    : Here the name should match
// with the leaf module, the order is not important.
// Coder       : Deepak Kumar Tala
//-----------------------------------------------------
module adder_explicit (
result        , // Output of the adder
carry         , // Carry output of adder
r1            , // first input
r2            , // second input
ci              // carry input
);

// Input Port Declarations
input    [3:0]   r1         ;
input    [3:0]   r2         ;
input            ci         ;

// Output Port Declarations
output    [3:0]  result     ;
output           carry      ;

// Port Wires
wire     [3:0]   r1         ;
wire     [3:0]   r2         ;
wire             ci         ;
wire     [3:0]   result     ;
wire             carry      ;

// Internal variables
wire             c1         ;
wire             c2         ;
wire             c3         ;

// Code Starts Here
addbit u0 (
.a          (r1[0])         ,
.b          (r2[0])         ,
.ci         (ci)            ,
.sum        (result[0])     ,
.co          (c1)
);

addbit u1 (
.a           (r1[1])        ,
.b           (r2[1])        ,
.ci          (c1)           ,
.sum         (result[1])    ,
.co          (c2)
);

addbit u2 (
.a           (r1[2])        ,
.b           (r2[2])        ,
.ci          (c2)           ,
.sum         (result[2])    ,
.co          (c3)
);

addbit u3 (
.a           (r1[3])        ,
.b           (r2[3])        ,
.ci          (c3)           ,
.sum         (result[3])    ,
.co          (carry)
);

endmodule // End Of Module adder
```

You could download file adder_explicit.v here

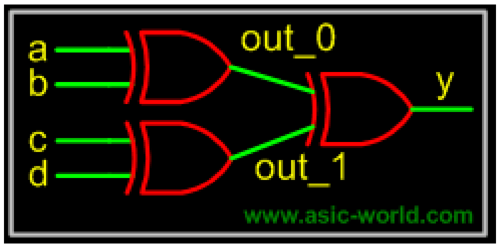
Instantiating a module

```
//-----------------------------------------------------
// This is simple parity Program
// Design Name : parity
// File Name   : parity.v
// Function    : This program shows how a verilog
//               primitive/module port connection are done
// Coder       : Deepak
//-----------------------------------------------------
module parity (
a      , // First input
b      , // Second input
c      , // Third Input
d      , // Fourth Input
y        // Parity  output
);

// Input Declaration
input       a       ;
input       b       ;
input       c       ;
input       d       ;
// Ouput Declaration
output      y       ;
// port data types
wire        a       ;
wire        b       ;
wire         c       ;
wire         d       ;
wire         y       ;
// Internal variables
wire         out_0 ;
wire         out_1 ;

// Code starts Here
xor u0 (out_0,a,b);

xor u1 (out_1,c,d);

xor u2 (y,out_0,out_1);

endmodule  // End Of Module parity
```

You could download file parity.v here

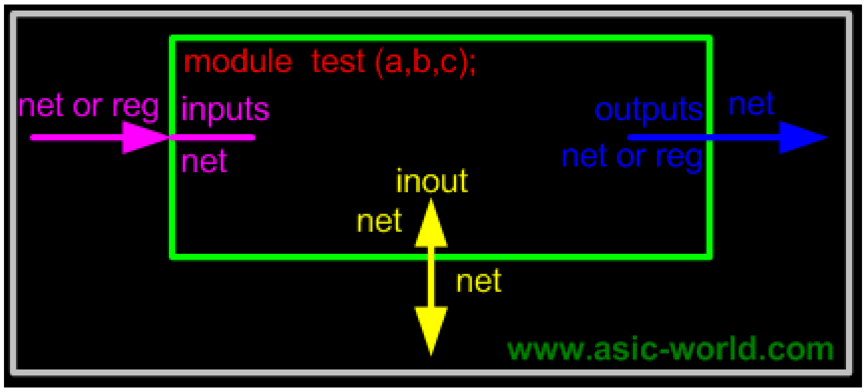
Question : What is the difference between u0 in module adder and u0 in module parity?

Schematic



Port Connection Rules

- Inputs : internally must always be of type net, externally the inputs can be connected to a variable of type reg or net.
- Outputs : internally can be of type net or reg, externally the outputs must be connected to a variable of type net.
- Inouts : internally or externally must always be type net, can only be connected to a variable net type.



- Width matching : It is legal to connect internal and external ports of different sizes. But beware, synthesis tools could report problems.

- Unconnected ports : unconnected ports are allowed by using a ",".
- The net data types are used to connect structure.
- A net data type is required if a signal can be driven a structural connection.

Example - Implicit Unconnected Port

```
module implicit();
reg clk,d,rst,pre;
wire q;

// Here second port is not connected
dff u0 ( q,,clk,d,rst,pre);

endmodule

// D fli-flop
module dff (q, q_bar, clk, d, rst, pre);
input clk, d, rst, pre;
output q, q_bar;
reg q;

assign q_bar = ~q;

always @ (posedge clk)
if (rst == 1'b1) begin
  q <= 0;
end else if (pre == 1'b1) begin
  q <= 1;
end else begin
  q <= d;
end

endmodule
```

You could download file implicit.v here

Example - Explicit Unconnected Port

```
module explicit();
reg clk,d,rst,pre;
wire q;

// Here q_bar is not connected
// We can connect ports in any order
dff u0 (
.q          (q),
.d  (d),
.clk        (clk),
.q_bar      (),
.rst        (rst),
.pre        (pre)
);
```

```verilog

endmodule

// D fli-flop
module dff (q, q_bar, clk, d, rst, pre);
input clk, d, rst, pre;
output q, q_bar;
reg q;

assign q_bar = ~q;

always @ (posedge clk)
if (rst == 1'b1) begin
  q <= 0;
end else if (pre == 1'b1) begin
  q <= 1;
end else begin
  q <= d;
end

endmodule
```

You could download file explicit.v here

BACK HOME NEXT