Verilog Operators Part-I

Feb-9-2014

SystemVerilog Migration

Arithmetic Operators

- Binary: +, -, *, /, % (the modulus operator)
- Unary: +, - (This is used to specify the sign)
- Integer division truncates any fractional part
- The result of a modulus operation takes the sign of the first operand
- If any operand bit value is the unknown value x, then the entire result value is x
- Register data types are used as unsigned values (Negative numbers are stored in two's complement form)

Example

```
1  module arithmetic_operators();
2
3  initial begin
4    $display (" 5  +  10 = %d", 5  + 10);
5    $display (" 5  -  10 = %d", 5  - 10);
6    $display (" 10 -  5  = %d", 10 - 5);
7    $display (" 10 *  5  = %d", 10 * 5);
8    $display (" 10 /  5  = %d", 10 / 5);
9    $display (" 10 /  -5 = %d", 10 / -5);
10   $display (" 10 %s  3  = %d","%", 10 % 3);
11   $display (" +5     = %d", +5);
12   $display (" -5     = %d", -5);
13   #10  $finish;
14 end
15
16 endmodule
```

You could download file arithmetic_operators.v here

```
5  +  10 =  15
5  -  10 =  -5
10 -  5  =   5
10 *  5  =  50
10 /  5  =  2
10 /  -5 = -2
10 %  3  =   1
+5     =  5
-5     =  -5
```

Relational Operators

Operator	Description
a < b	a less than b
a > b	a greater than b

a <= b	a less than or equal to b
a >= b	a greater than or equal to b

- The result is a scalar value (example a < b)
- 0 if the relation is false (a is bigger then b)
- 1 if the relation is true (a is smaller then b)
- x if any of the operands has unknown x bits (if a or b contains X)

Note: If any operand is x or z, then the result of that test is treated as false (0)

Example

```
1 module relational_operators();
2
3 initial begin
4   $display (" 5   <= 10 = %b", (5     <= 10));
5   $display (" 5   >= 10 = %b", (5     >= 10));
6   $display (" 1'bx <= 10 = %b", (1'bx  <= 10));
7   $display (" 1'bz <= 10 = %b", (1'bz  <= 10));
8   #10  $finish;
9 end
10
11 endmodule
```

You could download file relational_operators.v here

```
5   <= 10 = 1
5   >= 10 = 0
1'bx <= 10 = x
1'bz <= 10 = x
```

Equality Operators

There are two types of Equality operators. Case Equality and Logical Equality.

Operator	Description
a === b	a equal to b, including x and z (Case equality)
a !== b	a not equal to b, including x and z (Case inequality)
a == b	a equal to b, result may be unknown (logical equality)
a != b	a not equal to b, result may be unknown (logical equality)

- Operands are compared bit by bit, with zero filling if the two operands do not have the same length
- Result is 0 (false) or 1 (true)
- For the == and != operators, the result is x, if either operand contains an x or a z
- For the === and !== operators, bits with x and z are included in the comparison and must match for the result to be true

Note : The result is always 0 or 1.

Example

```
1  module equality_operators();
2
3  initial begin
4    // Case Equality
5    $display (" 4'bx001 === 4'bx001 = %b", (4'bx001 ==  4'bx001));
6    $display (" 4'bx0x1 === 4'bx001 = %b", (4'bx0x1 ==  4'bx001));
7    $display (" 4'bz0x1 === 4'bz0x1 = %b", (4'bz0x1 ==  4'bz0x1));
8    $display (" 4'bz0x1 === 4'bz001 = %b", (4'bz0x1 ==  4'bz001));
9    // Case Inequality
10   $display (" 4'bx0x1 !== 4'bx001 = %b", (4'bx0x1  ! ==  4'bx001));
11   $display (" 4'bz0x1 !== 4'bz001 = %b", (4'bz0x1  ! ==  4'bz001));
12   // Logical Equality
13   $display (" 5     ==  10    = %b", (5       ==   10));
14   $display (" 5     ==  5     = %b", (5       ==   5));
15   // Logical Inequality
16   $display (" 5     !=  5     = %b", (5       ! =  5));
17   $display (" 5     !=  6     = %b", (5       ! =  6));
18    #10  $finish;
19 end
20
21 endmodule
```

You could download file equality_operators.v here

```
4'bx001 ===  4'bx001 = 1
4'bx0x1 ===  4'bx001 = 0
4'bz0x1 ===  4'bz0x1 = 1
4'bz0x1 ===  4'bz001 = 0
4'bx0x1 !==  4'bx001 = 1
4'bz0x1 !==  4'bz001 = 1
5     ==   10     = 0
5     ==   5      = 1
5     !=   5      = 0
5     !=   6      = 1
```

Logical Operators

Operator	Description
!	logic negation
&&	logical and
||	logical or

- Expressions connected by && and || are evaluated from left to right
- Evaluation stops as soon as the result is known
- The result is a scalar value:
  - 0 if the relation is false
  - 1 if the relation is true
  - x if any of the operands has x (unknown) bits

Example

```
1  module logical_operators();
2
3  initial begin
4    // Logical AND
5    $display ("1'b1 && 1'b1 = %b", (1'b1 && 1'b1));
6    $display ("1'b1 && 1'b0 = %b", (1'b1 && 1'b0));
7    $display ("1'b1 && 1'bx = %b", (1'b1 && 1'bx));
```

```
8   // Logical OR
9   $display ("1'b1 || 1'b0 = %b", (1'b1 || 1'b0));
10  $display ("1'b0 || 1'b0 = %b", (1'b0 || 1'b0));
11  $display ("1'b0 || 1'bx = %b", (1'b0 || 1'bx));
12  // Logical Negation
13  $display ("! 1'b1     = %b", ( !   1'b1));
14  $display ("! 1'b0     = %b", ( !   1'b0));
15   #10  $finish;
16 end
17
18 endmodule
```

You could download file logical_operators.v here

```
1'b1 && 1'b1 = 1
1'b1 && 1'b0 = 0
1'b1 && 1'bx = x
1'b1 || 1'b0 = 1
1'b0 || 1'b0 = 0
1'b0 || 1'bx = x
! 1'b1      = 0
! 1'b0      = 1
```

Bit-wise Operators

Bitwise operators perform a bit wise operation on two operands. They take each bit in one operand and perform the operation with the corresponding bit in the other operand. If one operand is shorter than the other, it will be extended on the left side with zeroes to match the length of the longer operand.

Operator	Description
~	negation
&	and
|	inclusive or
^	exclusive or
^~ or ~^	exclusive nor (equivalence)

- Computations include unknown bits, in the following way:
  - ~x = x
  - 0&x = 0
  - 1&x = x&x = x
  - 1|x = 1
  - 0|x = x|x = x
  - 0^x = 1^x = x^x = x
  - 0^~x = 1^~x = x^~x = x
- When operands are of unequal bit length, the shorter operand is zero-filled in the most significant bit positions.

Example

```
1 module bitwise_operators();
2
3 initial begin
4   // Bit Wise Negation
5   $display (" ~4'b0001       = %b", (~4'b0001));
6   $display (" ~4'bx001       = %b", (~4'bx001));
7   $display (" ~4'bz001       = %b", (~4'bz001));
8   // Bit Wise AND
```

```
 9    $display (" 4'b0001 &  4'b1001 = %b", (4'b0001 &  4'b1001));
10    $display (" 4'b1001 &  4'bx001 = %b", (4'b1001 &  4'bx001));
11    $display (" 4'b1001 &  4'bz001 = %b", (4'b1001 &  4'bz001));
12    // Bit Wise OR
13    $display (" 4'b0001 |  4'b1001 = %b", (4'b0001 |  4'b1001));
14    $display (" 4'b0001 |  4'bx001 = %b", (4'b0001 |  4'bx001));
15    $display (" 4'b0001 |  4'bz001 = %b", (4'b0001 |  4'bz001));
16    // Bit Wise XOR
17    $display (" 4'b0001 ^  4'b1001 = %b", (4'b0001 ^  4'b1001));
18    $display (" 4'b0001 ^  4'bx001 = %b", (4'b0001 ^  4'bx001));
19    $display (" 4'b0001 ^  4'bz001 = %b", (4'b0001 ^  4'bz001));
20    // Bit Wise XNOR
21    $display (" 4'b0001 ~^ 4'b1001 = %b", (4'b0001 ~^ 4'b1001));
22    $display (" 4'b0001 ~^ 4'bx001 = %b", (4'b0001 ~^ 4'bx001));
23    $display (" 4'b0001 ~^ 4'bz001 = %b", (4'b0001 ~^ 4'bz001));
24    #10  $finish;
25  end
26
27  endmodule
```

You could download file bitwise_operators.v here

```
~4'b0001         = 1110
~4'bx001         = x110
~4'bz001         = x110
4'b0001 &  4'b1001 = 0001
4'b1001 &  4'bx001 = x001
4'b1001 &  4'bz001 = x001
4'b0001 |  4'b1001 = 1001
4'b0001 |  4'bx001 = x001
4'b0001 |  4'bz001 = x001
4'b0001 ^  4'b1001 = 1000
4'b0001 ^  4'bx001 = x000
4'b0001 ^  4'bz001 = x000
4'b0001 ~^ 4'b1001 = 0111
4'b0001 ~^ 4'bx001 = x111
4'b0001 ~^ 4'bz001 = x111
```

Do you have any Comment? mail me at:deepak@asic-world.com