Examples:

```
localparam p = 7;
reg [7:0] m [5:1][5:1];
integer i;

m[1][i]     // longest static prefix is m[1]

m[p][1]     // longest static prefix is m[p][1]

m[i][1]     // longest static prefix is m
```

11.6 Expression bit lengths

The number of bits of an expression is determined by the operands and the context. Casting can be used to set the size context of an intermediate value (see 6.24).

Controlling the number of bits that are used in expression evaluations is important if consistent results are to be achieved. Some situations have a simple solution; for example, if a bitwise AND operation is specified on two 16-bit variables, then the result is a 16-bit value. However, in some situations, it is not obvious how many bits are used to evaluate an expression or what size the result should be.

For example, should an arithmetic add of two 16-bit values perform the evaluation using 16 bits, or should the evaluation use 17 bits in order to allow for a possible carry overflow? The answer depends on the type of device being modeled and whether that device handles carry overflow.

SystemVerilog uses the bit length of the operands to determine how many bits to use while evaluating an expression. The bit length rules are given in 11.6.1. In the case of the addition operator, the bit length of the largest operand, including the left-hand side of an assignment, shall be used.

For example:

```
logic [15:0] a, b;   // 16-bit variables
logic [15:0] sumA;   // 16-bit variable
logic [16:0] sumB;   // 17-bit variable

sumA = a + b;        // expression evaluates using 16 bits
sumB = a + b;        // expression evaluates using 17 bits
```

11.6.1 Rules for expression bit lengths

The rules governing the expression bit lengths have been formulated so that most practical situations have a natural solution.

The number of bits of an expression (known as the *size* of the expression) shall be determined by the operands involved in the expression and the context in which the expression is given.

A *self-determined expression* is one where the bit length of the expression is solely determined by the expression itself—for example, an expression representing a delay value.

A *context-determined expression* is one where the bit length of the expression is determined by the bit length of the expression and by the fact that it is part of another expression. For example, the bit size of the right-hand expression of an assignment depends on itself and the size of the left-hand side.

Table 11-21 shows how the form of an expression shall determine the bit lengths of the results of the expression. In Table 11-21, `i`, `j`, and `k` represent expressions of an operand, and `L(i)` represents the bit length of the operand represented by `i`.

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

Expression	Bit length	Comments
Unsized 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

Multiplication may be performed without losing any overflow bits by assigning the result to something wide enough to hold it.

11.6.2 Example of expression bit-length problem

During the evaluation of an expression, interim results shall take the size of the largest operand (in case of an assignment, this also includes the left-hand side). Care has to be taken to prevent loss of a significant bit during expression evaluation. The following example describes how the bit lengths of the operands could result in the loss of a significant bit.

Given the following declarations:

```
logic [15:0] a, b, answer; // 16-bit variables
```

the intent is to evaluate the expression

```
answer = (a + b) >> 1; // will not work properly
```

where `a` and `b` are to be added, which can result in an overflow, and then shifted right by 1 bit to preserve the carry bit in the 16-bit answer.

A problem arises, however, because all operands in the expression are of a 16-bit width. Therefore, the expression `(a + b)` produces an interim result that is only 16 bits wide, thus losing the carry bit before the evaluation performs the 1-bit right shift operation.

The solution is to force the expression `(a + b)` to evaluate using at least 17 bits. For example, adding an integer value of `0` to the expression will cause the evaluation to be performed using the bit size of integers. The following example will produce the intended result:

```
answer = (a + b + 0) >> 1; // will work correctly
```

In the following example:

```
module bitlength();
   logic [3:0] a, b, c;
   logic [4:0] d;

   initial begin
      a = 9;
      b = 8;
      c = 1;
      $display("answer = %b", c ? (a&b) : d);
   end
endmodule
```

the `$display` statement will display

```
answer = 01000
```

By itself, the expression `a&b` would have the bit length 4, but because it is in the context of the conditional expression, which uses the maximum bit length, the expression `a&b` actually has length 5, the length of `d`.

11.6.3 Example of self-determined expressions

```
logic [3:0] a;
logic [5:0] b;
logic [15:0] c;

initial begin
   a = 4'hF;
   b = 6'hA;
   $display("a*b=%h", a*b);   // expression size is self-determined
   c = {a**b};                // expression a**b is self-determined
                              // due to concatenation operator {}
   $display("a**b=%h", c);
   c = a**b;                  // expression size is determined by c
   $display("c=%h", c);
end
```

Simulator output for this example:

```
a*b=16   // 'h96 was truncated to 'h16 since expression size is 6
a**b=1   // expression size is  4 bits (size of a)
c=ac61   // expression size is 16 bits (size of c)
```

11.7 Signed expressions

Controlling the sign of an expression is important if consistent results are to be achieved. 11.8.1 outlines the rules that determine if an expression is signed or unsigned.