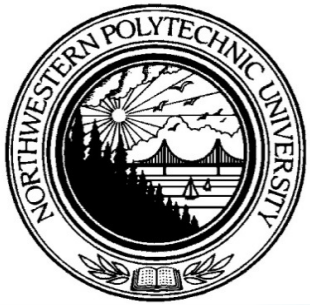



EE461 Verilog-HDL

Week 12 Compiler Directives



Alex Yang, Engineering School, NPU

Week 12 Outlines

- Defining Macros
  - `define & `undef macros
  - difference between `define and parameter
- Conditional Compilation Directive
  - `ifdef or `ifndef
- Reset all directives
  - `resetall
- Net Type Specification
  - `default_nettype

Defining Macros

- `define Macro (synthesizable)
  - define global constant
    - e.g. `define SIZE 8
    - should be in the front of module
    - it is fine within module, but don't recommend

```
`define SIZE 8
module example;
    reg [`SIZE - 1 : 0] data_out;
   ... ...
endmodule
```

Not good: (works)

```
module example;
    `define SIZE 8
    reg [`SIZE - 1 : 0] data_out;
   ... ...
endmodule
```

Defining Macros

- `define macro with arguments
  - like function declaration.
  - "\": start a newline. Actually all statements in define block should be in one line.

```
`define SIZE    8
`define    xor_b(x,y)    (x & !y)|(!x & y)
module example
      reg [ `SIZE - 1 : 0] data_out;
      c = `xor_b(a, b);
       ... ...
endmodule
```

```
`include "and_gate.v"
`include "or_gate.v"
`include "xor_gate.v"
`include "nand_gate.v"
`define GATE(M) M\
            _gate M\
            _g (.a(M\
            _a), .b(M\
            _b), .c(M\
            _c));
```

```
`define SIG(S) and_\
            S, or_\
            S,xor_\
            S,nand_\
            S; \
module top();
      wire `SIG(a)
      wire `SIG(b)
      wire `SIG(c)

      `GATE(and)
      `GATE(or)
      `GATE(xor)
      `GATE(nand)
endmodule
```

Defining Macros

- `undef Macro
  - disable previous defined macro
    - e.g. `undef SIZE

```
`define SIZE    8

module example
      reg [ `SIZE - 1 : 0] data_out;
                  ... ...
    `undef SIZE
       ... ...
endmodule
```

Defining Macros

- Difference between `define and parameter
  - `define: is global cross all submodules and last define will override the previous one. It can't be used in FSM for states
  - parameter: valid only within module

```
`define Width 8
`include "submodule.v"
module testDef;
        initial begin
          $display("Width in top:%0d",`Width);
        end
endmodule
```

```
`define Width 32
module submodule;
        initial begin
          $display("Width in submodule:%0d",`Width);
        end
endmodule
```

```
npu29.npu.edu - PuTTY
[yangjh@npu29 Test_define]$ ./simv
Chronologic VCS simulator copyright 1991-2011
Contains Synopsys proprietary information.
Compiler version E-2011.03; Runtime version E-2011.03;  Feb 28 03:43 2014
Width in submodule: 32
Width in top: 32
                V C S   S i m u l a t i o n   R e p o r t
Time: 0
CPU Time:      0.280 seconds;       Data structure size:   0.0Mb
Fri Feb 28 03:43:52 2014
```

Conditional Compilation Directive

- `ifdef or `ifndef
  - Optionally compile the statements
  - Is used to avoid multiple compilations for one module

top.v :
`include "submodule1.v" =>

`include "submodule2.v =>
module top.v;
endmodule

submodule1.v :
`include "submodule21.v" =>
module submodule1.v;
endmodule

submodule2.v
`include "submodule21.v" =>
module submodule2.v;
endmodule

submodule21.v :
module submodule21;
endmodule

Compile submodule21.v again. You will get error.

Conditional Compilation Directive

- `ifdef or `ifndef

top.v :
`include "submodule1.v" =>

`include "submodule2.v" =>
module top.v
endmodule

submodule1.v :
`ifndef XXX
`define XXX
`include "submodule21.v" =>
module submodule1.v
endmodule
`endif
submodule2.v :
`ifndef YYY
`define YYY
`include "submodule21.v" =>
module submodule2.v
endmodule
`endif

submodule21.v :
`ifndef ZZZ
`define ZZZ
module submodule21;
endmodule
`endif

ZZZ has been defined during above compilation.So submodule21.v can't be compiled twice.

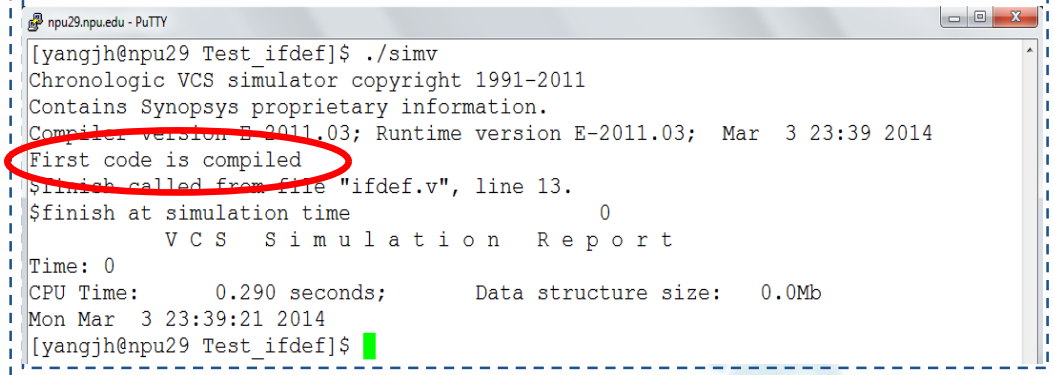
Conditional Compilation Directive

- `ifdef or `ifndef

```
module ifdef ();
    initial begin
    `ifdef FIRST
  $display("First code is compiled");
    `else
        `ifdef SECOND
   $display("Second code is compiled");
        `else
   $display("Default code is compiled");
        `endif
    `endif
     $finish;
end

endmodule
```

```
[ABC@npu29 ~]$ vcs +define+FIRST  ifdef.v
```



Reset all directives

- `resetall
  - resets all compiler directives to default values

```
/* Up-Down Counter with latch enable
and count enable. */
`resetall
`timescale 1ns/10ps

module CntrUpDown(D, LE, CE, Up,
                  Reset, Clk, Count);

parameter Width = 8;

input [Width-1:0] D;
input LE, CE, Up, Reset, Clk;
output [Width-1:0] Count;

wire [Width-1:0] D;
wire LE, CE, Up, Reset, Clk;
reg [Width-1:0] Count ;
reg [Width-1:0] CountD ;
```

```
// Combinational Block for CountD
  always @ (*)begin
    CountD = Count ;
    if (LE)  CountD = D ;
   else if (CE)
     CountD = Up ? Count + 1 : Count - 1 ;
  end

  always @ (posedge Clk) begin
    if (Reset)
      Count <= 0 ; // Reset Count
    else
      Count <= CountD ;
  end
endmodule
```

Net Type Specification

- `default_nettype
  - Can we use undefined net type variable before declaration? Yes. By default, it is wire type.
  - `default_nettype could help to change this default type for undefined variable, such as
    - `default_nettype wor / `default_nettype wand

```
module test(
                        input wire a1,
                        input wire a2,
                        output wire c1
 );

assign cl=a1&a2;        // cl is not c1
endmodule
```

Q: can it pass compilation?
Ans: Yes, no error. But c1 is not a1&a2.

```
`default_nettype none
// it is to require net declaration
module test(
                        input wire a1,
                        input wire a2,
                        output wire c1
 );
assign cl=a1&a2;
endmodule
```

Q: can it pass compilation?
Ans: No, error. Net variable can't be used before declaration if adding this directive