1 Expression bit lengths

The number of bits of any expression is determined by the operands and the context in which it occurs. Casting can be used to set the target size 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. The following typing system provides precise rules for determining expression bit lengths in all situations.

1.1 Bidirectional typing operations

The bit length of expressions is defined using the fundamental concepts:

Self-determined size We call *self-determined-size* the size that is intrinsic to an expression: i.e. it is solely based on the expression's internal structure and operands.

Resizing Expressions *may be resized* to sizes greater than or equal to their *self-determined-size*. This operation may change the size of the internal expression.

1.2 Expression categories for resizing

SystemVerilog expressions shall be categorized into two types based on their resizing behavior:

1.2.1 Atomically resizable expressions

Atomically resizable expressions *may be resized* without affecting their internal operand sizes. The following expressions are atomically resizable:

- Operands as defined in 11.2 (nets, variables, literals, function calls, etc.)
- Comparison expressions: `===`, `!==`, `==?`, `!=?`, `==`, `!=`, `>`, `>=`, `<`, `<=`
- Logical expressions: `&&`, `||`, `->`, `<->`
- Reduction expressions: `&`, `~&`, `|`, `~|`, `^`, `~^`, `^~`, `!`
- Assignment expressions: `=`, `+=`, `-=`, `*=`, `/=`, `%=`, `&=`, `|=`, `^=`
- Shift assignment expressions: `<<=`, `>>=`, `<<<=`, `>>>=`
- Concatenation expressions: `{...}`
- Replication expressions: `{.{...}}`
- Set membership expressions: `inside`

When an atomically resizable expression is resized to a target size, only the expression's result shall be extended — its operands shall remain unmodified.

Rule (Atomic-Resize): If e has a *self-determined-size* of t and n is larger than t and e is atomically resizable, then e *may be resized* to n.

1.2.2 Non-atomically resizable expressions

Non-atomically resizable expressions propagate resizing to their operands when a target size is specified. These expressions require their operands to be adjusted to specific sizes based on the resizing rules. The following expression are not atomically resizable:

- Binary and bitwise expression: `+`, `-`, `*`, `/`, `%`, `&`, `|`, `^`, `^~`, `~^`
- Unary arithmetic, bitwise, increment and decrement expressions: `+`, `-`, `~`, `++`, `--`
- Shift and power expression: `>>`, `<<`, `**`, `>>>`, `<<<`
- Conditional expression: `?:`

Binary arithmetic and bitwise expressions propagate the target size to both operands:

Rule (Binary-Resize): If a *may be resized* to n and b *may be resized* to n, then $a \oplus b$ *may be resized* to n.

Unary arithmetic, unary bitwise negation and unary increment and decrement expressions propagate the target size to their single operand:

Rule (Unary-Resize): If e *may be resized* to n, then $\oplus e$ *may be resized* to n.

Shift and power expressions propagate the target size only to the left operand, while the right operand remains *self-determined*:

Rule (Shift-Resize): If a *may be resized* to n and b has a *self-determined-size* of t_b, then $a \oplus b$ *may be resized* to n.

Conditional expressions propagate the target size to both branch expressions, while the condition remains *self-determined*:

Rule (Conditional-Resize): If c has a *self-determined-size* size of t_C and t_e *may be resized* to n and f_e *may be resized* to n, then c ? t_e : f_e *may be resized* to n.

1.3 *Self-determined* expression sizing rules

The *self-determined-size* of an expression, solely based on its internal structure and operands, shall be computed according to the following rules:

1.3.1 Operands

For operands as defined in 11.2, the *self-determined-size* is always well-defined and determined by their declaration, literal specification, or result type:

Rule (Operand-Size): If e is an operand and s its size, then e shall have a *self-determined-size* of s.

Examples:

- Sized integer literals: `8'hFF` has a *self-determined-size* of 8, `32'd123` has a *self-determined-size* of 32,
- Unsized integer literals: `123`, `'hABC` have a *self-determined-size* of at least 32 bits,
- Parameters, nets, variables and structure fields have their size defined by their declaration: `logic [15:0] data` has a *self-determined-size* of 16,
- Bit-select: `data[5]` has a *self-determined-size* of 1,
- Part-select: `data[7:0]` has a *self-determined-size* of 8, `data[base +: 4]` has a *self-determined-size* of 4,
- Function calls: Have their size defined by their return type — a function returning `logic [31:0]` has a *self-determined-size* of 32,
- Variadic sized function calls: For functions whose return type depends on their arguments, the arguments' sizes shall be determined as if they were in an assignment context. Once all argument sizes are determined, the function's result type becomes known and defines the *self-determined-size*.

1.3.2 Binary arithmetic and bitwise expressions

For binary arithmetic and bitwise expressions, the *self-determined-size* is the maximum of the operand sizes. The smaller operand is *resized* to match the larger operand's size.

Rule (Binary-Left-Size): If a has a *self-determined-size* of t and b *may be resized* to t, then $a \oplus b$ shall have a *self-determined-size* of t.

Rule (Binary-Right-Size): If b has a *self-determined-size* of t and a *may be resized* to t, then $a \oplus b$ shall have a *self-determined-size* of t.

1.3.3 Unary expressions

For unary expressions (Unary arithmetic, unary bitwise negation and unary increment and decrement), the *self-determined-size* is identical to the operand size.

Rule (Unary-Size): If e has a *self-determined-size* of t, then $\oplus e$ shall have a *self-determined-size* of t.

1.3.4 Relational and equality expressions

For relational and equality expressions, the *self-determined-size* is always 1 bit. The smaller operand shall be resized to match the larger operand's size for comparison purposes.

Rule (Relational-Left-Size): If a has a *self-determined-size* of t and b *may be resized* to t, then $a \oplus b$ shall have a *self-determined-size* of 1.

Rule (Relational-Right-Size): If b has a *self-determined-size* of t and a *may be resized* to t, then $a \oplus b$ shall have a *self-determined-size* of 1.

1.3.5 Logical expressions

For binary logical expressions, the *self-determined-size* is always 1 bit. All operands are *self-determined.*

Rule (Logical-Size): If a has a *self-determined-size* of t_a and b has a *self-determined-size* of t_b, then $a \oplus b$ shall have a *self-determined-size* of 1.

1.3.6 Reduction expressions

For reduction expressions, including `!`, the *self-determined-size* is always 1 bit. The operand is *self-determined.*

Rule (Reduction-Size): If e has a *self-determined-size* of t, then $\oplus e$ shall have a *self-determined-size* of 1.

1.3.7 Shift and power expressions

For shift and power expressions, the *self-determined-size* is determined by the left operand. The right operand shall be *self-determined.*

Rule (Shift-Size): If a has a *self-determined-size* of t and b has a *self-determined-size* of t_b, then $a \oplus b$ shall have a *self-determined-size* of t.

1.3.8 Assignment expressions

For assignment expressions, the *self-determined-size* is determined by the left-hand side. When the left-hand side has a larger size than the right-hand side, the right-hand side shall *be resized.* Otherwise, the right-hand side shall be *self-determined.*

Rule (Assignment-Left-Size): If the left-hand side l has a size of t and e *may be resized* to t, then $l \oplus e$ shall have a *self-determined-size* of t.

Rule (Assignment-Right-Size): If the left-hand side l has a size of t, e has a *self-determined-size* of t_e and t is smaller than t_e, then $l \oplus e$ shall have a *self-determined-size* of t.

For shift assignment expressions, the right operand is always *self-determined.*

Rule (Shift-Assignment-Size): If the left-hand side l has a size of t and e has a *self-determined-size* of t_e, then $l \oplus e$ shall have a *self-determined-size* of t.

1.3.9 Conditional expressions

For conditional expressions using the `?:` operator, the *self-determined-size* is the maximum size of the two branch expressions. The smaller branch shall be resized to match the larger branch. The condition shall be *self-determined.*

Rule (Conditional-Left-Size): If c has a *self-determined-size* of t_c, a has a *self-determined-size* of t, and b *may be resized* to t, then $c?a:b$ shall have a *self-determined-size* of t.

Rule (Conditional-Right-Size): If c has a *self-determined-size* of t_c, b has a *self-determined-size* of t, and a *may be resized* to t, then $c?a:b$ shall have a *self-determined-size* of t.

1.3.10 Concatenation expressions

For concatenation expressions, the *self-determined-size* is the sum of the *self-determined sizes* of all operands.

Rule (Concatenation-Size): If e_1 has a *self-determined-size* of t_1, …, e_k has a *self-determined-size* of t_k, and t is the sum of $t_1, \dots, t_k$, then $\{e_1, \dots, e_k\}$ shall have a *self-determined-size* of t.

1.3.11 Replication expressions

The *self-determined-size* of a replication is the *self-determined-size* of the inner concatenation multiplied by the replication amount.

Rule (Replication-Size): If i is the amount of the replication and e_{in} has a *self-determined-size* of t_{in}, and t is $i \times t_{in}$, then $\{i\{e_{in}\}\}$ shall have a *self-determined-size* of t.

2 Examples

Consider the following SystemVerilog declarations:

```
logic [7:0] var8;        // 8-bit variable
logic [31:0] var32;      // 32-bit variable
logic [15:0] var16;      // 16-bit variable
logic cond;              // condition signal
logic [63:0] result;     // 64-bit result variable
```

2.1 Basic Expression Sizing

In the previous context, the expression `var8` has *self-determined-size* 8. Indeed, by rule **Operand-Size**:

- `var8` is an operand with size 8 (by declaration `logic [7:0] var8`)

In the previous context, the expression `var16[15:8] + 4'b1001` has *self-determined-size* 8. Indeed, by rule **Binary-Left-Size**:

- `var16[15:8]` has *self-determined-size* 8 (by rule **Operand-Size**, part-select of 8 bits)
- `4'b1001` *may be resized* to 8 by rule **Resize**:
  - `4'b1001` has *self-determined-size* 4 (by rule **Operand-Size**, sized integer literal)
  - 8 is larger than 4
  - `4'b1001` is atomically resizable (operands are atomically resizable)

If we tried to apply rule **Binary-Right-Size** on the previous expression we would end up stuck resizing `var16[15:8]` to 4 bits.

In the previous context, the expression `var16[5] + 8'hFF` has *self-determined-size* 8. Indeed, by rule **Binary-Right-Size**:

- `8'hFF` has *self-determined-size* 8 (by rule **Operand-Size**, sized integer literal)
- `var16[5]` *may be resized* to 8 by rule **Resize**:
  - `var16[5]` has *self-determined-size* 1 (by rule **Operand-Size**, bit-select)
  - 8 is larger than 1
  - `var16[5]` is atomically resizable (operands are atomically resizable)

2.2 Relational Expression Example

In the previous context, the expression `var16 > 16'd100` has *self-determined-size* 1. Indeed, by rule **Relational-Left-Size**:

- `var16` has *self-determined-size* 16 (by rule **Operand-Size**, declaration `logic [15:0] var16`)
- `16'd100` *may be resized* to 16 — this is actually not needed as it already has size 16:
  - `16'd100` has *self-determined-size* 16 (by rule **Operand-Size**, sized integer literal)

2.3 Reduction Expression Example

In the previous context, the expression `&var16[7:0]` has *self-determined-size* 1. Indeed, by rule **Reduction-Size**:

- `var16[7:0]` has *self-determined-size* 8 (by rule **Operand-Size**, part-select of 8 bits)

2.4 Replication Expression Example

In the previous context, the expression `{4{var8}}` has *self-determined-size* 32. Indeed, by rule **Replication-Size**:

- The replication amount i is 4
- `var8` has *self-determined-size* 8 (by rule **Operand-Size**, declaration `logic [7:0] var8`)
- The result size is $4 \times 8 = 32$

2.5 Complex Replication with Concatenation

In the previous context, the expression `{2{var16[7:0], 4'hF}}` has *self-determined-size* 24. With rule **Replication-Size**:

- The replication amount i is 2
- The inner concatenation `{var16[7:0], 4'hF}` has *self-determined-size* 12 by rule **Concatenation-Size**:
  - `var16[7:0]` has *self-determined-size* 8 (by rule **Operand-Size**, part-select)
  - `4'hF` has *self-determined-size* 4 (by rule **Operand-Size**, sized literal)
  - Sum is $8 + 4 = 12$
- The result size is $2 \times 12 = 24$

2.6 Assignment with Target Size Extension

In the previous context, the expression `var32 = var16[7:0] + 1` has *self-determined-size* 32. With rule **Assignment-Left-Size**:

- Left-hand side `var32` has size 32 (by declaration `logic [31:0] var32`)
- Right-hand side `var16[7:0] + 1` *may be resized* to 32 by rule **Binary-Resize**:
  - `var16[7:0]` *may be resized* to 32 by rule **Resize**:
    - `var16[7:0]` has *self-determined-size* 8 (by rule **Operand-Size**, part-select)
    - 32 is larger than 8
    - `var16[7:0]` is atomically resizable (operands are atomically resizable)
  - `1` *may be resized* to 32 (unsized literals have *self-determined-size* at least 32)

2.7 Assignment with Result Truncation

In the previous context, the expression `var8 = var32 + var16` has *self-determined-size* 8. Indeed, by rule **Assignment-Right-Size**:

- Left-hand side `var8` has size 8 (by declaration `logic [7:0] var8`)
- Right-hand side `var32 + var16` has *self-determined-size* 32 by rule **Binary-Left-Size**:
  - `var32` has *self-determined-size* 32 (by rule **Operand-Size**, declaration `logic [31:0] var32`)
  - `var16` *may be resized* to 32 by rule **Resize**:
    - `var16` has *self-determined-size* 16 (by rule **Operand-Size**, declaration `logic [15:0] var16`)
    - 32 is larger than 16
    - `var16` is atomically resizable (operands are atomically resizable)
- 8 is smaller than 32

2.8 Conditional Expression with True Branch Determining Size

In the previous context, the expression `cond ? var32 : var8` has *self-determined-size* 32. Indeed, by rule **Conditional-Left-Size**:

- Condition `cond` has *self-determined-size* 1 (by rule **Operand-Size**, declaration `logic cond`)
- True branch `var32` has *self-determined-size* 32 (by rule **Operand-Size**, declaration `logic [31:0] var32`)
- False branch `var8` *may be resized* to 32 by rule **Resize**:
  - `var8` has *self-determined-size* 8 (by rule **Operand-Size**, declaration `logic [7:0] var8`)
  - 32 is larger than 8
  - `var8` is atomically resizable (operands are atomically resizable)

2.9 Conditional Expression with False Branch Determining Size

In the previous context, the expression `cond ? var8 : var32` has *self-determined-size* 32. Indeed, by rule **Conditional-Right-Size**:

- Condition `cond` has *self-determined-size* 1 (by rule **Operand-Size**, declaration `logic cond`)
- False branch `var32` has *self-determined-size* 32 (by rule **Operand-Size**, declaration `logic [31:0] var32`)
- True branch `var8` *may be resized* to 32 by rule **Resize**:
  - `var8` has *self-determined-size* 8 (by rule **Operand-Size**, declaration `logic [7:0] var8`)
  - 32 is larger than 8
  - `var8` is atomically resizable (operands are atomically resizable)

2.10 Conditional Expression with Context-Driven Sizing

In the previous context, the expression `result = cond ? var32[7:0] : var32[15:8]` has *self-determined-size* 64. Indeed, by rule **Assignment-Left-Size**:

- Left-hand side `result` has size 64 (by declaration `logic [63:0] result`)
- Right-hand side `cond ? var32[7:0] : var32[15:8]` *may be resized* to 64 by rule **Conditional-Resize**:
  - Condition `cond` has *self-determined-size* 1 (by rule **Operand-Size**, declaration `logic cond`)
  - True branch `var32[7:0]` *may be resized* to 64 by rule **Resize**:
    - `var32[7:0]` has *self-determined-size* 8 (by rule **Operand-Size**, part-select)
    - 64 is larger than 8
    - `var32[7:0]` is atomically resizable (operands are atomically resizable)
  - False branch `var32[15:8]` *may be resized* to 64 by rule **Resize**:
    - `var32[15:8]` has *self-determined-size* 8 (by rule **Operand-Size**, part-select)
    - 64 is larger than 8
    - `var32[15:8]` is atomically resizable (operands are atomically resizable)

A Algorithm Overview

This appendix presents an algorithm to compute the size of all sub-expressions of a SystemVerilog expression. The algorithm operates in two phases:

First, the *self-determined-size* of the expression is computed using the algorithm 1. This algorithm traverses the expression tree bottom-up to determine the natural size of each expression based solely on its internal structure and operands.

Second, the expression and all its sub-expressions are resized to the target size using the algorithm 2. During this propagation phase, all self-determined sub-expressions are resized to their *self-determined-size*, while the other sub-expressions inherit their size from the surrounding context.

Assuming that call to the Determine function are cached, the algorithm runs in linear time with respect to the number of operations in the SystemVerilog expression. The reasoning implemented in this algorithm follows the typing rules explained in the previous section 1.

Algorithm 1 Compute the *self-determined-size* of an expression.

```
procedure DETERMINE(expr)
Require: expr must be a SystemVerilog expression.
Ensure: The output is the self-determined-size of expr.
    switch expr
        when expr is operand:
            The self-determined-size is the operand's declared or literal size.

        when expr is lhs ⊕ rhs:
            // ⊕ can be +, -, *, /, %, &, |, ^, ^~, ~^
            Compute the self-determined-size of lhs, as lhs_size.
            Compute the self-determined-size of rhs, as rhs_size.
            The self-determined-size of expr is max(lhs_size, rhs_size).

        when expr is ⊕arg:
            // ⊕ can be +, -, ~, ++, --
            Compute the self-determined-size of arg, as arg_size.
            The self-determined-size of expr is arg_size.

        when expr is lhs ⊕ rhs:
            // ⊕ can be ===, !==, ==?, !=?, ==, !=, >, >=, <, <=
            The self-determined-size of expr is 1 bit.

        when expr is lhs ⊕ rhs:
            // ⊕ can be &&, ||, ->, <->
            The self-determined-size of expr is 1 bit.

        when expr is ⊕arg:
            // ⊕ can be &, ~&, |, ~|, ^, ~^, ^~, !
            The self-determined-size of expr is 1 bit.

        when expr is lhs ⊕ rhs:
            // ⊕ can be >>, <<, **, >>>, <<<
            Compute the self-determined-size of lhs as lhs_size.
            The self-determined-size of expr is lhs_size.

        when expr is lval ⊜ rhs:
            // ⊜ can be =, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=, <<<=, >>>=
            Retrieve the size of the left-hand side variable lval as lval_size.
            The self-determined-size of expr is lval_size.

        when expr is cond ? true_expr : false_expr:
            Compute the self-determined-size of true_expr, as true_size.
            Compute the self-determined-size of false_expr, as false_size.
            The self-determined-size of expr is max(true_size, false_size).

        when expr is {expr_1, expr_2, ..., expr_N}:
            for i ∈ {1, ..., N} do
                Compute the self-determined-size of expr_i, as expr_i_size.
            end for
            The self-determined-size of expr is expr_1_size + ··· + expr_N_size.

        when expr is {N{inner_expr}}:
            Compute the self-determined-size of inner_expr, as inner_size.
            The self-determined-size of expr is N × inner_size.

    end switch
end procedure
```

Algorithm 2 Resize an expression, propagating the size.

```
procedure PROPAGATE(expr, target_size)
Require: expr is a SystemVerilog expression. target_size is the size expr will be resized to.
Ensure: All sub-expressions of expr are annotated with their final size in expr.
    switch expr
        when expr is operand:
            Annotate expr with target_size.

        when expr is lhs ⊕ rhs:
            // ⊕ can be +, -, *, /, %, &, |, ^, ^~, ~^
            Propagate target_size into lhs.
            Propagate target_size into rhs.
            Annotate expr with target_size.

        when expr is ⊕arg:
            // ⊕ can be +, -, ~, ++, --
            Propagate target_size into arg.
            Annotate expr with target_size.

        when expr is lhs ⊕ rhs:
            // ⊕ can be ===, !==, ==?, !=?, ==, !=, >, >=, <, <=
            Compute the self-determined-size of lhs, as lhs_size.
            Compute the self-determined-size of rhs, as rhs_size.
            Let arg_size be max (lhs_size, rhs_size).
            Propagate arg_size into lhs.
            Propagate arg_size into rhs.
            Annotate expr with target_size.

        when expr is lhs ⊕ rhs:
            // ⊕ can be &&, ||, ->, <->
            Compute the self-determined-size of lhs, as lhs_size.
            Propagate lhs_size into lhs.
            Compute the self-determined-size of rhs, as rhs_size.
            Propagate rhs_size into rhs.
            Annotate expr with target_size.

        when expr is ⊕arg:
            // ⊕ can be &, ~&, |, ~|, ^, ~^, ^~, !
            Compute the self-determined-size of arg, as arg_size.
            Propagate arg_size into arg.
            Annotate expr with target_size.

        when expr is lhs ⊕ rhs:
            // ⊕ can be >>, <<, **, >>>, <<<
            Propagate target_size into lhs.
            Compute the self-determined-size of rhs, as rhs_size.
            Propagate rhs_size into rhs.
            Annotate expr with target_size.
```

```
        when expr is lval ⊜ rhs:
            // ⊜ can be =, +=, -=, *=, /=, %=, &=, |=, ^=
            Let lval_size be the size of the left-hand side lval.
            Compute the self-determined-size of rhs, as rhs_size.
            Let arg_size be max (lval_size, rhs_size).
            Propagate arg_size into rhs.
            Annotate expr with target_size.

        when expr is lval ⊜ rhs:
            // ⊜ can be <<=, >>=, <<<=, >>>=
            Compute the self-determined-size of rhs, as rhs_size.
            Propagate rhs_size into rhs.
            Annotate expr with target_size.

        when expr is cond ? true_expr : false_expr:
            Compute the self-determined-size of cond, as cond_size.
            Propagate cond_size into cond.
            Propagate target_size into true_expr.
            Propagate target_size into false_expr.
            Annotate expr with target_size.

        when expr is {expr_1, expr_2, ..., expr_N}:
            for i ∈ {1, ..., N} do
                Compute the self-determined-size of expr_i, as expr_i_size.
                Propagate expr_i_size into expr_i.
            end for
            Annotate expr with target_size.

        when expr is {N{inner_expr}}:
            Compute the self-determined-size of inner_expr, as inner_size.
            Propagate inner_size into inner_expr.
            Annotate expr with target_size.

    end switch
end procedure
```