Module #02 : Lexical Tokens

Verilog source text files consists of the following lexical tokens –

2.1. White Space :

- Used to separate words
- Can contain spaces, tabs, new-lines and form feeds
- Thus a statement can be extended over multiple lines without any special continuation characters

Example 2.1 :

```
module sample (a, b, c, d);
 input a; // space in the start is ignored
reg [8*6  :1] string = "Earth ” ; // would not be ignored
 wire temp;
assign  = (a & b &c) | (a & b &c) | (a & b &c) | (a & b &c) | (a & b &c) | (a
& b &c) ; //Multiple line statement
```

Module #02 : Lexical Tokens

2.2. Comments :

Specified in two ways –

- Begin the comment with double slashes (//) : All texts between these characters (//) and the end of the lines will be ignored by the Verilog Simulators

- Enclose comments between the characters /* and */ : This method allows you to continue comments on more than one line

Example 2.2:

```
 assign c = a & b; //This is a simple comment

/* this commnet continues on more than one lines !!!
assign x = temp_reg;
assign y = temp1_reg;
*/
```

Module #02 : Lexical Tokens

2.3. Numbers:

- Number values can be specified in binary, octal, decimal or hexadecimal
- Number storage is defined as a number of bits

Example 2.3:

3'b001, A 3-bit number
5'd30, (= 5'b11110)
16'h5ED4, (= 16'd24276)

Module #02 : Lexical Tokens

2.4. Identifiers :

- User defined words for variables, function names, module names, block names and instance names
- Begins with a letter or underscore
- Never begins with a number and $
- Identifiers are case-sensitive in Verilog

Example 2.4:

Allowed Identifiers: abcd, my_module, my_design_5, myDesign5, mydesign$

Not Allowed Identifiers : 5myDesign, $myDesign

Module #02 : Lexical Tokens

2.5. Operators:

- Operators are one, two or sometimes three characters
- Used to performs operations on variables

Example 2.5:

>, +, -, ~, &, !=, ===

Module #02 : Lexical Tokens

2.6. Verilog Keywords:
- Verilog Language Specific words
- They can not be used as Verilog identifiers

Example 2.6:

assign, always, case, while, wire, reg, and, or, module, begin, input, output, inout, posedge, negedge

Module #02 : Lexical Tokens

```
module myDesign(a_i, b_i, ctrl_i, clk_i, result_o);
   input a_i;
   input b_i;
   input clk_i;
   input [1:0] ctrl_i;
   output result_o;
   reg result_o; // Registered Output
/* A small ALU operations, based on the ctrl_i signal
  values */
  always @(posedge clk_i)
   begin
    case(ctrl_i)
       2'b00 : result_o <= a_i & b_i;
       2'b01 : result_o <= a_i + 1'b1;
       2'b10 : result_o <= a_i | b_i;
       2'b11 : result_o <= a_i * b_i;
      default : result_o <= a_i;
    endcase
   end
endmodule
```

Verilog Keyword

whitespace

identifier

Single line comment

Multi line comment

operator

Number