

# Verilog HDL

# Topics

- Introduction to Verilog HDL
- Data types
- Operators
- Verilog code structures
- Verilog modelling styles and examples

# Verilog evolution

- Verilog was designed in early 1984 by Gateway Design Automation. Initially the original language was used as a simulation and verification tool.
- Gateway Design Automation and its Verilog-based tools were later acquired by Cadence Design System. Since then, Cadence has been a strong force behind popularizing the Verilog hardware description language.
- In 1993, efforts for standardization of this language started. Verilog became the IEEE standard, IEEE Std. 1364-1995, in 1995.
- A new version of Verilog was approved by IEEE in 2001. This version that is referred to as Verilog-2001 is the present standard used by most users and tool developers

# Code format

- Verilog code is free format, with spaces and new lines serving as separators.
- Source text is **case-sensitive**, i.e., identifiers using lowercase or uppercase characters are distinguished from each other.
- The language uses certain keywords, all of which must use lowercase characters.
- Comments may appear anywhere in a Verilog source text. A comment designator starting with `//` makes the rest of the line, up to a new-line character, a comment. The symbols `/*` and `*/` bracket a section of code as a comment, and they go across new-line characters.

# Code format

- A **module** is the main structure for definition of hardware components and testbenches in verilog.
- Modules begin with the module keyword and end with **endmodule**. Immediately following the module keyword, port list of the module appears enclosed in parenthesis.
- Declaration of mode, type, and size of ports can either appear in the port list or as separate declarations.

---

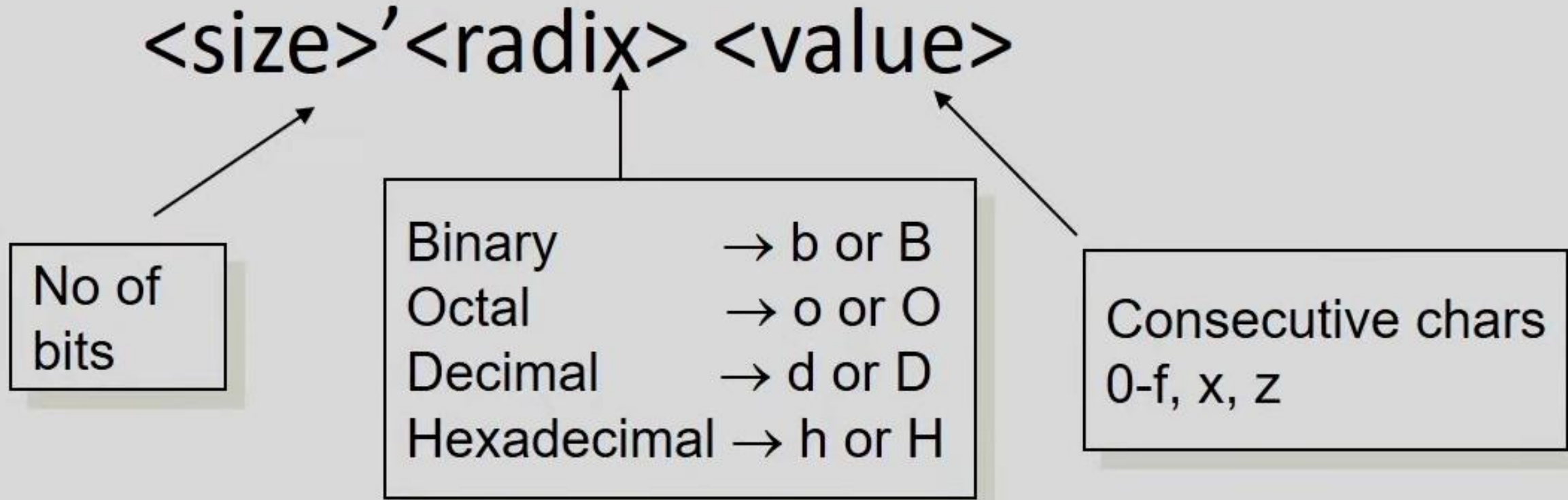
```
module name (ports or ports and their declarations);  
    port declarations if not in the header;  
    other declarations;  
    . . .  
    statements  
    . . .  
endmodule
```

---

# Number Representation in Verilog

`<size>'<radix> <value>`

No of  
bits



|             |          |
|-------------|----------|
| Binary      | → b or B |
| Octal       | → o or O |
| Decimal     | → d or D |
| Hexadecimal | → h or H |

Consecutive chars  
0-f, x, z

# Wires and variables

Verilog has two main data types, **net** and **reg**. A **net** represents a wire driven by a hardware structure or output of a gate. A **reg** represents a variable that can be assigned values in a behavioral description of a component in a Verilog procedural block.

# Verilog Operators

|                                |     |          |    |    |    |    |    |
|--------------------------------|-----|----------|----|----|----|----|----|
| <b>Bitwise Operators</b>       | &   |          | ^  | ~  | ~^ | ^~ |    |
| <b>Reduction Operators</b>     | &   | ~&       |    | ~  | ^  | ~^ | ^~ |
| <b>Arithmetic Operators</b>    | +   | -        | *  | /  | %  |    |    |
| <b>Logical Operators</b>       | &&  |          | !  |    |    |    |    |
| <b>Compare Operators</b>       | <   | >        | <= | >= | == |    |    |
| <b>Shift Operators</b>         | >>  | <<       |    |    |    |    |    |
| <b>Concatenation Operators</b> | { } | { n{ } } |    |    |    |    |    |
| <b>Conditional Operator</b>    | ?:  |          |    |    |    |    |    |



# Arithmetic Operators

- $*$   $\rightarrow$  multiply
- $/$   $\rightarrow$  divide
- $+$   $\rightarrow$  add
- $-$   $\rightarrow$  subtract
- $\%$   $\rightarrow$  modulus

## Some Examples

$$5 + 10 = 15$$

$$5 - 10 = -5$$

$$10 - 5 = 5$$

$$10 * 5 = 50$$

$$10 / 5 = 2$$

$$10 / -5 = -2$$

$$10 \% 3 = 1$$

# Logical Operators

- `& &` → logical AND
- `| |` → logical OR
- `!` → logical NOT
- Operands evaluated to ONE bit value: *0*, *1* or *x*
- Result is ONE bit value: *0*, *1* or *x*

`A = 1;`

`B = 0;`

`C = x;`

`A & & B → 1 & & 0 → 0`

`A | | !B → 1 | | 1 → 1`

`C | | B → x | | 0 → x`



but `C & & B = 0`

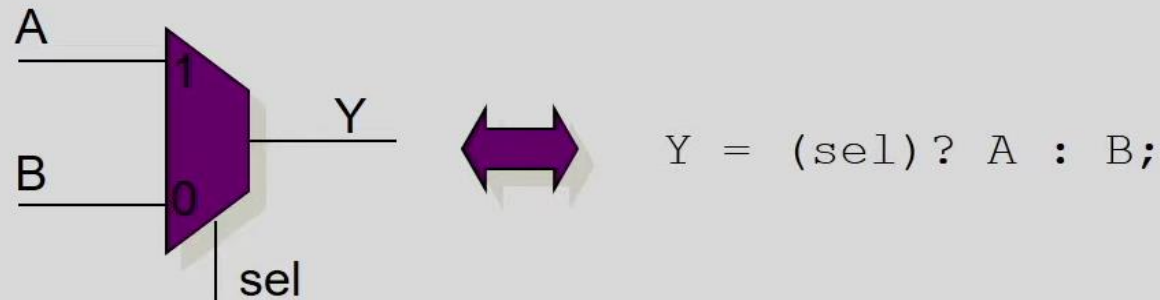
# Shift, Conditional Operator

- `>>` → shift right
- `<<` → shift left

- `a = 4'b1010;`

`d = a >> 2;` // `d = 0010`, `c = a << 1;` // `c = 0100`

- `cond_expr ? true_expr : false_expr`



# Keywords

- **Note : All keywords are defined in lower case**
- **Some examples of Keywords are:**
- module, endmodule
- input, output, inout
- reg
- parameter
- begin, end

```
module memory ("ports");  
input clock;  
input address;  
output reg read_enable;  
inout data;  
parameter DATA_WIDTH = 8 ;  
parameter ADDR_WIDTH = 8 ;  
parameter RAM_DEPTH = 1 << ADDR_WIDTH;  
endmodule
```



Multiple statements must be enclosed

---

```
module mux2_1 (input [3:0] i0, i1, input s, output [3:0] y
               );
    assign y = s ? i1 : i0;
endmodule
```

---

**Figure 3.31 A 2-to-1 Mux using Condition Operator**

## Logic Value System

Verilog uses a 4-value logic value system. Values in this system are **0**, **1**, **Z**, and **X**. Value **0** is for logical **0** which in most cases represents a path to ground (Gnd). Value **1** is logical **1** and it represents a path to supply (Vdd). Value **Z** is for float, and **X** is used for un-initialized, un-defined, un-driven, unknown, and value conflicts. Values **Z** and **X** are used for wired-logic, busses, initialization values, tri-state structures, and switch-level logic.

logic\_ckt.v - D:/codes

```
1  // Verilog Code Structure
2
3  module logic_ckt(a,b,c)
4  input a,b;
5  output c;
6  wire s1,s2;
7  reg r1,r2;
8  assign s1 = blah...blah;
9  assign s2 = blah...blah;
10 always @(a or b )begin
11 r1= a & b;
12 end
13 endmodule
```

mux2\_1.v - D:/codes

```
1 // Example of a verilog code
2
3 module mux2_1(i0,i1,s,y);
4     input [3:0] i0,i1;
5     input s;
6     output [3:0] y;
7     assign y=s?i1:i0;
8 endmodule
```



# Verilog Hardware Modelling Styles

- Structural → Gate-level or Switch-level (Transistor)
- Dataflow
- Behavioral

majority\_ckt1.v \* - D:/codes

```
1 // Majority circuit using wire
2
3 module majority_ckt1(input a,b,c,output y);
4 wire s1,s2,s3,s4;
5 assign s1=a&b;
6 assign s2=b&c;
7 assign s3=c&a;
8 assign s4=s1|s2;
9 assign y=s3|s4;
10 endmodule
```

majority\_ckt2.v - D:/codes

```
1 // Majority circuit using reg
2
3 module majority_ckt2(input a,b,c,output reg y);
4 always @(a or b or c) begin
5 y=(a&b)|(b&c)|(c&a);
6 end
7 endmodule
```

halfadder\_dataflow.v \* - D:/codes

```
1 // Dataflow design
2
3 module halfadder_dataflow(a,b,sum,carry);
4 input a,b;
5 output sum,carry;
6 assign sum=a^b;
7 assign carry=a&b;
8 endmodule
```

halfadder\_structural.v \* - D:/codes

```
1 // Structural design
2
3 module halfadder_structural(a,b,sum,carry);
4 input a,b;
5 output sum,carry;
6 xor g1(sum,a,b);
7 and g2(carry,a,b);
8 endmodule
```

halfadder\_behavioral.v - D:/codes

1 *// Behavioral design*

2



3 **module** halfadder\_behavioral (a,b,sum,carry) ;

4 **input** a,b;

5 **output** sum,carry;

6 **assign** {carry,sum}=a+b;

7 **endmodule**

HA.v \* - D:/codes

```
1 module HA(a,b,s,c);
2   input a,b;
3   output s,c;
4   xor g1(s,a,b);
5   and g2(c,a,b);
6 endmodule
```

FA.v - C:/Users/samui/Dropbox/Courses/VHDL-Verilog/codes/Verilog/codes

```
1 module FA(a,b,cin,s,cout);
2   input a,b,cin;
3   output s,cout;
4   wire s1,c1,c2;
5   // instantiate HAs
6   HA m1(a,b,s1,c1);
7   HA m2(s1,cin,s,c2);
8   or o1(c,c1,c2);
9 endmodule
```

4\_bit\_RCA.v \* - D:/codes

```
1 module 4bit_RCA(x,y,s,co);
2   input [3:0] x,y; // Two 4-bit inputs
3   output [3:0] s;
4   output co;
5   wire w1,w2,w3;
6   // instantiating 4 1-bit full adders in Verilog
7   FA u1(x[0],y[0],1'b0,s[0],w1);
8   FA u2(x[1],y[1],w1,s[1],w2);
9   FA u3(x[2],y[2],w2,s[2],w3);
10  FA u4(x[3],y[3],w3,s[3],co);
11 endmodule
```