Digital System Design with HDL (I)

Lecture 4

Dr. Ming Xu

Dept of Electrical & Electronic Engineering

XJTLU

In This Session

- Verilog Modules
  - Module Definitions
  - Module Port Declarations
  - Module Instantiations

Module Definitions

- A *module* is a circuit or subcircuit in Verilog code.
- It is comprised of the interface and the design behavior.

module *module_name* (*port_name*, *port_name*, ...);
  module_port_declarations
  module_items
endmodule

```
module fulladd (Cin, x, y, s, Cout);
 input Cin, x, y;
 output s, Cout;

 assign {Cout, s} = x + y + Cin;

endmodule
```


Module Definitions

Module_items

  data type declarations
  module instances
  primitive instances
  procedural blocks
  continuous assignments
  function or task definitions

They may appear in any order, but data type declarations should be listed before the signals are referenced.

Module Port Declarations

A ***port*** refers to an input or output connection in a circuit.

Syntax:

port_type [port_size] *port_name*, *port_name*, ... ;

port_type is declared as:

- **input** for scalar or vector input ports.
- **output** for scalar or vector output ports.
- **inout** for scalar or vector bi-directional ports.

port_size is a range from [*msb* : *lsb*]

Module Port Declarations

Examples

input a,b,sel; // 3 scalar ports

output [7:0] result;

inout [15:0] data_bus;

input [15:12] addr; // msb:lsb may be any integer

parameter word = 32;
input [word-1:0] addr; // use constant expressions

Module Instantiations

- A Verilog module can be included as a subcircuit in another module.
- Both modules must be defined in the same file or the Verilog compiler must be told where they are.
- A module may be instantiated using **port order** or **port names**.

Module Instantiations

Port Order Connections

module_name instance_name (*signal*, *signal*, ...);

- Signal connections are listed in the same order as the port list in the module definition.
- Unconnected ports are designated by two commas with no signal listed.
- instance_name is required so that multiple instances of the same module are unique from one another.

Module Instantiations

Port Name Connections

module_name instance_name (.*port_name*(*signal*),
(.*port_name*(*signal*), ...);

- Port names and signals connected to them are listed in pairs, in any order.

Module Instantiations

Example: A 1-bit adder module *fulladd*

```
module fulladd (Cin, x, y, s, Cout);
  input Cin, x, y;
  output s, Cout;

  assign {Cout, s} = x + y + Cin;

endmodule
```


Module Instantiations

Example: A 4-bit adder built using 4 instances of *fulladd*

```
module adder4 (carryin, X, Y, S, carryout);
  input carryin;
  input [3:0] X, Y;
  output [3:0] S;
  output carryout;
  wire [3:1] C;

  fulladd stage0 (carryin, X[0], Y[0], S[0], C[1]);
  fulladd stage1 (C[1], X[1], Y[1], S[1], C[2]);
  fulladd stage2 (C[2], X[2], Y[2], S[2], C[3]);
  fulladd stage3 (.Cout(carryout), .s(S[3]), .y(Y[3]), .x(X[3]), .Cin(C[3]));

endmodule
```


Module Instantiations

Parameter Redefinition
Parameters in a module may be redefined for each instance.

Explicit Parameter Redefinition
module_name instance_name (*signal*, *signal*, ...);
defparam *instance_name*.*parameter_name* = *value*;

Implicit Parameter Redefinition
module_name #(*value*) *instance_name* (*signals*);

In implicit redefinition, parameters must be redefined in the same order they are declared within the module.

Module Instantiations

Example:
A bit-counting module which contains two parameters n and logn

```
module bit_count (X, Count);
 parameter n = 4;
 parameter logn = 2;
 input [n-1:0] X;
 output reg [logn:0] Count;
 integer k;

 always @(X)
 begin
  Count = 0;
  for (k = 0; k < n; k = k+1)
   Count = Count + X[k];
 end

endmodule
```


Module Instantiations

Example:
Overriding module parameters using explicit parameter redefinition.

```
module common (X, Y, C);
 input [7:0] X, Y;
 output [3:0] C;
 wire [7:0] T;

 // Make T[i] = 1 if X[i] == Y[i]
 assign T = X ~^ Y;

 bit_count cbits (T, C);
  defparam cbits.n = 8, cbits.logn = 3;

endmodule
```


Module Instantiations

Example:
Overriding module parameters using implicit parameter redefinition.

```
module common (X, Y, C);
 input [7:0] X, Y;
 output [3:0] C;
 wire [7:0] T;

 // Make T[i] = 1 if X[i] == Y[i]
 assign T = X ~^ Y;

 bit_count #(8,3) cbits (T, C);

endmodule
```


The **generate** Construct

- n instances of the *fulladd* subcircuit are required to build a n-bit ripple-carry adder.
- The **generate** construct (Verilog 2001) allows subcircuits to be instantiated in a loop.
- The loop index must be declared of type **genvar** – a positive integer that appears only in **generate** blocks.

Syntax:

generate
  [procedural statements]
    [instantiation statements]
endgenerate

The **generate** Construct

- Each instance has an instance name addbit[i].stage.

```
module ripple_g (carryin, X, Y, S, carryout);
  parameter n = 4;
  input carryin;
  input [n-1:0] X, Y;
  output [n-1:0] S;
  output carryout;
  wire [n:0] C;

  genvar i;
  assign C[0] = carryin;
  assign carryout = C[n];

  generate
    for (i = 0; i <= n-1; i = i+1)
    begin:addbit
      fulladd stage (C[i], X[i], Y[i], S[i], C[i+1]);
    end
  endgenerate

endmodule
```