

Verilog Expressions

Verilog expressions, although superficially similar to C, have additional properties to better describe hardware. This lecture provides more details on Verilog operators and evaluation of expressions. More details are available in the Verilog standard, IEEE Std 1800-2009.

After this lecture you should be able to predict the value of a Verilog expression that uses the operators described below.

Logic Values

Verilog 4-state types, such as `logic`, can have four values: 0 (false), 1 (true), x (unknown) and z (high impedance). These are used for modeling logic.

2-state types such as `bit` and `integer` can have values 0 and 1. These are primarily used as counters or array indices.

Numeric Literals

In addition to declaring the size and base as described previously, literals (constants) can also be declared as signed by prefixing the base with 's' (e.g. `4'shf` for a 4-bit -1). Decimal numbers are treated as signed by default.

A numeric constant can also include x (unknown) or z (high-impedance) values. These have useful interpretations for synthesis (don't-care or high-impedance) but when used in simulations the result, in most cases, will be unknown (x).

The notation `'0` and `'1` indicate all-zeros or all-ones of arbitrary length.

Signedness and Size

Variables and expressions can have a size (measured in bits) and can be signed or unsigned. The behaviour of some operators depends on the signedness of the operands.

Signedness does not affect the values of the bits (1/0) only how operators act on them. Negative values of signed values are assumed to be in two's-complement form.

Decimal literals are signed and based literals are unsigned unless `s` is used with the base.

The result of an expression is unsigned unless all the operands are signed.

Figure 1¹ shows how the size of an expression depends on its operands.

Values are left-padded or left-truncated as necessary. Padding replicates the sign bit only for signed values.

Arrays

Variables may have multiple "packed" and "unpacked" dimensions.

Packed dimensions are given before the signal name. These bits are stored contiguously ("packed") and the packed item can also be treated as a scalar – a single number – in expressions. Packed dimensions typically model bit fields within a word (e.g. a 32-bit word composed of 4 bytes of 8 bits).

Unpacked dimensions appear after the signal name. These bits may not be stored contiguously. Unpacked dimensions model memories where only one element can be accessed at a time.

In array references, the unpacked dimension(s) are specified first, followed by the packed dimensions (if any).

Array Literals

Array literals (constants) can be defined by grouping the individual elements withing `{...}`. The quote distinguishes array literal syntax from the syntactically similar concatenation operator.

Examples

The examples below illustrate the rules described above.

```
module ex12 ;  
    initial begin
```

¹Tables are from the IEEE System Verilog Standard, IEEE Std 1800-2012.

Table 11-21—Bit lengths resulting from self-determined expressions

| Expression | Bit length | Comments |
|--|--------------------|--------------------------------------|
| Unsigned constant number | Same as integer | |
| Sized constant number | As given | |
| i op j, where op is: + - * / % & ^ ^~ ~^ | max(L(i),L(j)) | |
| op i, where op is: + - ~ | L(i) | |
| i op j, where op is: === !=== == != > >= < <= | 1 bit | Operands are sized to max(L(i),L(j)) |
| i op j, where op is: && -> <-> | 1 bit | All operands are self-determined |
| op i, where op is: & ~& ~ ^ ~^ ^~ ! | 1 bit | All operands are self-determined |
| i op j, where op is: >> << ** >>> <<< | L(i) | j is self-determined |
| i ? j : k | max(L(j),L(k)) | i is self-determined |
| {i, ...,j} | L(i)+..+L(j) | All operands are self-determined |
| {i{j, ...,k}} | i × (L(j)+..+L(k)) | All operands are self-determined |

Figure 1: Size of Expressions.

```

logic [3:0] x ;
logic signed [15:0] y ;
logic [3:0] [7:0] z [15:0] ;

x = 4'b01xz ; //
x = -1 + 0 ; // x=15

y = -1 + 4'shf ; // y=-2
x = y ; // x=14

z[0] = '1 ; // z[0]=4294967295
z[0] = {4{4'b1}} ; // z[0]=4369
z[0][0][7] = 1 ; // z[0]=4497
z[15:0] = '{16{z[0]}} ; // z[*]=4497

$finish ;
end

endmodule

```

Operators

Figure 2 below lists the Verilog operators and Figure 3 their precedence. Operators that differ from those in C and that are widely supported for synthesis are described below.

Arithmetic vs Logical Shift Left shift always zero-fills on the right. Arithmetic right shifts (>>>) replicate the sign bit if the result is signed. Logical right shifts (>>) always zero-fill.

Logical Reduction Operators These unary (one operand) operators apply a logical operation to the bits of the operand. For example, to test if any bit is set we can apply the or-reduction operator.

Conditional Operator The result when the conditional expression contains x or z is not what you might expect (the rules are complex).

Equality The regular equality comparison operators (e.g. ==) return x if either operand contains

Table 11-1—Operators and data types

| Operator token | Name | Operand data types |
|------------------------|---|---|
| = | Binary assignment operator | Any |
| += -= /= *= | Binary arithmetic assignment operators | Integral, real , shortreal |
| %= | Binary arithmetic modulus assignment operator | Integral |
| &= = ^= | Binary bitwise assignment operators | Integral |
| >>= <<= | Binary logical shift assignment operators | Integral |
| >>>= <<<= | Binary arithmetic shift assignment operators | Integral |
| ?: | Conditional operator | Any |
| + - | Unary arithmetic operators | Integral, real , shortreal |
| ! | Unary logical negation operator | Integral, real , shortreal |
| ~ & ~& ~ ^ ~^ ^~ | Unary logical reduction operators | Integral |
| + - * / ** | Binary arithmetic operators | Integral, real , shortreal |
| % | Binary arithmetic modulus operator | Integral |
| & ^ ^~ ~^ | Binary bitwise operators | Integral |
| >> << | Binary logical shift operators | Integral |
| >>> <<< | Binary arithmetic shift operators | Integral |
| && -> <-> | Binary logical operators | Integral, real , shortreal |
| < <= > >= | Binary relational operators | Integral, real , shortreal |
| === != | Binary case equality operators | Any except real and shortreal |
| == != | Binary logical equality operators | Any |
| ==? !=? | Binary wildcard equality operators | Integral |
| ++ -- | Unary increment, decrement operators | Integral, real , shortreal |
| inside | Binary set membership operator | Singular for the left operand |
| dist | Binary distribution operator | Integral |
| { } { } | Concatenation, replication operators | Integral |
| {<<{ } } {>>{ } } | Stream operators | Integral |

Figure 2: Verilog operators.

x or z.

The 4-state comparison (===) compares the two operands for an exact match, including x and z.

The wildcard equality comparison (==?) excludes bits that are x or z in the right operand from the comparison. The result is always 0 or 1.

Concatenation Bits can be concatenated by separating expressions with commas and surrounding them with braces ({}).

Table 11-2—Operator precedence and associativity


| Operator | Associativity | Precedence |
|---|---------------|--|
| () [] :: . | Left | Highest  Lowest |
| + - ! ~ & ~& ~ ^ ~^ ^~ ++ -- (unary) | | |
| ** | Left | |
| * / % | Left | |
| + - (binary) | Left | |
| << >> <<< >>> | Left | |
| < <= > >= inside dist | Left | |
| == != === !== ==? !=? | Left | |
| & (binary) | Left | |
| ^ ~^ ^~ (binary) | Left | |
| (binary) | Left | |
| && | Left | |
| | Left | |
| ?: (conditional operator) | Right | |
| -> <-> | Right | |
| = += -= *= /= %= &= ^= = <<= >>= <<<= >>>= := :/ <= | None | |
| { } { } | Concatenation | |

Figure 3: Operator Precedence.

Concatenations of variables can be used on the left hand side of an assignment.

Replication The syntax is similar to concatenation but uses two pairs of nested braces and repetition value.

Array Slices The array subscript operator can be used to extract contiguous portions (slices) of an array. The bit order cannot be reversed.

Cast Although not an operator, a cast ('') can be used to change the type, size or signedness of an expression.

```

logic [15:0] x ;
logic signed [15:0] y ;

x = 16'hfff0 ; // x=65520
y = x >>> 1 ; // y=0x7ff8
y = signed'(x) >>> 1 ; // y=0xffff8
y = |y ; // y=1

x = 8'h4x ; // x=X
y = x == 8'h4x ; // y=X
y = x[6:3] === 7'b100x ; // y=1
y = x ==? 8'h4x ; // y=1
y = {x[7:4],x[6]} ; // y=0x0009
$finish ;
end

endmodule

```

Examples

```

module ex13 ;
    initial begin

```