

Elaboration-time user error messages

Objectives:

- To enable user controlled verification of parameter values during the model elaboration, the proposal introduces a new elaboration-time assert statement and several elaboration-time reporting system tasks.
- This proposal has dependency on *generate constructs* in sequences and properties.

23 Compiler directives

Add new clause

23.5 Elaboration-time assertions and reporting system tasks

It is often necessary to validate the actual parameter values used in a SystemVerilog model. This can be achieved by using the immediate assertion **assert** that evaluates a `constant_expression` and optionally executes an elaboration-time action. The assertion must be placed outside procedural code to be recognized as an elaboration-time assertion.

```
elaboration_assertion ::=
    [assert] (constant_expression) [elaboration_time_action]

elaboration_time_action ::=
    [pass_elaboration_action] [else fail_elaboration_action]
    | ;

pass_elaboration_action ::= elaboration_action

fail_elaboration_action ::= elaboration_action

elaboration_action ::=
    $fatal(list_of_arguments);
    | $error(list_of_arguments);
    | $warning(list_of_arguments);
    | $info(list_of_arguments);
```

Syntax 23-?? Elaboration-time assertion syntax

`list_of_arguments` can only contain a formatting string and constant expressions (including constant function calls). If a **assert** remains outside procedural code after generate statement unrolling, the argument `constant_expression` is evaluated and if it is *true* then the `pass_elaboration_action` is executed, if any, else if *false* then the `fail_elaboration_action` is executed, if any.

If either of the elaboration-action blocks is omitted the compiler may issue a default message consisting of the type of the task (error, warning, info) and the file and line number of the elaboration-time assertion that triggered the message.

if **assert** is missing in front of a `pass_elaboration_action` it is equivalent to writing **assert(1'b1)**`pass_elaboration_action;`

If `$fatal` is executed then after outputting the message the compilation is aborted. If `$error` is executed then the message is issued and compilation continues. However, no simulation run-time code is generated. The other two tasks `$warning` and `$info` only output their text message but do not affect the compilation and run-time code generation.

Example: Sometimes it is desirable to assert validate elaboration-time constants, such as bounds on a parameter, in a way that can be enforced during model compilation. In this example, if the module parameter value is outside the range 1 to 8, an error is issued and the compilation will be aborted.

```
module test #(N=12) (input [N-1:0] in, output [N-1:0] out);
  assert((N > 0) && (N <= 8))
    else $error("Parameter N = %0d is out of valid range in module instance
m%", N);
  assign out = in;
endmodule
```

Example: In this simple example, the generate construct builds a concatenation (##1) of subsequences, each of length 1, over a bit from a vector passed as argument to the top sequence definition. Elaboration-time messages are used to indicate if the vector is just a scalar, otherwise it will issue an information message indicating which conditional branch was taken.

```
sequence s(vect);
  generate
    if ($bits(vect) == 1) begin : err $error("Not a vector"); end
    for (genvar i = 0; i < $bits(vect); i++) begin : Loop
      if (i==0) begin : Cond
        sequence t; vect[0]; endsequence
        $info("i=0 branch taken");
      end : Cond
      else begin : Cond
        sequence t; vect[i] ##1 Loop[i-1].Cond.t; endsequence
        $info("i != 0 branch taken")
      end : Cond
    end : Loop
  endgenerate
  Loop[$bits(vect)-1].Cond.t;
endsequence
```

In Table *Syntax 17-4*, and *A.2.10*

REPLACE

```
sequence_item ::=
  assertion_variable_declaration
  | sequence_declaration
```

WITH

```
sequence_item ::=
  assertion_variable_declaration
  | sequence_declaration
  | elaboration_assertion
```

In A.1.4 Module items

ADD right after the title

```
elaboration_assertion ::=  
    [assert] (constant_expression) [elaboration_time_action]
```

```
elaboration_time_action ::=  
    [pass_elaboration_action] [else fail_elaboration_action]  
    | ;
```

```
pass_elaboration_action ::= elaboration_action
```

```
fail_elaboration_action ::= elaboration_action
```

```
elaboration_action ::=  
    $fatal(list_of_arguments);  
    | $error(list_of_arguments);  
    | $warning(list_of_arguments);  
    | $info(list_of_arguments);
```

REPLACE

```
module_common_item ::=  
    module_or_generate_item_declaration  
    | interface_instantiation  
    | program_instantiation  
    | concurrent_assertion_item  
    | bind_directive  
    | continuous_assign  
    | net_alias  
    | initial_construct  
    | final_construct  
    | always_construct  
    | loop_generate_construct  
    | conditional_generate_construct
```

WITH

```
module_common_item ::=  
    module_or_generate_item_declaration  
    | interface_instantiation  
    | program_instantiation  
    | concurrent_assertion_item  
    | bind_directive  
    | continuous_assign  
    | net_alias  
    | initial_construct  
    | final_construct  
    | always_construct
```

- | loop_generate_construct
- | conditional_generate_construct
- | [elaboration_assertion](#)

In A.1.7 Program items

REPLACE

```
program_generate_item37 ::=  
  loop_generate_construct  
  | conditional_generate_construct  
  | generate_region
```

WITH

```
program_generate_item37 ::=  
  loop_generate_construct  
  | conditional_generate_construct  
  | generate_region  
  | elaboration\_assertion
```