

Verilog Syntax

3.1 Lexical Conventions

3.1.1 Whitespace [chap 4, p.66]

Blank spaces(**\b**) 、 tabs(**\t**) 、 newlines(**\n**)

3.1.2 Comments [chap 4, p.62]

```
A = b && c ; // This is a one-line comment
             /*This is a multiple line
              Comment*/
```

```
/*This is /* an illegal */ comment */
```

```
/* This is //a legal comment */
```

3.1.4 Number Specification

Sized numbers

1. Sized numbers are represented as **<size>'<base format> <number>**

Decimal (**'d** or **'D**) , **hexadecimal**(**'h** or **'H**) , **binary**(**'b** or **'B**) , **octal**(**'o** or **'O**)

[chap 4, p. 66.73. 77]

```
4'b1111 // This is a 4-bit    binary number
12'habc  // This is a 12-bit   hexadecimal number 1010-1011-1100
16'd255  // This is a 16-bit   decimal number 0000-0000-1111-1111
```

2. Unsized numbers

```
23456    // This a 32-bit    decimal number by default
`hc3      // This is a 32-bit  hexadecimal number
`o21      // This is a 32-bit  octal number
```

3. X or Z values (Chapt. 2, p. 24)

unknown by an **x** , **high impedance** is denoted by **z**

```
12'h13x // This is a 12-bit hex number ; 4 least significant bits unknown
```

6'hx // This is a 6-bit hex number ; **6 bits unknown**
32'bz // This is a **32-bit high impedance number**

4. Negative numbers

-8'd3 // 8-bit negative number stored as 2's complement of 3
-8'sd3 // Used for performing signed integer math
4'd-2 // **Illegal specification**

5. Underscore characters and question marks

12'b1111_0000_1010 // Use of underline characters for readability
4'b10?? // Equivalent of a **4'b10zz**

3.1.5 Strings [chap 4, p.68]

"Hello Verilog World" // is a string
"a / b" // is a string

3.2.1 Value Set [chap 4, p.77]

Table 3-1 Value Levels

Value Level	Condition in Hardware Circuits
0	Logic zero, false condition
1	Logic one, true condition
x	Unknown logic value
z	High impedance, floating state

3.2.2 Nets [chap 4, p.75]

wire a; // Declare net a for the above circuit
wire b,c; // Declare two wires b, c for the above circuit
wire d = 1'b0; // Net d is fixed to logic value 0 at declaration

3.2.3 Registers [chap 4, p.73]

Example 3-1

reg reset;
initial
begin

reset = 1'b1; // initialize reset to 1 to reset the digital circuit
#100 reset = 1'b0; // after 100 time units reset is deasserted

Example of Register

// declare a variable reset that can hold its value
// this construct will be discussed later

end

3.2.4 Vectors [chap 4, p.73]

```
wire a; // scalar net variable, default
wire [7:0] bus; // 8-bit bus
wire [31:0] busA, busB, busC; // 3 buses of 32-bit width
reg clock; // scalar register, default
reg [0:40] virtual_addr; // Vector register, virtual address 41 bits
wide
```

Vector Part Select [chap 4, p.75]

```
busA[7] // bit # 7 of vector busA
bus [2:0] // 3 least significant bits of vector bus,
// using bus [0:2] is illegal because the significant bit should
// always be on the left of a range specification
virtual_addr [0:1] // Two most significant bits of vector virtual_addr
```

Variable Vector Part Select

[<starting_bit>+:width] – part-select increments from starting bit
[<starting_bit> -:width] – part-select decrements from strating bit

```
reg [255:0] data1; // Little endian notation
reg [0:255] data2; // Big endian notation
reg [7:0] byte;
```

//Using a variable part select,one can choose parts

```
byte = data1[31 -:8]; // starting bit = 31,width = 8 =>data[31:24]
byte = data1[24+:8]; // starting bit = 24,width = 8 =>data[31:24]
byte = data2[31 -:8]; // starting bit = 31,width = 8 =>data[24:31]
byte = data2[24+:8]; // starting bit = 24,width = 8 =>data[24:31]
```

//The starting bit can also be a variable. The width has

//to be constant. Therefore, one can use the variable part select

//in a loop to select all bytes of the vector

```
for(j=0;j<=31;j=j+1)
```

```
    byte = data1[(j*8)+:8]; // Sequence is [7:0],[15:8].....[255:248]
```

//Can initialize a part of the vector

```
data1[(byteNum*8)+:8] = 8'b0; // If byteNum = 1,clear 8 bits [15:8]
```

3.2.5 Integer, Real, and Time Register Data Types

Integer

```
integer counter;    // general purpose variable used as a counter
initial
    counter = -1;    // A negative one is stored in the counter
```


Real

```
real delta;          // Define a real variable called delta
initial
    begin
        delta = 4e10;    // delta is assigned in scientific notation
        delta = 2.13;    // delta is assigned a value 2.13
    end
integer i;           // Define an integer i
initial
    i = delta;          // i gets the value 2 (rounded value of 2.13)
```

Time

```
time save_sim_time;    // Define a time variable save_sim_time
initial
    save_sim_time = $time;    // Save the current simulation time
```

3.2.6 Arrays



```
integer count [0:7];    // An array of 8 count variables
reg bool [31:0];        // Array of 32 one-bit Boolean register variables
time chk_point [1:100]; // Array of 100 time checkpoint variables
reg [4:0] port_id [0:7]; // Array of 8 port_ids; each port_id is 5 bits wide
integer matrix [4:0][0:255]; // Two dimensional array of integers
reg [63:0] array_4d [15:0][7:0][7:0][255:0]; // Four dimensional array
wire [7:0] w_array2 [5:0]; // Declare an array of 8 bit vector wire
wire w_array1 [7:0][5:0]; // Declare an array of single bit wires

count [5] = 0;          // Reset 5th element of array of count variables
chk_point [100] = 0;    // Reset 100th time check point value
port_id [3] = 0;        // Reset 3rd element (a 5-bit value) of port_id array
```

```

matrix [1][0] = 33559;           // Set value of element indexed by [1][0] to 33559
array_4d [0][0][0][0][15:0] = 0; // Clear bits 15:0 of the register
                                   // accessed by indices [0][0][0][0]

port_id = 0;                     // Illegal syntax – Attempt to write the entire array
matrix [1] = 0;                 // Illegal syntax – Attempt to write [1][0].....[1][255]

```

3.2.7 Memories

```

reg mem1bit [0:1023];           // Memory mem1bit with 1K 1-bit words
reg [7:0] membyte [0:1023];     // Memory membyte with 1K 8-bit words(bytes)
membyte [511]                  // Fetches 1 byte word whose address is 511

```

3.2.8 Parameters

```

parameter port_id = 5;          // Defines a constant port_id
parameter cache_line_width = 256; // Constant defines width of cache line
parameter signed [15:0] WIDTH;  // Fixed sign and range for parameter
                                   // WIDTH

```

3.2.9 Strings

Each character takes 8 bits(1 byte)

Table 3-3 Special Characters

Escaped Characters	Character Displayed
<code>\n</code>	newline
<code>\t</code>	tab
<code>%%</code>	%
<code>\\</code>	\
<code>\”</code>	“
<code>\ooo</code>	Character written in 1-3 octal digits

3.3 System Tasks and Compiler Directives

3.3.1 System Tasks: user call for certain routine operation.

`$<keyword>`

1. displaying information Usage: `$display(p1, p2, p3,..., pn);`

Table 3-4 String Format Specifications (p1, p2, p3,..., pn)

Format	Display
%d or %D	Display variable in decimal
%b or %B	Display variable in binary
%s or %S	Display string
%h or %H	Display variable in hex
%c or %C	Display ASCII character
%m or %M	Display hierarchical name(no argument required)
%v or %V	Display strength
%o or %O	Display variable in octal
%t or %T	Display in current time format
%e or %E	Display real number in scientific format(e.g.,3e10)
%f or %F	Display real number in decimal format(e.g.,2.13)
%g or %G	Display real number in scientific or decimal,whichever is shorter

2, Monitoring information

Example 3-5

Monitor Statement

// Monitor time and value of the signals clock and reset

// Clock toggles every 5 time units and reset goes down at 10 time units

initial

begin

\$monitor(\$time,

“Value of signals clock = %b reset = %b”, clock, reset);

end

Partial output of the monitor statement:

-- 0 Value of signals clock = 0 reset = 1

-- 5 Value of signals clock = 1 reset = 1

-- 10 Value of signals clock = 0 reset = 0

Stopping and finishing in a simulation

Usage: \$stop, \$finish

Example 3-6

Stop and Finish Tasks

// Stop at time 100 in the simulation and examine the results

// Finish the simulation at time 1000

```

initial          // to be explained later.time = 0
begin
clock = 0;
reset = 1;
#100 $stop;      // This will suspend the simulation at time = 100
#900 $finish;    // This will terminate the simulation at time = 1000
end

```

3.3.2 Compiler Directives

```

`define
#define

```

Example 3-7 *`define Directive*

```

// define a text macro that defines default word size
// Used as `WORD_SIZE in the code
`define WORD_SIZE 32

// define an alias.A $stop will be substituted wherever `S appears
`define S $stop;

// define a frequently used text string
`define WORD_REG reg [31:0]
// you can then define a 32-bit register as `WORD_REG reg32;

`include

```

Example 3-8 *`include Directive*

```

// Include the file header.v,which contains declarations in the
// main verilog file design.v
`include header.v
...
...
<Verilog code in file design.v>
...

```

4.2.2 Port Declaration

Verilog Keyword	Type of Port
input	Input port
output	Output port
inout	Bidirectional port

4.2.3 Port Connection Rules

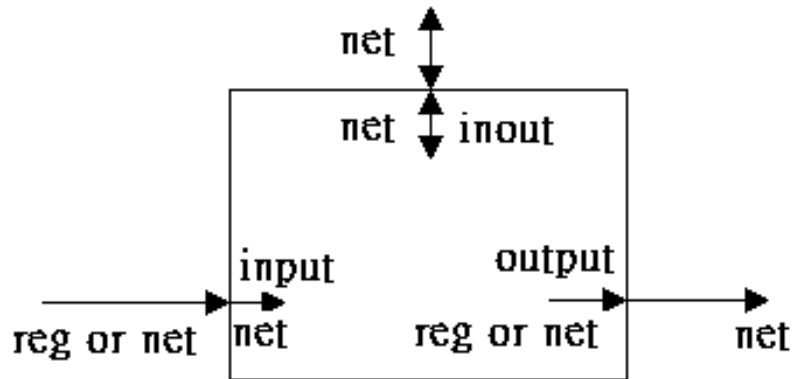


Figure 4-4 Port Connection Rules

4.2.4 Connecting Ports to External Signals

Connecting by ordered list

Example 4-7

Connection by Ordered List

Module Top;

// Declare connection variables

reg [3:0] A,B;

reg C_IN;

wire [3:0] SUM;

wire C_OUT;

//Instantiate fulladd4,call it fa_ordered

//Signals are connected to ports in order(by position)

Fulladd4 fa_ordered(SUM,C_OUT,A,B,C_IN);

...

<stimulus>

...

endmodule

Module fulladd4(sum,c_out,a,b,c_in);

Output [3:0]sum;

Output c_out;

Input [3:0]a,b;

Input c_in;

...

<module internals>

...

endmodule