The proposal is aligned with P1800-2008 Draft 4

Elaboration-time user error messages

Objectives:

- To enable user controlled verification of parameter values during model elaboration and issuing of informative message, the proposal introduces elaboration system tasks.

19 Utility system tasks and functions

Add to 19.1 General

Elaboration system tasks ([Editor please complete clause reference]**)**

$fatal	$error
$warning	$info

Add new clause after the last existing clause

19.10 Elaboration system tasks [Editor please insert after 19.9 and adjust clause numbers as needed]

It is often necessary to validate the actual parameter values used in a SystemVerilog model and report any error without generating the executable simulation model. This is achieved by using elaboration system tasks. These tasks have the same names as the severity system tasks (Editor please insert reference to 19.9 Severity system tasks) that can be used during simulation. However, the elaboration system tasks shall be called outside procedural code and their activation can be controlled by conditional generate constructs. If such a task is called from within a procedure, then it becomes a simulation-time severity system task.

```
elaboration_system_task ::=
        $fatal [ ( [ list_of_arguments ] ) ] ;
      | $error [ ( [ list_of_arguments ] ) ] ;
      | $warning [ ( [ list_of_arguments ] ) ] ;
      | $info [ ( [ list_of_arguments ] ) ] ;
```

Syntax 19-?? Elaboration system task syntax [Note to the Editor: Please assign appropriate number]

`list_of_arguments` may only contain a formatting string and constant expressions, including constant function calls. If a call to such a task remains in the elaborated model after any generate construct expansion, the task is executed. Depending on the severity of the task the elaboration may be aborted or continue to successful completion. If more than one elaboration system task call is present, they may be executed in any order.

If `$fatal` is executed then after outputting the message the elaboration may be aborted, and in no case shall simulation be executed. Some of the elaboration system task calls may not be executed either.

If `$error` is executed then the message is issued and the elaboration continues. However, no simulation shall be executed.

The other two tasks `$warning` and `$info` only output their text message but do not affect the rest of the elaboration and the simulation.

All of the elaboration system tasks shall print a tool-specific message, indicating the severity of the exception condition and specific information about the condition, which shall include the following information:

— The file name and line number of the elaboration system task call. The file name and line number shall be same as `` `__LINE__ `` and `` `__FILE__ `` compiler directives respectively.
— The hierarchical name of the scope in which the elaboration system task call is made.

The tool-specific message shall include the user-defined message if specified.

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

```
module test #(N = 1) (input [N-1:0] in, output [N-1:0] out);
   if ((N < 1) || (N > 8)) // conditional generate construct
      $error("Parameter N  has an invalid value of %0d", 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 system tasks are used to indicate if the vector is only a 1-bit vector, otherwise informational messages are issued that indicate which conditional branches were generated.

```
   generate
       if ($bits(vect) == 1) begin : err $error("Only a 1-bit  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 generated");
             end : Cond
             else begin : Cond
                    sequence t; vect[i] ##1 Loop[i-1].Cond.t; endsequence
                    $info("i = %0d branch generated", i);
             end : Cond
       end : Loop
   endgenerate
   // instantiate the last generated sequence in a property
   property p;
       @(posedge clk) trig |-> Loop[$bits(vect)-1].Cond.t;
   endproperty
```

In A.1.4 Module items

ADD right after the title

elaboration_system_task ::=
  $fatal [**(** [list_of_arguments] **)**] **;**
| **$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_system_task

In A.1.7 Program items

REPLACE

program_generate_item[36] ::=
loop_generate_construct
| conditional_generate_construct
| generate_region

WITH

program_generate_item[36] ::=
loop_generate_construct

| conditional_generate_construct
| generate_region
| elaboration_system_task