Verilog HDL:
Language Essentials

Pravin Zode

Outline

- Lexical Conventions
- Comments
- Number Specifications
- Value sets and Strengths
- Data Types

Lexical Conventions

- Lexical conventions in Verilog HDL are similar to the C programming language.
- Verilog contains a stream of tokens such as:
  - Comments
  - Delimiters
  - Numbers
  - Strings
  - Identifiers
  - Keywords
- Verilog HDL is a case-sensitive language.
- All keywords in Verilog HDL are written in lowercase.

White Space

- Whitespace in Verilog includes
  - Blank spaces (\b)
  - Tabs (\t)
  - Newlines (\n)
- Whitespace is ignored by Verilog except when separating tokens
- Whitespace within strings is not ignored.

Comments

- Comments improve code readability and documentation
- Two types of comments in Verilog:
  - One-line comments: Start with // and extend to the end of the line.
  - Multiple-line comments: Start with /* and end with */
- Multiple-line comments cannot be nested
- One-line comments can be embedded within multiple-line comments

Operators

- Operators in Verilog are classified into three types:
  - Unary operators: Precede the operand
  - Binary operators: Appear between two operands
  - Ternary operators: Use two separate operators to work with 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
```

Number Specification

- Verilog supports two types of number specification
  - **Sized Numbers**: Size is explicitly specified, e.g., 4'b1010 (4-bit binary), 8'hA3 (8-bit hexadecimal)
  - **Unsized Numbers**: Size is not explicitly mentioned
  - Default size is 32 bits in most cases, e.g. 123 (interpreted as a 32-bit decimal number)

Value Set

Value	Description
0	A logic zero, or false condition.
1	A logic one, or true condition.
x or X	Unknown or uninitialized.
z or Z	High impedance, tri-stated, or floating.

Strength

The strengths are used to resolve the value of a signal when it is driven by multiple sources

Strength	Description	Strength level
supply1	Supply drive for VCC	**7**
supply0	Supply drive for VSS, or GND	**7**
strong1	Strong drive to logic one	**6**
strong0	Strong drive to logic zero	**6**
pull1	Medium drive to logic one	**5**
pull0	Medium drive to logic zero	**5**
large	Large capacitive	**4**
weak1	Weak drive to logic one	**3**
weak0	Weak drive to logic zero	**3**
medium	Medium capacitive	**2**
small	Small capacitive	**1**
highz1	High impedance with weak pull-up to logic one	**0**
highz0	High impedance with weak pull-down to logic zero	**0**

Net Data Type

- Every signal in Verilog must be associated with a data type.
- Net data type models an interconnection (net) between components.
- Nets can take values: 0, 1, X (unknown), and Z (high impedance).
- A signal with a net data type must be driven at all times.
- The value of a net updates when the driver value changes.
- The most common synthesizable net data type in Verilog is wire

Net Data Type

Type	Description
wire	A simple connection between components.
wor	Wired-OR. If multiple drivers, their values are OR'd together.
wand	Wired-AND'd. If multiple drivers, their values are AND'd together.
supply0	Used to model the VSS, (GND), power supply (supply strength inherent).
supply1	Used to model the VCC power supply (supply strength inherent).
tri	Identical to wire. Used for readability for a net driven by multiple sources.
trior	Identical to wor. Used for readability for nets driven by multiple sources.
triand	Identical to wand. Used for readability for nets driven by multiple sources.
tri1	Pulls up to logic one when tri-stated.
tri0	Pulls down to logic zero when tri-stated.
trireg	Holds last value when tri-stated (capacitance strength inherent).

Variable Data Types

- Variable data types in Verilog model storage.
- They can take values: 0, 1, X (unknown), and Z (high impedance).
- Unlike nets, variable data types do not have an associated strength.
- They retain their assigned value until the next assignment.

Variable Data Types

Type	Description
reg	A variable that models logic storage. Can take on values 0, 1, X, and Z.
integer	A 32-bit, 2's complement variable representing whole numbers between $2,147,483,648_{10}$ and +2,147,483,647.
real	A 64-bit, floating point variable representing real numbers between $-(2.2 \times 10^{-308})10$ and $+(2.2 \times 10^{308})_{10}$.
time	An unsigned, 64-bit variable taking on values from 0_{10} to $+(9.2 \times 10^{18})$.
realtime	Same as time. Just used for readability.

Vector Data Types

- A vector is a one-dimensional array of elements.
- All net data types can be used to form vectors.
- The variable type reg can also be used to create vectors

<type> [<MSB_index> : <LSB_index>] vector_name

```
wire [7:0] Sum;     // This defines an 8-bit vector called "Sum" of type wire. The
                    // MSB is given the index 7 while the LSB is given the index 0.

reg [15:0] Q;       // This defines a 16-bit vector called "Q" of type reg.
```

Individual bit addressing

```
Sum[0];         // This is the least significant bit of the vector "Sum" defined above.
Q[15:8];        // This is the upper 8-bits of the 16-bit vector "Q" defined above.
```

Arrays

- An array is a multidimensional collection of elements
- It can be thought of as a "vector of vectors.“
- Vectors within the array all have the same dimensions.
- To declare an array:
  - First, define the element type and dimensions. Then, specify the array name and its dimensions.

Arrays

```
<element_type>  [<MSB_index>:<LSB_index>]  array_name  [<array_start_index>:
<array_end_index>];
```

Example:

```
reg[7:0] Mem[0:4095];  // Defines an array of 4096, 8-bit vectors of type reg
integer A[1:100];      // Defines an array of 100 integers.
```

```
Mem[2];          // This is the 3rd element within the array named "Mem".
                 // This syntax represents an 8-bit vector of type reg.

Mem[2][7];       // This is the MSB of the 3rd element within the array named "Mem".
                 // This syntax represents a single bit of type reg.

A[2];            // This is the 2nd element within the array named "A". Recall
                 //  that A was declared with a starting index of 1.
                 // This syntax represents a 32-bit, signed integer.
```

Expressing Number with different bases

Syntax	Description
'b	Unsigned binary.
'o	Unsigned octal.
'd	Unsigned decimal.
'h	Unsigned hexadecimal.
'sb	Signed binary.
'so	Signed octal.
'sd	Signed decimal.
'sh	Signed hexadecimal.

```
<size_in_bits>'<base><value>
```

- If a number is entered in Verilog without specifying its syntax, it is treated as an integer
- Verilog supports different number bases (binary, octal, decimal, hexadecimal)
- The underscore (_) can be inserted between numerals to improve readability.

Expressing Number with different bases

```
10              // This is treated as decimal 10, which is a 32-bit signed vector.
4'b1111         // A 4-bit number with the value 1111₂.
8'b1011_0000    // An 8-bit number with the value 10110000₂.
8'hFF           // An 8-bit number with the value 11111111₂.
8'hff           // An 8-bit number with the value 11111111₂.
6'hA            // A 6-bit number with the value 001010₂. Note that leading zeros
                //  were added to make the value 6-bits.
8'd7            // An 8-bit number with the value 00000111₂.
32'd0           // A 32-bit number with the value 0000_0000₁₆.
'b1111          // A 32-bit number with the value 0000_000F₁₆.
8'bZ            // An 8-bit number with the value ZZZZ_ZZZZ.
```

Assigning Between Different Types

- Verilog is a weakly typed (loosely typed) language, allowing assignments between different data types.
- In contrast, strongly typed languages (e.g., VHDL) only allow assignments between like types.
- **Reason:** Verilog treats all data types as groups of bits, making type conversion flexible.
- When assigning between different types: Verilog automatically truncates or adds leading bits as needed.
- Example scenario: Assume ABC_TB is declared as reg[2:0].

```
ABC_TB = 2'b00;     // ABC_TB will be assigned 3'b000. A leading bit is automatically
                       added.
ABC_TB = 5;         // ABC_TB will be assigned 3'b101.  The integer is truncated to
                       3-bits.
ABC_TB = 8;         // ABC_TB will be assigned 3'b000.  The integer is truncated to
                       3-bits.
```

Thank you !

Happy Learning