


EE461 Verilog-HDL

Week 2 Semantics & Syntax

Alex Yang, Engineering School, NPU

Week 2 Outlines

- Verilog Semantics & Syntax
  - Modules
  - Lexical Convention
  - Data Types
  - Numbers in Verilog
  - Strings
  - Port Connection
  - Hierarchical Identifiers
  - $display, $write, $monitor and $strobe

Modules

- Module Structure

```
module moduleName(Port List);
        Port Declarations;
        Data Type Declarations;
        Circuit Functionality;
endmodule
```

Modules

- Ports & Data Type Declaration in Modules

```
module FourBitFA1(
                        a_i,                    //Port list
                        b_i,
                        ci_i,
                        sum_o,
                        co_o
);
    input [3:0]         a_i;                    //Port direction declaration
    input [3:0]         b_i;                    //By default, data type is wire
    input [3:0]         ci_i;                   // [3:0] => 4-bit input
    output [3:0]        sum_o;
    output              co_o;

    reg [3:0]           sum_o;                  //Date type declaration
    reg                 co_o;

    always @(a_i or b_i or ci_i) begin          // Or always @(*)
            sum_o  = a_i ^ b_i  ^ ci_i;         // a_i  XOR   b_i    XOR   ci_i
            co_o = a_i & b_i  |  b_i & ci_i  | a_i & ci_i;          // & => logic AND; | => logic  OR
    end
endmodule
```

Lexical Convention

- White Space
- Comments
- Case Sensitivity
- Identifiers
- Escaped Identifiers

Lexical Convention

- White Space
  - Blank Space
  - Tabs \t
  - Newline \n

Bad code style: no indentation/no space/no newline

```
module FourBitFA1(a_i,b_i,ci_i,sum_o,co_o);
input [3:0]         a_i,b_i,ci_i;
output [3:0] sum_o;
output co_o;
assign {co_o,sum_o}=a_i+b_i+ci_i;   endmodule
```

Lexical Convention

- White Space

  Good code style:

```
module FourBitFA1(
                    a_i,
                    b_i,
                    ci_i,
                    sum_o,
                    co_o
);
  input [3:0]       a_i;
  input[3:0]        b_i;
  input[3:0]        ci_i;
  output[3:0]       sum_o;
  output            co_o;

  assign {co_o,sum_o}=a_i+b_i+ci_i;
endmodule
```

Lexical Convention

- Comments
  - Single line comment: //
  - Block comments: /*xxxxxx*/

To make your program readable, please put meaningful comments after each statement or a block of functional statements.

Bad comment:

```
a++;                    //Make “a” increase 1
```

Lexical Convention

- Case Sensitivity
  - Verilog is case sensitive
  - Keyword or reserved word is low case

input/output/inout => keywords in low case

Input/Wire/Reg/If => not keywords which could be defined as variable name.

However, never use Verilog keywords as names, even if the case is different.

Lexical Convention

- Identifiers

  Are names for module, function, task and variable

  - The first character must start with _ or a-z/A-Z, like _a-zA-Zxxxx.
  - The rest of them must be _ or a-z/A-Z/0-9 or $.
  - Identifiers can be up to 1024 characters long.

  Which one is a legal identifier?

  _1234_; $_; _ _; $$; 12abc_%; _$_; &xyz; input$clk;

Lexical Convention

- Escaped Identifiers

  If you really want to define an identifier in other characters except mentioned above, such as “$1234~” , escaped identifier could help.

  Format: \xxxxx(one space)

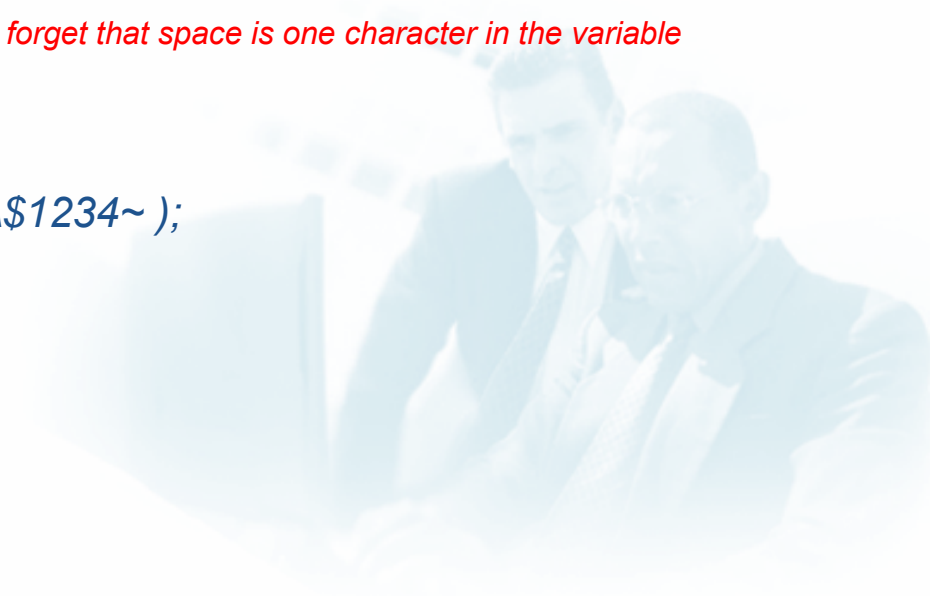
  \$1234~ ;

  Notice that a space must be at the end.

Lexical Convention

- Escaped Identifiers

```
module EscIden;
        integer __;             //Can we add space at the end of "__"?
                                // Ans: yes, we can. But variable name is still "__"
      integer \$1234~ ;         //One space must be at the end
      integer \$1234~,abc ; //What is variable name?
      initial begin
                __=10;
                \$1234~ = 20;          // Don't forget that space is one character in the variable
                \$1234~,abc = 30;
                $display("__ =%d", __ );
                $display("$1234~ =%d",  \$1234~ );
                #10 $finish;
      end
endmodule
```



Data Types

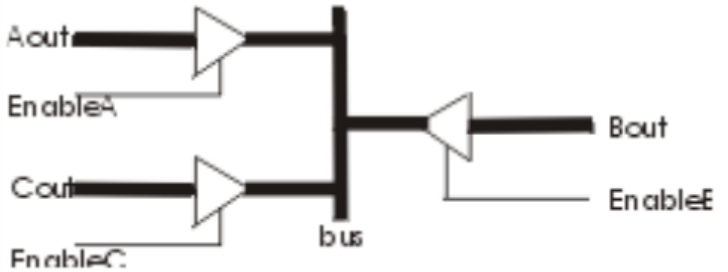
- Nets – Structural Connection
  - wire, tri — Synthesizable
  - wor, trior — Non-synthesizable
  - wand, triand — Non-synthesizable
  - tri0 tri1 — Non-synthesizable
  - supply0 supply1 — Synthesizable
  - trireg — Non-synthesizable
- Register – Store Data
  - reg — unsigned — Synthesizable
  - integer — signed-32bits — Synthesizable
  - time — signed-64bits — Non-synthesizable
  - real — signed-64bits floating Number — Non-synthesizable

Data Types

- Nets – Structural Connection

tri: tri-state Synthesizable

All drivers connected to a tri variable must be z, except one(which determine the value of tri)



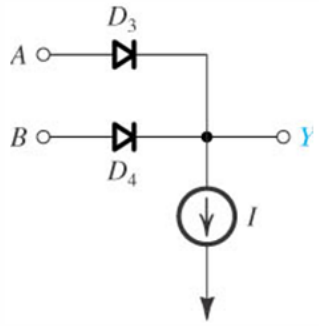

```
module triState(… …);
        tri [7:0] bus;
        reg [7:0] Aout, Bout, Cout;
        reg EnableA, EnableB, EnableC;

        assign bus = EnableA ? Aout : 8'hzz;
        assign bus = EnableB ? Bout : 8'hzz;
        assign bus = EnableC ? Cout : 8'hzz;
… …  // If EnableA = 1 & EnableB = 1 EnableC=0, assuming Aout/ Bout is not Hi-Z;
          // what will happen?
          // Ans: error because tri variable just allows one value to drive.
endmodule
```

Data Types

- ## Nets – Structural Connection

wor, trior : wire OR — Non-synthesizable




```
module TestWor();
        wor Y;
        reg A, B;

        assign Y = A;
        assign Y = B;
        initial begin                          //Non-synthesizable block, only for simulation
                $monitor("%g Y = %b A = %b B = %b", $time, Y, A, B);
                                               //%g – General format floating point number
                #1 A = 0;
                #1 B = 0;
                #1 A = 1;
                #1 A = 0;
                #1 B = 1;
                #1 A = 1;
                #1 A = 0;
                #1 $finish;
        end
endmodule                                      //What are results?
```

Data Types

- Nets – Structural Connection

wor, trior : wire OR Non-synthesizable

```
module TestTrior;
        trior bus;
        reg Aout, Bout;
        reg EnableA, EnableB;

        assign bus = EnableA ? Aout : 8'hz;
        assign bus = EnableB ? Bout : 8'hz;

         initial begin
                                $monitor("time=%g, bus=%b",$time, bus);
                                EnableA=1;
                                EnableB=1;
                                Aout=0;
                                Bout=0;
                                #1 Aout=1;
                                #1 Bout=1;
                                #2 $finish;          //What are results?
          end
endmodule
```

Data Types

- Nets – Structural Connection

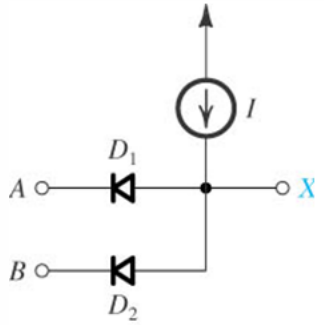
  Values of wor/trior: 0, 1, x(unknow), z

wor/trior	0	1	X	Z
0	0	1	X	0
1	1	1	1	1
X	X	1	X	X
Z	0	1	X	Z

Data Types

- ## Nets – Structural Connection

wand, triand : wire AND Non-synthesizable




```
module testWand();
        wand X;
        reg A, B;

        assign X = A;
        assign X = B;

        initial begin
                $monitor("%g X = %b A = %b B = %b", $time, X, A, B);
                #1 A = 0;
                #1 B = 0;
                #1 A = 1;
                #1 A = 0;
                #1 B = 1;
                #1 A = 1;
                #1 A = 0;
                #1 $finish;
        end
endmodule
```

Data Types

- Nets – Structural Connection

wand/triand	0	1	X	Z
0	0	0	0	0
1	0	1	X	1
X	0	X	X	X
Z	0	1	X	Z

Data Types

- ## Nets – Structural Connection

  tri0 tri1 : Net pulls-down or pulls-up when not driven, namely tri0=0 if no value is bounded to it or z is assigned to it; likewise, tri1 =1. Non-synthesizable

```
module testTri0();
    wire    w1, w2, w3, w4;
    tri0    t01, t02, t03, t04;
    tri1    t11, t12, t13, t14;

    assign  w1 = 0;
    assign  t01 = 0;
    assign  t11 = 0;
    assign  w2 = 1'bz;
    assign  t02 = 1'bz;     // if t02=1'bx, what is result?
    assign  t12 = 1'bz;     //Ans: t02 = x;
    assign  w3 = 1;
    assign  t03 = 1;
    assign  t13 = 1;

    initial begin
        #1;         $display(w1, w2, w3, w4);     //By default, w4 = 1'bz if there is no value to bound to variable
                    $display(t01, t02, t03, t04);
                    $display(t11, t12, t13, t14);
    end
endmodule
```

Results:

```
0 z 1 z
0 0 1 0
0 1 1 1
```

Data Types

- ## Nets – Structural Connection

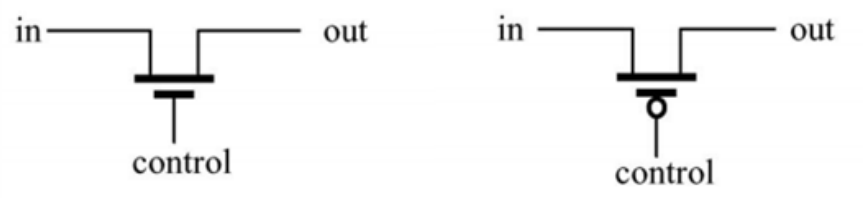
supply0 supply1 : Net has a constant logic 0 or logic 1 (supply strength) synthesizable

```
module testSupply( a, b, y);
    input       a;
    input       b;
    output      y;

    supply0     gnd;
    supply1     vdd;
    wire        im1;

    pmos  M1(y, vdd, a);    // pmos Name(output, input, control);
    pmos  M2(y, vdd, b);
    pmos  M3(y, im1, a);
    pmos M4(im1, gnd, b);

endmodule
```



Data Types

- Nets – Structural Connection

trireg: Retains last value, when driven by z (tristate).

```
module testTrireg ();
        trireg[7:0]   data;
        reg[1:0]      flag;

        assign        data = (flag==1)?       10    :
                             (flag==0)?       8'bzz:
                             (flag==3)?       30    :  255;
        initial begin
                      flag = 1;
                      #200 flag=0;
                      #200 flag=3;
                      #200 flag=0;
                      #200 flag=2;
                      #200 flag=0;
                      $monitor("Time=%g, data=%d", $time, data);
                      #10  $finish;
        end
endmodule
```

Results:

Time=1000,data=255

Data Types

- Register – Store Data
  - reg unsigned Synthesizable
  - integer signed-32bits Synthesizable
  - time signed-64bits Non-synthesizable
  - real signed-64bits floating Number Non-synthesizable

Let us focus on integer, time and real first.

Data Types

- Register – Store Data
  - integer signed-32bits Synthesizable

```
module testInteger;
    wire pwrGood, pwrOn, pwrStable;        // Explicitly declare wires.
    integer i;                             // 32-bit, signed (2's complement).
    time t;                                // 64-bit, unsigned, behaves like a 64-bit reg.
    real r;                                // Real data type of implementation defined size.
                                           // An assign statement continuously drives a wire:
    assign pwrStable = 1'b1;               //Net type must be in “assign” block to be bounded  to a value
    assign pwrOn = 1; // 1 or 1'b1
    assign pwrGood = pwrOn & pwrStable;
    initial begin                          //integer/time/real must be in initial or always block to be assigned  to a value
      i = 123.456;                         // There must be a digit on either side
      r = 123456e-3;                       // of the decimal point if it is present.
      t = 123456e-3;                       // Time is rounded to 1 second by default.
      $display("i=%0g",i," t=%6.2f",t," r=%f",r); //%6.2f: at leas t 6 digits
    #2 $display("TIME=%0d",$time," ON=",pwrOn," STABLE=",pwrStable,“ GOOD=",pwrGood);
      $finish;
    end
endmodule
```

Results:

i=123 t=123.00 r=123.456000

TIME=2 ON=1 STABLE=1 GOOD=1

Data Types

- Register – Store Data
  - time signed-64bits Non-synthesizable

```
module testTime;
    time t; // 64-bit, unsigned, behaves like a 64-bit reg.
    initial begin
        t = $time;
        #1 t=$time;
        #1 t=$time;
        #2 $finish;
    end
    initial begin
        $monitor("t=%6.2f",t);
        #2 $display("TIME=%0d",$time);
    end
endmodule
```

Results:

```
t=  0.00
t=  1.00
TIME=2
t=  2.00
```

Data Types

- Register – Store Data
  - time signed-64bits integer Non-synthesizable

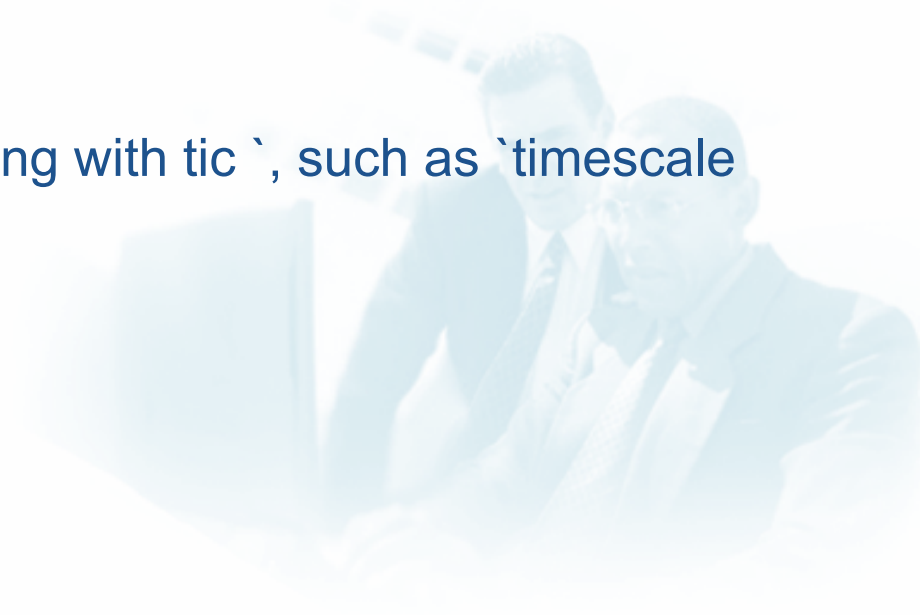
Other system built-in tasks:

$stime signed-32bits short integer

$realtime signed-64bits floating number

Compiler Directives:

Are special commands beginning with tic `, such as `timescale



Data Types

- Discussion about `timescale

Delay unit is specified using 'timescale, which is declared as

`timescale time_unit base / precision base

- time_unit is the amount of time a delay of #1 represents. The time unit must be 1 10 or 100
- base is the time base for each unit, ranging from seconds to femtoseconds, and must be: s ms us ns ps or fs
- precision and base represent how many decimal points of precision to use relative to the time units.

For example : `timescale 1 ns / 100 ps means time values to be read as ns and to be rounded to the nearest 100 ps.

Notice that precision base can’t be greater than unit base.

Data Types

- Discussion about `timescale

```
`timescale 10ps / 1ps
module sampleDesign (z,x1,x2);
    input x1, x2;
    output z;
    nor #3.57 (z, x1, x2);
endmodule
```

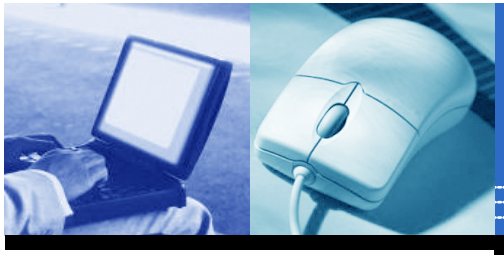
The nor gate's delay is 36 ps (3.57 x 10 = 35.7 ps rounded to 36).

Question: `timescale 10ns/1ns
#1.234 what is delay time?

Ans: 1.234x10ns = 12.34ns rounded to 12ns

How about `timescale 10ns/10ns #1.234 ?
12.34ns rounded to 10ns

If `timescale 10ns/100ns and #1.234, what do you get?



Numbers in Verilog

- Real Number

Real Number	Decimal notation
1.2	1.2
0.6	0.6
3.5E6	3,500000.0

- Signed & Unsigned Number

Real Number	Decimal notation
32'hDEAD_BEEF	Unsigned or signed positive number
-14'h1234	Signed negative number(2'complement)

Number format: 16'h1234; size(16) must be number of binary bit, not hex, not oct, etc.

Numbers in Verilog

- Integer Number
  - 32 bits, negative number(2'complement)
  - the question mark (?) character is the Verilog alternative for the z character.
  - Underscore(_) is legal.

Integer	Saved as
1	00000000000000000000000000000001
8'hAA	10101010
6'b10_0011	100011
'hF	00000000000000000000000000001111

Integer	Saved as
6'hCA	001010
6'hA	001010
16'bZ	ZZZZZZZZZZZZZZZZ
8'bx	xxxxxxxx

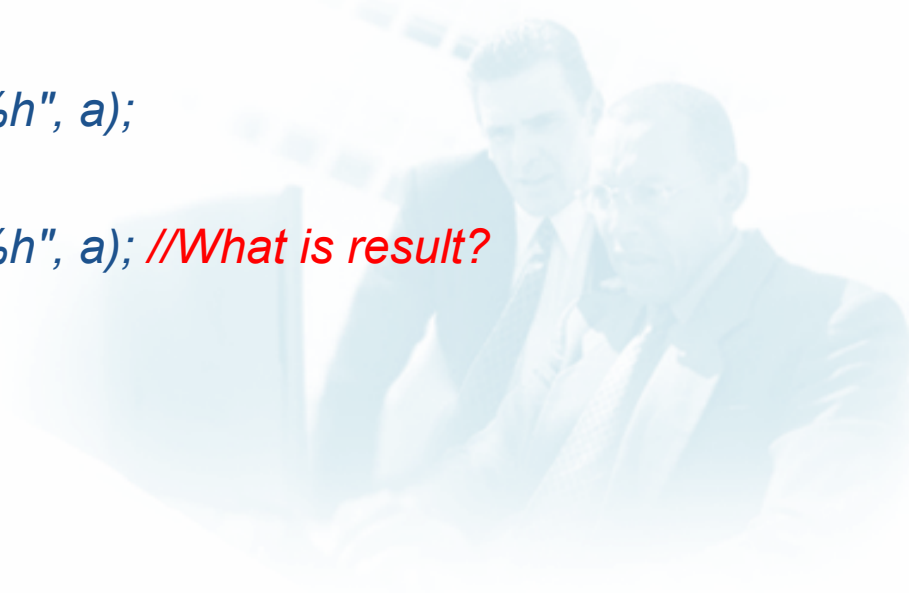
8'bx0 = xxxx_xxx0

4'd-4 -> illegal for negative number

Numbers in Verilog

```
module signedNumber;
reg [31:0]        a;

initial begin
        a = 14'h1234;
        $display ("Current Value of a = %h", a);
        a = -14'h1234;
        $display ("Current Value of a = %h", a); //What is result?
        a = 32'hDEAD_BEEF;
        $display ("Current Value of a = %h", a);
        a = -32'hDEAD_BEEF;
        $display ("Current Value of a = %h", a); //What is result?
        #10 $finish;
end
endmodule
```



Strings

```
module strings();
        reg [8*21:0]        string ; // Need to assign 21 characters to this variable

        initial begin
                string = "This is sample string";
                $display ("%s \n", string);        //If displaying “&”, how should we do?
        end                                        //Ans: $display("%s","&")
endmodule
```

How to display special character in $display system task, like %?

Character	Description
\n	New line character
\t	Tab character
\\	Backslash (\) character
\"	Double quote (") character
\ddd	Display a character (ASCII Code) specified in 1-3 octal digits (0 <= d <= 7)
%%	Percent (%) character

Question: $display("\1234") what is result? // print: S4 (S- ASCII oct # 123)

$display("\045"); (045 = %) does it work to display “%”? **// Ans: $display("\045\045")**

Port Connection

- Connected By Port Order(Don't recommend)

```
module addBit ( a_i,b_i,ci_i, sum_o,co_o);
… …
endmodule
module adderImplicit ( result_o, carry_o, r1_i , r2_i , ci_i);
    // Input Port Declarations
    input    [3:0]   r1_i, r2_i        ;
    input            ci_i;
    // Output Port Declarations
    output   [3:0]  result_o      ;
    output           carry_o       ;
    // Internal variables
    wire             c1_w,c2_w,c3_w ;
```

Port Connection

```
// Code Starts Here
addBit u0 (
                    r1_i[0]      ,
                    r2_i[0]      ,
                    ci_i         ,
                    result_o[0]  ,
                    c1_w
);
addBit u1 (
                    r1_i[1]      ,
                    r2_i[1]      ,
                    c1_w         ,
                    result_o[1]  ,
                    c2_w
);
endmodule // End Of Module adder
```

Port Connection

- Connected By Port Name(Recommend)

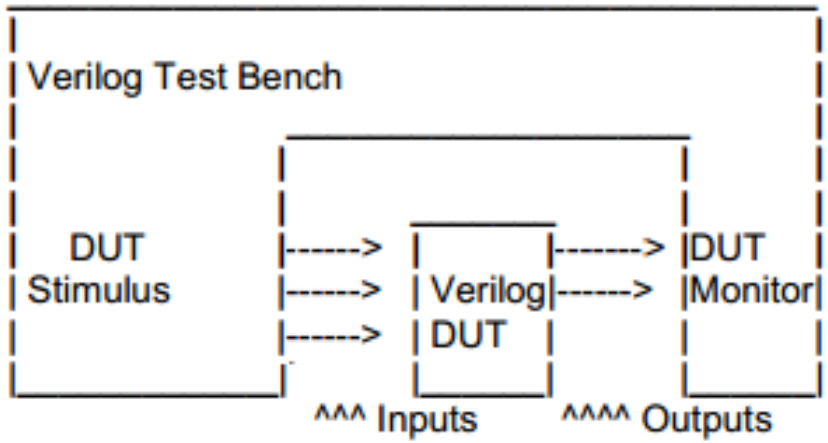
```
module adderExplicit(… …);
    … …
    addBit uAddBit0 (
                    .a_i        (r1_i[0])     ,
                    .b_i        (r2_i[0])     ,
                    .ci_i       (ci_i)        ,
                    .sum_o      (result_o[0])   ,
                    .co_o       (c1_w)
    );
    addBit uAddBit1 (
                    .a_i        (r1_i[1])     ,
                    .b_i        (r2_i[1])     ,
                    .ci_i       (c1_w)        ,
                    .sum_o      (result_o[1])   ,
                    .co_o       (c2_w)
    );
    … …
endmodule // End Of Module adder
```

Port Connection

- Port Connection Rule
  - 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.

Port Connection

- Port Connection Rule



- Data type to DUT in testbench: reg
- Data type from DUT in testbench: wire

Notice that testbench is not design module, and it is to verify your design module functionality.

Hierarchical Identifiers

```
`include "addBit.v“
module          adderHier (… …);
                … …
                addBit u0 (r1_i[0],r2_i[0],ci_i,result[_o0],c1_w);
                addBit u1 (r1_i[1],r2[_i1],c1_w,result_o[1],c2_w);
                addBit u2 (r1_i[2],r2_i[2],c2_w,result_o[2],c3_w);
                addBit u3 (r1_i[3],r2_i[3],c3_w,result_o[3],carry_o);
endmodule // End Of Module adder
module tb();
                reg [3:0] r1_r,r2_r;
                reg  ci_r;
                wire [3:0] result_w;
                wire  carry_w; // Drive the inputs
                initial begin
                                r1_r = 0;  r2_r = 0;  ci_r = 0;  #10 r1_r = 10;  #10 r2_r = 2;  #10 ci_r = 1;
                                #10 $display("+-----------------------------------------------------+");
                                $finish;
                end // Connect the lower module
                adderHier U (result_w,carry_w,r1_r,r2_r,ci_r);                  // Hier demo here
                initial begin
                                $display("+-----------------------------------------------------+");
                                $display("|  r1  |  r2  |  ci  | u0.sum | u1.sum | u2.sum | u3.sum |");
                                $display("+-----------------------------------------------------+");
                $monitor("|  %h   |  %h   |  %h   |   %h   |  %h   |  %h   |   %h   |",  r1_r,r2_r,ci_r, tb.U.u0.sum_o, tb.U.u1.sum_o,
                                tb.U.u2.sum_o, tb.U.u3.sum_o);
                end
endmodule
```

$display, $write, $monitor and $strobe

- $display – system task

$display("<format>", exp1, exp2, ...); // formatted write to display

format indication

%b %B binary
%c %C character (low 8 bits)
%d %D decimal
%0d for minimum width field
%e %E E format floating point %15.7E
%f %F F format floating point %3.2F
//3-total 3 bits; precision 2 bits, 2.71828->2.71
%g %G G general format floating point
%h %H hexadecimal
%l %L library binding information
%m %M hierarchical name, no expression
%o %O octal
%s %S string, 8 bits per character, 2´h00 does not print
%t %T simulation time, expression is $time
%u %U unformatted two value data 0 and 1
%v %V net signal strength
%z %Z unformatted four value data 0, 1, x, z

$display, $write, $monitor and $strobe

- $display – system task

escape sequences, quoted characters in strings

\n	newline
\t	tab
\\	backslash
\"	quote
\ddd	octal for other character in ASCII code
%%	percent

```
Question: a=10,   $display("%%%0d", a);        //Ans: %10
                  $display("%%%%0d", a);       //Ans: %%0d    10
                  $display("%%%%%0d", a);      //Ans: %%10
```

- Difference between $display and $write
  - $display - automatic insertion of newline, and one display statement is just to print one time on the monitor
  - $write – same as $display, without newline

$display, $write, $monitor and $strobe

- Difference between $display and $monitor
  - $monitor - automatic insertion of newline, and one display statement is to print one times if values in the statement change.

  Example:

  $monitor(“Time=%g, a=%d”, $time, a);

  // If “a” changes, one more line printout is shown

- Difference between $display and $strobe

  According to scheduling semantics of verilog,

  $display executes before the nonblocking statements update LHS. Therefore if $display contains LHS variable of nonblocking assignment, the results are not proper.

  The $strobe command shows updated values at the end of the time step after all other commands, including nonblocking assignments, have completed.

$display, $write, $monitor and $strobe

- Difference between $display and $strobe

```
module testStrobe;
        reg             a;
        initial begin
                a = 0;
                a <= 1;
                $display(" $display a=%b", a);
                $strobe (" $strobe a=%b", a);
                #1 $display("#1 $display a=%b", a);
                #1 $finish;
        end
endmodule
```

Results:

$display a=0

$strobe a=1

#1 $display a=1