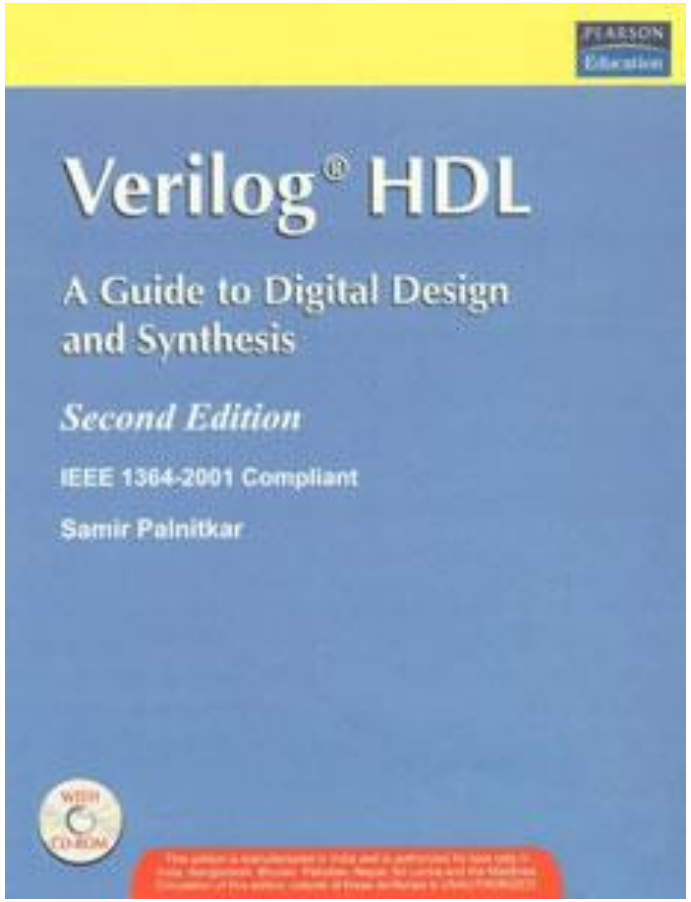
Verilog HDL

BASIC CONCEPTS

Reference

Verilog HDL: A Guide to Digital Design and Synthesis, 2e, Samir Palnitkar

Chapters 3



Outline

- **Lexical conventions**
- Data types
- System tasks and compiler directives

Lexical Conventions

Lexical Conventions

- The basic lexical conventions used by Verilog HDL are similar to those in the C.
- Verilog contains a stream of tokens.
- Tokens can be comments, delimiters, numbers, strings, identifiers, and keywords.
- Verilog HDL is case-sensitive.
- All keywords are in lowercase.

Whitespace

- Blank spaces, tabs and newlines comprise the whitespace.
- Whitespace is ignored by Verilog except when it separates tokens.
- Whitespace is not ignored in strings.

Comments

- Comments can be inserted in the code for readability and documentation.
- A one-line comment starts with "//".
- A multiple-line comment starts with "/*" and ends with "*/".
- Multiple-line comments cannot be nested. However, one-line comments can be embedded in multiple-line comments.

```
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 */
```

Operators

- Operators are of three types: unary, binary, and ternary.
- Unary operators : precede the operand.
- Binary operators : appear between two operands.
- Ternary operators : have two separate operators that separate three operands.

```
a = ~ b; /* ~ is a unary operator. b is the operand */
a = b && c; /* && is a binary operator. b and c are
operands */
a = b ? c : d; /* ?: is a ternary operator. b, c and d
are operands */
```

Operators

Arithmetic Operators	+, -, *, /, %
Relational Operators	<, <=, >, >=
Logical Equality Operators	==, !=
Case Equality Operators	===, !==
Logical Operators	!, &&, ||
Bit-Wise Operators	~, &, |, ^(xor), ~^(xnor)
Unary Reduction Operators	&, ~&, |, ~|, ^, ~^
Shift Operators	>>, <<
Conditional Operators	? :
Concatenation Operator	{ }
Replication Operator	{ { } }

Number Specification

Sized numbers : <size> '<base format> <number>

<size> : Number of bits in a number in decimal

'<base format> : decimal ('d or 'D), hexadecimal ('h or 'H), binary ('b or 'B) and octal ('o or 'O)

<number> : digits from 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a (or A), b or (B), c (or C), d (or D), e (or E), f (or F).

```
4'b1111 // This is a 4-bit binary number
12'habc // This is a 12-bit hexadecimal number
16'd255 // This is a 16-bit decimal number
```

Number Specification

Unsized numbers : '<base format> <number>

Numbers written without <size> have default number of bits that is simulator and machine specific (must be at least 32)

Numbers written without <base format> are decimal numbers by default.

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

Number Specification

X or Z values :

- An unknown value is denoted by an x.
- High impedance (floating state) value is denoted by z.
- An x or z sets four bits for a number in the hexadecimal base, three bits for a number in the octal base, and one bit for a number in the binary base.

```
4'b100z /* This is a 4-bit binary number; last bit is high
impedance number */

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

Number Specification

Underscore characters and question marks :

- An underscore "_" is allowed anywhere in a number except the first character.
- Underscore characters would improve readability of numbers and are ignored by Verilog
- A question mark "?" is the Verilog HDL alternative for z

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

Strings

- A string is a sequence of characters that are enclosed by double quotes.
- Strings are treated as a sequence of one-byte ASCII values.

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

Identifiers and Keywords

Keywords :

- Keywords are special identifiers reserved to define the language constructs.
- Keywords are in lowercase.
- Eg : and, if, for, begin, end …. (See Appendix C of the book for all keywords)

Identifiers and Keywords

Identifiers :

- Identifiers are names given to objects so that they can be referenced in the design.
- Identifiers are made up of alphanumeric characters, the underscore (_), or the dollar sign ($).
- Identifiers are case sensitive.
- Identifiers start with an alphabetic character or an underscore.

Identifiers and Keywords

```
reg value; // reg is a keyword; value is an identifier
input clk; // input is a keyword, clk is an identifier
```

Outline

- Lexical conventions
- **Data types**
- System tasks and compiler directives

Data Types

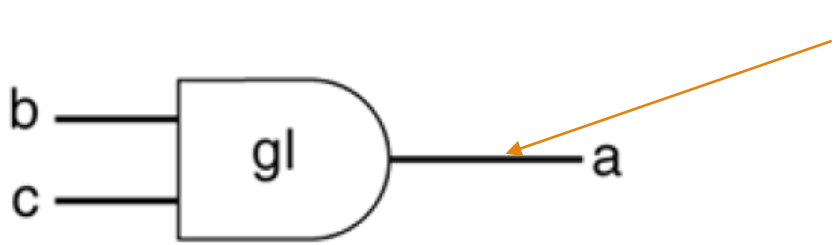
Value Set

Verilog supports four values to model the functionality of real hardware.

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

Nets (wires)

- Nets represent connections between hardware elements.
- Nets are declared with the keyword **wire**.
- Nets get the output value of their drivers. If a net has no driver, it gets the value z (default value).



Net **a** is driven by the output of the gate **gl**.

```
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.
```

Registers

- Registers represent data storage elements.
- Registers retain value until another value is placed onto them.
- A variable that can hold a value.
- Declared by keyword **reg**.

```
reg reset; // declare a variable reset that can hold its value
initial // this construct will be discussed later
begin
  reset = 1'b1; //initialize reset to 1 to reset the digital circuit.
  #100 reset = 1'b0; // after 100 time units reset is deasserted.
end
```

Don’t confuse the term registers in Verilog with hardware registers built from flipflops in real circuits.

Vectors

- Nets or reg data types can be declared as vectors (multiple bit widths).
- If bit width is not specified, the default is 1-bit.

```
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
```

Vector part select

```
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
```

It is possible to address bits or parts of vectors declared above

```
busA[7] // bit # 7 of vector busA
bus[2:0] // Three least significant bits of vector bus,
```

Integer, Real, and Time (Register Data Types)

Integer

- The default width for an integer is the host-machine word size, but is at least 32 bits.
- Integers can store signed values as well while variables declared as reg could only store unsigned.

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

Integer, Real, and Time (Register Data Types)

Real

- Specified in decimal notation (e.g., 3.14) or in scientific notation (e.g., 3e6, which is 3 x 10^6)

```
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)
```

Integer, Real, and Time (Register Data Types)

Time

- Verilog simulation is done with respect to simulation time.
- Time register datatype is used in Verilog to store simulation time.
- Simulation time is measured in terms of simulation seconds.

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

Strings

- Strings can be stored in reg.
- Each character in the string takes up 8 bits (1 byte).
- If **register width > string size** : bits to the left of the string is filled with zeros.
- If **register width < string size** : leftmost bits in the string are truncated.
- Special characters such as \n, \t are allowed.

```
reg [8*18:1] string_value; // Declare a variable that is 18 bytes wide
initial
    string_value = "Hello Verilog World"; // String can be stored
// in variable
```

Outline

- Lexical conventions
- Data types
- **System tasks and compiler directives**

System Tasks and Compiler Directives

System Tasks

- All system tasks appear in the form **$<keyword>**.
- Operations such as displaying on the screen, monitoring values of nets, stopping, and finishing are done by system tasks.
- Some examples are:

$bitstoreal	$countdrivers	$display	$fclose
$fdisplay	$fmonitor	$fopen	$fstrobe
$fwrite	$finish	$getpattern	$history
$incsave	$input	$itor	$key
$list	$log	$monitor	$monitoroff
$monitoron	$nokey	$time	

System Task : Displaying Information

- $display is the main system task for displaying values of variables or strings or expressions.
- The format of $display is very similar to printf in C.
- A $display inserts a newline at the end of the string by default.
- A $display without any arguments produces a newline.

- Usage: $display(p1, p2, p3,....., pn);

p1, p2, p3,..., pn can be quoted strings or variables or expressions

$display : Format Specification

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

$display : Examples

```
//Display the string in quotes
$display("Hello Verilog World");
-- Hello Verilog World


//Display value of current simulation time 230
$display($time);
-- 230
```

$display : Examples

```
// Display value of 41-bit virtual address 1fe0000001c at time 200
reg [40:0] virtual_addr;
$display("At time %d virtual address is %h", $time, virtual_addr);
-- At time 200 virtual address is 1fe0000001c


//Display value of port_id 5 in binary
reg [4:0] port_id;
$display("ID of the port is %b", port_id);
-- ID of the port is 00101
```

Compiler Directives

- All compiler directives are defined by using the **'<keyword>** construct.
- We will focus on two important directives only; 'define and 'include.

'define

- similar to the #define construct in C.

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

Compiler Directives

`include

- Similarly to the #include in C.

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

END OF CHAPTER 3