Annex A

(normative)

Formal syntax definition

The formal syntax of Verilog-AMS HDL is described using Backus-Naur Form (BNF). The syntax of Verilog-AMS HDL source is derived from the starting symbol source_text. The syntax of a library map file is derived from the starting symbol library_text. The following grammar is designed as a presentation grammar and should not be interpreted as an unambiguous production grammar. The compiler developer will be required to implement the various semantic restrictions outlined throughout this reference manual to remove any ambiguities.

BNF keywords and syntax tokens are denoted by **`red font`**. Specificic syntax items relevent to Verilog-AMS are denoted by blue font. Specific syntax items relevent to Analog-System Verilog Assertions are denoted by green font.

A.1 Source text

A.1.1 Library source text

library_text ::= { library_description }

library_description ::=
  library_declaration
| include_statement
| config_declaration

library_declaration ::=
  `library` library_identifier file_path_spec [{ **`,`** file_path_spec }]
  [**`-incdir`** file_path_spec { **`,`** file_path_spec }] **`;`**

file_path_spec ::= file_path

include_statement ::= **`include`** file_path_spec **`;`**

A.1.2 Verilog source text

source_text ::= { description }

description ::=
  module_declaration
| udp_declaration
| config_declaration
| paramset_declaration
| nature_declaration
| discipline_declaration
| connectrules_declaration

module_declaration ::=
  { attribute_instance } module_keyword module_identifier [module_parameter_port_list]
    list_of_ports **`;`** { module_item }
  `endmodule`
| { attribute_instance } module_keyword module_identifier [module_parameter_port_list]

[list_of_port_declarations] ; { non_port_module_item }
endmodule

module_keyword ::= module | macromodule | connectmodule

A.1.3 Module parameters and ports

module_parameter_port_list ::= # (parameter_declaration { , parameter_declaration })

list_of_ports ::= (port { , port })

list_of_port_declarations ::=
(port_declaration { , port_declaration })
| ( )

port ::=
[port_expression]
| . port_identifier ([port_expression])

port_expression ::=
port_reference
| { port_reference { , port_reference } }

port_reference ::=
port_identifier [[constant_range_expression]]

port_declaration ::=
{attribute_instance} inout_declaration
| {attribute_instance} input_declaration
| {attribute_instance} output_declaration

A.1.4 Module items

module_item ::=
port_declaration ;
| non_port_module_item

module_or_generate_item ::=
{ attribute_instance } module_or_generate_item_declaration
| { attribute_instance } local_parameter_declaration ;
| { attribute_instance } parameter_override
| { attribute_instance } continuous_assign
| { attribute_instance } gate_instantiation
| { attribute_instance } udp_instantiation
| { attribute_instance } module_instantiation
| { attribute_instance } initial_construct
| { attribute_instance } always_construct
| { attribute_instance } loop_generate_construct
| { attribute_instance } conditional_generate_construct
| { attribute_instance } analog_construct
| { attribute_instance } assertion_item

module_or_generate_item_declaration ::=
net_declaration
| reg_declaration
| integer_declaration
| real_declaration
| time_declaration
| realtime_declaration

| event_declaration
| genvar_declaration
| task_declaration
| function_declaration
| branch_declaration
| analog_function_declaration
| clocking_declaration
| **default clocking** clocking_identifier **;**
| **default disable iff** expression_or_dist **;**

non_port_module_item ::=
module_or_generate_item
| generate_region
| specify_block
| { attribute_instance } parameter_declaration **;**
| { attribute_instance } specparam_declaration
| aliasparam_declaration

parameter_override ::= **defparam** list_of_defparam_assignments **;**

A.1.5 Configuration source text

config_declaration ::=
config config_identifier **;**
design_statement
{config_rule_statement}
endconfig

design_statement ::= **design** { [library_identifier**.**]cell_identifier } **;**

config_rule_statement ::=
default_clause liblist_clause **;**
| inst_clause liblist_clause **;**
| inst_clause use_clause **;**
| cell_clause liblist_clause **;**
| cell_clause use_clause **;**

default_clause ::= **default**

inst_clause ::= **instance** inst_name

inst_name ::= topmodule_identifier{**.**instance_identifier}

cell_clause ::= **cell** [library_identifier**.**]cell_identifier

liblist_clause ::= **liblist** { library_identifier }

use_clause ::= **use** [library_identifier**.**]cell_identifier[**:config**]

A.1.6 Nature Declaration

nature_declaration ::=
nature nature_identifier [**:** parent_nature] [**;**]
{ nature_item }
endnature

parent_nature ::=
nature_identifier
| discipline_identifier **.** potential_or_flow

nature_item ::= nature_attribute

nature_attribute ::= nature_attribute_identifier **=** nature_attribute_expression **;**

A.1.7 Discipline Declaration

discipline_declaration ::=
    discipline discipline_identifier [**;**]
        { discipline_item }
    enddiscipline

discipline_item ::=
    nature_binding
  | discipline_domain_binding
  | nature_attribute_override

nature_binding ::= potential_or_flow nature_identifier **;**

potential_or_flow ::= **potential** | **flow**

discipline_domain_binding ::= **domain** discrete_or_continuous **;**

discrete_or_continuous ::= **discrete** | **continuous**

nature_attribute_override ::= potential_or_flow **.** nature_attribute

A.1.8 Connectrules Declaration

connectrules_declaration ::=
    connectrules connectrules_identifier **;**
        { connectrules_item }
    endconnectrules

connectrules_item ::=
    connect_insertion
  | connect_resolution

connect_insertion ::= **connect** connectmodule_identifier [connect_mode]
        [parameter_value_assignment] [connect_port_overrides] **;**

connect_mode ::= **merged** | **split**

connect_port_overrides ::=
    discipline_identifier **,** discipline_identifier
  | **input** discipline_identifier **,** **output** discipline_identifier
  | **output** discipline_identifier **,** **input** discipline_identifier
  | **inout** discipline_identifier **,** **inout** discipline_identifier

connect_resolution ::= **connect** discipline_identifier { **,** discipline_identifier } **resolveto**
    discipline_identifier
  | **exclude**
  ;

A.1.9 Paramset Declaration

paramset_declaration ::=
    { attribute_instance } **paramset** paramset_identifier module_or_paramset_identifier **;**
        paramset_item_declaration { paramset_item_declaration }
        paramset_statement { paramset_statement }
    endparamset

paramset_item_declaration ::=
    { attribute_instance } parameter_declaration **;**

| { attribute_instance } local_parameter_declaration ;
| aliasparam_declaration
| { attribute_instance } integer_declaration
| { attribute_instance } real_declaration

paramset_statement ::=
.module_parameter_identifier = paramset_constant_expression ;
| .system_parameter_identifier = paramset_constant_expression;
| analog_function_statement

paramset_constant_expression ::=
constant_primary
| hierarchical_parameter_identifier
| unary_operator { attribute_instance } constant_primary
| paramset_constant_expression binary_operator { attribute_instance } paramset_constant_expression
| paramset_constant_expression ? { attribute_instance } paramset_constant_expression :
paramset_constant_expression

A.2 Declarations

A.2.1 Declaration types

A.2.1.1 Module parameter declarations

local_parameter_declaration ::=
localparam [**signed**] [range] list_of_param_assignments
| **localparam** parameter_type list_of_param_assignments

parameter_declaration ::=
parameter [**signed**] [range] list_of_param_assignments
| **parameter** parameter_type list_of_param_assignments

specparam_declaration ::= **specparam** [range] list_of_specparam_assignments ;

parameter_type ::=
integer | **real** | **realtime** | **time** | **string**

aliasparam_declaration ::= **aliasparam** parameter_identifier = parameter_identifier ;

A.2.1.2 Port declarations

inout_declaration ::=
inout [discipline_identifier] [net_type | **wreal**] [**signed**] [range] list_of_port_identifiers

input_declaration ::=
input [discipline_identifier] [net_type | **wreal**] [**signed**] [range] list_of_port_identifiers

output_declaration ::=
output [discipline_identifier] [net_type | **wreal**] [**signed**] [range] list_of_port_identifiers
| **output** [discipline_identifier] **reg** [**signed**] [range] list_of_variable_port_identifiers
| **output** output_variable_type list_of_variable_port_identifiers

A.2.1.3 Type declarations

branch_declaration ::= **branch** (branch_terminal [, branch_terminal]) list_of_branch_identifiers ;

branch_terminal ::=
net_identifier
| net_identifier [constant_expression]
| net_identifier [constant_range_expression]

event_declaration ::= **event** list_of_event_identifiers ;

integer_declaration ::= **integer** list_of_variable_identifiers **;**

net_declaration ::=
net_type [discipline_identifier] [**signed**]
[delay3] list_of_net_identifiers **;**
| net_type [discipline_identifier] [drive_strength] [**signed**]
[delay3] list_of_net_decl_assignments **;**
| net_type [discipline_identifier] [**vectored** | **scalared**] [**signed**]
range [delay3] list_of_net_identifiers **;**
| net_type [discipline_identifier] [drive_strength] [**vectored** | **scalared**] [**signed**]
range [delay3] list_of_net_decl_assignments **;**
| **trireg** [discipline_identifier] [charge_strength] [**signed**]
[delay3] list_of_net_identifiers **;**
| **trireg** [discipline_identifier] [drive_strength] [**signed**]
[delay3] list_of_net_decl_assignments **;**
| **trireg** [discipline_identifier] [charge_strength] [**vectored** | **scalared**] [**signed**]
range [delay3] list_of_net_identifiers **;**
| **trireg** [discipline_identifier] [drive_strength] [**vectored** | **scalared**] [**signed**]
range [delay3] list_of_net_decl_assignments **;**
| discipline_identifier [range] list_of_net_identifiers **;**
| discipline_identifier [range] list_of_net_decl_assignments **;**
| **wreal** [discipline_identifier] [range] list_of_net_identifiers **;**
| **wreal** [discipline_identifier] [range] list_of_net_decl_assignments **;**
| **ground** [discipline_identifier] [range] list_of_net_identifiers **;**

real_declaration ::= **real** list_of_real_identifiers **;**

realtime_declaration ::= **realtime** list_of_real_identifiers **;**

reg_declaration ::= **reg** [discipline_identifier] [**signed**] [range]
list_of_variable_identifiers **;**

time_declaration ::= **time** list_of_variable_identifiers **;**

A.2.2 Declaration data types

A.2.2.1 Net and variable types

net_type ::=
supply0 | **supply1** |**tri** | **triand** | **trior** | **tri0** | **tri1** | **uwire** | **wire** | **wand** | **wor**

output_variable_type ::= **integer** | **time**

real_type ::=
real_identifier { dimension } [**=** constant_arrayinit]
| real_identifier **=** constant_expression

variable_type ::=
variable_identifier { dimension } [**=** constant_arrayinit]
| variable_identifier **=** constant_expression

A.2.2.2 Strengths

drive_strength ::=
(strength0 **,** strength1 **)**
| **(** strength1 **,** strength0 **)**
| **(** strength0 **,** **highz1** **)**
| **(** strength1 **,** **highz0** **)**
| **(** **highz0** **,** strength1 **)**
| **(** **highz1** **,** strength0 **)**

strength0 ::= **supply0** | **strong0** | **pull0** | **weak0**

strength1 ::= **supply1** | **strong1** | **pull1** | **weak1**

charge_strength ::= **(small)** | **(medium)** | **(large)**

A.2.2.3 Delays

delay3 ::=
    # delay_value
  | **# (** mintypmax_expression [**,** mintypmax_expression [**,** mintypmax_expression]] **)**

delay2 ::=
    # delay_value
  | **# (** mintypmax_expression [**,** mintypmax_expression] **)**

delay_value ::=
    unsigned_number
  | real_number
  | identifier

A.2.3 Declaration lists

list_of_branch_identifiers ::= branch_identifier [range] { **,** branch_identifier [range] }

list_of_defparam_assignments ::= defparam_assignment { **,** defparam_assignment }

list_of_event_identifiers ::= event_identifier { dimension } { **,** event_identifier { dimension } }

list_of_net_decl_assignments ::= net_decl_assignment { **,** net_decl_assignment }

list_of_net_identifiers ::= ams_net_identifier { **,** ams_net_identifier }

list_of_param_assignments ::= param_assignment { **,** param_assignment }

list_of_port_identifiers ::= port_identifier { **,** port_identifier }

list_of_real_identifiers ::= real_type { **,** real_type }

list_of_specparam_assignments ::= specparam_assignment { **,** specparam_assignment }

list_of_variable_identifiers ::= variable_type { **,** variable_type }

list_of_variable_port_identifiers ::= port_identifier [**=** constant_expression]
        { **,** port_identifier [**=** constant_expression] }

A.2.4 Declaration assignments

defparam_assignment ::= hierarchical_parameter_identifier **=** constant_mintypmax_expression

net_decl_assignment ::=
    net_identifier **=** expression
  | net_identifier { dimension } = constant_optional_arrayinit

ams_net_identifier ::=
    net_identifier { dimension }
  | hierarchical_net_identifier

param_assignment ::=
    parameter_identifier **=** constant_mintypmax_expression { value_range }
  | parameter_identifier range **=** constant_arrayinit { value_range }

specparam_assignment ::=
    specparam_identifier **=** constant_mintypmax_expression
  | pulse_control_specparam

pulse_control_specparam ::=
    PATHPULSE$ = (reject_limit_value [**,** error_limit_value] **)**

| **PATHPULSE$**specify_input_terminal_descriptor**$**specify_output_terminal_descriptor
= (reject_limit_value [**,** error_limit_value] **)**

error_limit_value ::= limit_value

reject_limit_value ::= limit_value

limit_value ::= constant_mintypmax_expression

A.2.5 Declaration ranges

dimension ::= **[** dimension_constant_expression **:** dimension_constant_expression **]**

range ::= **[** msb_constant_expression **:** lsb_constant_expression **]**

value_range ::=
value_range_type **(** value_range_expression **:** value_range_expression **)**
| value_range_type **(** value_range_expression **:** value_range_expression **]**
| value_range_type **[** value_range_expression **:** value_range_expression **)**
| value_range_type **[** value_range_expression **:** value_range_expression **]**
| value_range_type **'{** string { **,** string } **}**
| **exclude** constant_expression

value_range_type :: = **from** | **exclude**

value_range_expression ::= constant_expression | **-inf** | **inf**

variable_dimension ::=
dimension
| **[** dimension_constant_expression **]**

A.2.6 Function declarations

analog_function_declaration ::=
analog function [analog_function_type] analog_function_identifier **;**
analog_function_item_declaration { analog_function_item_declaration }
analog_function_statement
endfunction

analog_function_type ::= **integer** | **real**

analog_function_item_declaration ::=
analog_block_item_declaration
| input_declaration **;**
| output_declaration **;**
| inout_declaration **;**

function_declaration ::=
function [**automatic**] [function_range_or_type] function_identifier **;**
function_item_declaration { function_item_declaration }
function_statement
endfunction
| **function** [**automatic**] [function_range_or_type] function_identifier **(** function_port_list **) ;**
{ block_item_declaration }
function_statement
endfunction

function_item_declaration ::=
block_item_declaration
| { attribute_instance } tf_input_declaration **;**

function_port_list ::=

{ attribute_instance } tf_input_declaration { , { attribute_instance } tf_input_declaration }

function_range_or_type ::=
[signed] [range]
| integer
| real
| realtime
| time

A.2.7 Task declarations

task_declaration ::=
task [automatic] task_identifier ;
{ task_item_declaration }
statement_or_null
endtask
| task [automatic] task_identifier ([task_port_list]) ;
{ block_item_declaration }
statement_or_null
endtask

task_item_declaration ::=
block_item_declaration
| { attribute_instance } tf_input_declaration ;
| { attribute_instance } tf_output_declaration ;
| { attribute_instance } tf_inout_declaration ;

task_port_list ::= task_port_item { , task_port_item }

task_port_item ::=
{ attribute_instance } tf_input_declaration
| { attribute_instance } tf_output_declaration
| { attribute_instance } tf_inout_declaration

tf_input_declaration ::=
input [discipline_identifier] [reg] [signed] [range] list_of_port_identifiers
| input task_port_type list_of_port_identifiers

tf_output_declaration ::=
output [discipline_identifier] [reg] [signed] [range] list_of_port_identifiers
| output task_port_type list_of_port_identifiers

tf_inout_declaration ::=
inout [discipline_identifier] [reg] [signed] [range] list_of_port_identifiers
| inout task_port_type list_of_port_identifiers

task_port_type ::=
integer | real | realtime | time

A.2.8 Block item declarations

analog_block_item_declaration ::=
{ attribute_instance } parameter_declaration ;
| { attribute_instance } integer_declaration
| { attribute_instance } real_declaration

block_item_declaration ::=
{ attribute_instance } reg [discipline_identifier] [signed] [range]
list_of_block_variable_identifiers ;

| { attribute_instance } **integer** list_of_block_variable_identifiers **;**
| { attribute_instance } **time** list_of_block_variable_identifiers **;**
| { attribute_instance } **real** list_of_block_real_identifiers **;**
| { attribute_instance } **realtime** list_of_block_real_identifiers **;**
| { attribute_instance } event_declaration
| { attribute_instance } local_parameter_declaration **;**
| { attribute_instance } parameter_declaration **;**

list_of_block_variable_identifiers ::= block_variable_type { **,** block_variable_type }

list_of_block_real_identifiers ::= block_real_type { **,** block_real_type }

block_variable_type ::= variable_identifier { dimension }

block_real_type ::= real_identifier { dimension }

A.2.9 Assertion declarations

concurrent_assertion_item ::=
[block_identifier **:**] concurrent_assertion_statement
| checker_instantiation

concurrent_assertion_statement ::=
assert_property_statement
| assume_property_statement
| cover_property_statement
| cover_sequence_statement
| restrict_property_statement

assert_property_statement::=
assert property (property_spec **)** action_block

assume_property_statement::=
assume property (property_spec **)** action_block

cover_property_statement::=
cover property (property_spec **)** statement_or_null

expect_property_statement ::=
expect (property_spec **)** action_block

cover_sequence_statement::=
cover sequence ([clocking_event] [**disable iff (** expression_or_dist **)**]
sequence_expr **)** statement_or_null

restrict_property_statement::=
restrict property (property_spec **) ;**

property_instance ::=
hierarchical_property_identifier [**(** [property_list_of_arguments] **)**]

property_list_of_arguments ::=
[property_actual_arg] { **,** [property_actual_arg] } { **, .** identifier **(** [property_actual_arg] **)** }
| **.** identifier **(** [property_actual_arg] **)** { **, .** identifier **(** [property_actual_arg] **)** }

property_actual_arg ::=
property_expr
| sequence_actual_arg

assertion_item_declaration ::=
property_declaration
| sequence_declaration
| let_declaration

property_declaration ::=

```
        property property_identifier [ ( [ property_port_list ] ) ] ;
            { assertion_variable_declaration }
             property_statement_spec
        endproperty [ : property_identifier ]
property_port_list ::=
        property_port_item {, property_port_item}
property_port_item ::=
        { attribute_instance } [ local [ property_lvar_port_direction ] ] property_formal_type
            port_identifier {variable_dimension} [ = property_actual_arg ]
property_lvar_port_direction ::= input
property_formal_type ::=
        sequence_formal_type
      | property
property_spec ::=
        [clocking_event ] [ disable iff ( expression_or_dist ) ] property_expr
property_statement_spec ::=
        [ clocking_event ] [ disable iff ( expression_or_dist ) ] property_statement
property_statement ::=
        property_expr ;
      | case ( expression_or_dist ) property_case_item { property_case_item } endcase
      | if ( expression_or_dist ) property_expr [ else property_expr ]
property_case_item::=
        expression_or_dist { , expression_or_dist } : property_statement
      | default [ : ] property_statement
property_expr ::=
        sequence_expr
      | strong ( sequence_expr )
      | weak ( sequence_expr )
      | ( property_expr )
      | not property_expr
      | property_expr or property_expr
      | property_expr and property_expr
      | sequence_expr |-> property_expr
      | sequence_expr |=> property_expr
      | property_statement
      | sequence_expr #-# property_expr
      | sequence_expr #=# property_expr
      | nexttime property_expr
      | nexttime [ constant_expression ] property_expr
      | s_nexttime property_expr
      | s_nexttime [ constant_expression ] property_expr
      | always property_expr
      | always [ cycle_delay_const_range_expression ] property_expr
      | s_always [ constant_range ] property_expr
      | s_eventually property_expr
      | eventually [ constant_range ] property_expr
      | s_eventually [ cycle_delay_const_range_expression ] property_expr
      | property_expr until property_expr
      | property_expr s_until property_expr
      | property_expr until_with property_expr
      | property_expr s_until_with property_expr
```

```
        | property_expr implies property_expr
        | property_expr iff property_expr
        | accept_on ( expression_or_dist ) property_expr
        | reject_on ( expression_or_dist ) property_expr
        | sync_accept_on ( expression_or_dist ) property_expr
        | sync_reject_on ( expression_or_dist ) property_expr
        | property_instance
        | clocking_event property_expr
sequence_declaration ::=
        sequence sequence_identifier [ ( [ sequence_port_list ] ) ] ;
            { assertion_variable_declaration }
            sequence_expr ;
        endsequence [ : sequence_identifier ]
sequence_port_list ::=
        sequence_port_item {, sequence_port_item}
sequence_port_item ::=
        { attribute_instance } [ local [ sequence_lvar_port_direction ] ] sequence_formal_type
            port_identifier {variable_dimension} [ = sequence_actual_arg ]
sequence_lvar_port_direction ::= input | inout | output
sequence_formal_type ::=
        data_type_or_implicit
      | sequence
      | event
      | untyped
sequence_expr ::=
        cycle_delay_range sequence_expr { cycle_delay_range sequence_expr }
      | sequence_expr cycle_delay_range sequence_expr { cycle_delay_range sequence_expr }
      | expression_or_dist [ boolean_abbrev ]
      | sequence_instance [ sequence_abbrev ]
      | ( sequence_expr {, sequence_match_item } ) [ sequence_abbrev ]
      | sequence_expr and sequence_expr
      | sequence_expr intersect sequence_expr
      | sequence_expr or sequence_expr
      | first_match ( sequence_expr {, sequence_match_item} )
      | expression_or_dist throughout sequence_expr
      | sequence_expr within sequence_expr
      | clocking_event sequence_expr
cycle_delay_range ::=
        ## constant_primary
      | ## [ cycle_delay_const_range_expression ]
      | ##[*]
      | ##[+]
sequence_method_call ::=
        sequence_instance . method_identifier
sequence_match_item ::=
        variable_assignment
      | operator_assignment
      | inc_or_dec_expression
      | subroutine_call
sequence_instance ::=
         hierarchical_sequence_identifier [ ( [ sequence_list_of_arguments ] ) ]
```

```
sequence_list_of_arguments ::=
        [sequence_actual_arg] { , [sequence_actual_arg] } { , . identifier ( [sequence_actual_arg] ) }
      | . identifier ( [sequence_actual_arg] ) { , . identifier ( [sequence_actual_arg] ) }
sequence_actual_arg ::=
        event_expression
      | sequence_expr
boolean_abbrev ::=
        consecutive_repetition
      | non_consecutive_repetition
      | goto_repetition
sequence_abbrev ::= consecutive_repetition
consecutive_repetition ::=
        [* const_or_range_expression ]
      | [*]
      | [+]
non_consecutive_repetition ::= [= const_or_range_expression ]
goto_repetition ::= [-> const_or_range_expression ]
const_or_range_expression ::=
        constant_expression
      | cycle_delay_const_range_expression
cycle_delay_const_range_expression ::=
        constant_expression : constant_expression
      | constant_expression : $
expression_or_dist ::= expression [ dist { dist_list } ]
assertion_variable_declaration ::=
        var_data_type list_of_variable_identifiers ;
let_declaration ::=
        let let_identifier [ ( [ let_port_list ] ) ] = expression ;
let_identifier ::=
        identifier
let_port_list ::=
        let_port_item {, let_port_item}
let_port_item ::=
        { attribute_instance } let_formal_type port_identifier { variable_dimension } [ = expression ]
let_formal_type ::=
        data_type_or_implicit
let_expression ::=
        [ package_scope ] let_identifier [ ( [ let_list_of_arguments ] ) ]
let_list_of_arguments ::=
        [ let_actual_arg ] {, [ let_actual_arg ] } {, . identifier ( [ let_actual_arg ] ) }
      | . identifier ( [ let_actual_arg ] ) { , . identifier ( [ let_actual_arg ] ) }
let_actual_arg ::=
        expression
```

A.3 Primitive instances

A.3.1 Primitive instantiation and instances

gate_instantiation ::=
cmos_switchtype [delay3] cmos_switch_instance { , cmos_switch_instance } ;
| enable_gatetype [drive_strength] [delay3] enable_gate_instance { , enable_gate_instance } ;
| mos_switchtype [delay3] mos_switch_instance { , mos_switch_instance } ;
| n_input_gatetype [drive_strength] [delay2] n_input_gate_instance { , n_input_gate_instance } ;
| n_output_gatetype [drive_strength] [delay2] n_output_gate_instance { , n_output_gate_instance } ;
| pass_en_switchtype [delay2] pass_enable_switch_instance { , pass_enable_switch_instance } ;
| pass_switchtype pass_switch_instance { , pass_switch_instance } ;
| **pulldown** [pulldown_strength] pull_gate_instance { , pull_gate_instance } ;
| **pullup** [pullup_strength] pull_gate_instance { , pull_gate_instance } ;

cmos_switch_instance ::= [name_of_gate_instance] (output_terminal , input_terminal ,
ncontrol_terminal , pcontrol_terminal)

enable_gate_instance ::= [name_of_gate_instance] (output_terminal , input_terminal , enable_terminal)

mos_switch_instance ::= [name_of_gate_instance] (output_terminal , input_terminal , enable_terminal)

n_input_gate_instance ::= [name_of_gate_instance] (output_terminal , input_terminal
{ , input_terminal })

n_output_gate_instance ::= [name_of_gate_instance] (output_terminal { , output_terminal } ,
input_terminal)

pass_switch_instance ::= [name_of_gate_instance] (inout_terminal , inout_terminal)

pass_enable_switch_instance ::= [name_of_gate_instance] (inout_terminal , inout_terminal ,
enable_terminal)

pull_gate_instance ::= [name_of_gate_instance] (output_terminal)

name_of_gate_instance ::= gate_instance_identifier [range]

A.3.2 Primitive strengths

pulldown_strength ::=
(strength0 , strength1)
| (strength1 , strength0)
| (strength0)

pullup_strength ::=
(strength0 , strength1)
| (strength1 , strength0)
| (strength1)

A.3.3 Primitive terminals

enable_terminal ::= expression

inout_terminal ::= net_lvalue

input_terminal ::= expression

ncontrol_terminal ::= expression

output_terminal ::= net_lvalue

pcontrol_terminal ::= expression

A.3.4 Primitive gate and switch types

cmos_switchtype ::= **cmos** | **rcmos**

enable_gatetype ::= **bufif0** | **bufif1** | **notif0** | **notif1**

mos_switchtype ::= **nmos** | **pmos** | **rnmos** | **rpmos**

n_input_gatetype ::= **and** | **nand** | **or** | **nor** | **xor** | **xnor**

n_output_gatetype ::= **buf** | **not**

pass_en_switchtype ::= **tranif0** | **tranif1** | **rtranif1** | **rtranif0**

pass_switchtype ::= **tran** | **rtran**

A.4 Module instantiation and generate construct

A.4.1 Module instantiation

module_instantiation ::=
    module_or_paramset_identifier [parameter_value_assignment]
        module_instance { , module_instance } ;

parameter_value_assignment ::= # (list_of_parameter_assignments)

list_of_parameter_assignments ::=
    ordered_parameter_assignment { , ordered_parameter_assignment }
  | named_parameter_assignment { , named_parameter_assignment }

ordered_parameter_assignment ::= expression

named_parameter_assignment ::=
    . parameter_identifier ([mintypmax_expression])
  | . system_parameter_identifier ([constant_expression])

module_instance ::= name_of_module_instance ([list_of_port_connections])

name_of_module_instance ::= module_instance_identifier [range]

list_of_port_connections ::=
    ordered_port_connection { , ordered_port_connection }
  | named_port_connection { , named_port_connection }

ordered_port_connection ::= { attribute_instance } [expression]

named_port_connection ::= { attribute_instance } . port_identifier ([expression])

A.4.2 Generate construct

generate_region ::=
    generate { module_or_generate_item } **endgenerate**

genvar_declaration ::=
    genvar list_of_genvar_identifiers ;

list_of_genvar_identifiers ::=
    genvar_identifier { , genvar_identifier }

analog_loop_generate_statement ::=
    for (genvar_initialization ; genvar_expression ; genvar_iteration)
        analog_statement

loop_generate_construct ::=
    for (genvar_initialization ; genvar_expression ; genvar_iteration)
        generate_block

genvar_initialization ::=
    genvar_identifier = constant_expression

genvar_expression ::=
    genvar_primary

| unary_operator { attribute_instance } genvar_primary
| genvar_expression binary_operator { attribute_instance } genvar_expression
| genvar_expression **?** { attribute_instance } genvar_expression **:** genvar_expression

genvar_iteration ::=
genvar_identifier **=** genvar_expression

genvar_primary ::=
constant_primary
| genvar_identifier

conditional_generate_construct ::=
if_generate_construct
| case_generate_construct

if_generate_construct ::=
if (constant_expression **)** generate_block_or_null
[**else** generate_block_or_null]

case_generate_construct ::=
case (constant_expression **)** case_generate_item { case_generate_item } **endcase**

case_generate_item ::=
constant_expression { **,** constant_expression } **:** generate_block_or_null
| **default** [**:**] generate_block_or_null

generate_block ::=
module_or_generate_item
| **begin** [**:** generate_block_identifier] { module_or_generate_item } **end**

generate_block_or_null ::=
generate_block
| **;**

A.4.3 Checker instantiation

checker_instantiation ::=
checker_identifier name_of_module_instance **(** [list_of_checker_port_connections] **) ;**

list_of_checker_port_connections ::=
ordered_checker_port_connection { **,** ordered_checker_port_connection }
| named_checker_port_connection { **,** named_checker_port_connection }

ordered_checker_port_connection ::= { attribute_instance } [property_actual_arg]

named_checker_port_connection ::=
{ attribute_instance } **.** port_identifier [**(** [property_actual_arg] **)**]
| { attribute_instance } **.***

A.5 UDP declaration and instantiation

A.5.1 UDP declaration

udp_declaration ::=
{ attribute_instance } **primitive** udp_identifier **(** udp_port_list **) ;**
udp_port_declaration { udp_port_declaration }
udp_body
endprimitive
| { attribute_instance } **primitive** udp_identifier **(** udp_declaration_port_list **) ;**
udp_body

```
endprimitive
```

A.5.2 UDP ports

udp_port_list ::= output_port_identifier , input_port_identifier { , input_port_identifier }
udp_declaration_port_list ::=
    udp_output_declaration , udp_input_declaration { , udp_input_declaration }
udp_port_declaration ::=
    udp_output_declaration ;
  | udp_input_declaration ;
  | udp_reg_declaration ;
udp_output_declaration ::=
    { attribute_instance } output port_identifier
  | { attribute_instance } output [discipline_identifier] reg port_identifier [= constant_expression]
udp_input_declaration ::= { attribute_instance } input list_of_port_identifiers
udp_reg_declaration ::= { attribute_instance } reg [discipline_identifier] variable_identifier

A.5.3 UDP body

udp_body ::= combinational_body | sequential_body
combinational_body ::= table combinational_entry { combinational_entry } endtable
combinational_entry ::= level_input_list : output_symbol ;
sequential_body ::= [udp_initial_statement] table sequential_entry { sequential_entry } endtable
udp_initial_statement ::= initial output_port_identifier = init_val ;
init_val ::= 1'b0 | 1'b1 | 1'bx | 1'bX | 1'B0 | 1'B1 | 1'Bx | 1'BX | 1 | 0
sequential_entry ::= seq_input_list : current_state : next_state ;
seq_input_list ::= level_input_list | edge_input_list
level_input_list ::= level_symbol { level_symbol }
edge_input_list ::= { level_symbol } edge_indicator { level_symbol }
edge_indicator ::= (level_symbol level_symbol) | edge_symbol
current_state ::= level_symbol
next_state ::= output_symbol | -
output_symbol ::= 0 | 1 | x | X
level_symbol ::= 0 | 1 | x | X | ? | b | B
edge_symbol ::= r | R | f | F | p | P | n | N | *

A.5.4 UDP instantiation

udp_instantiation ::= udp_identifier [drive_strength] [delay2] udp_instance { , udp_instance } ;
udp_instance ::= [name_of_udp_instance] (output_terminal , input_terminal { , input_terminal })
name_of_udp_instance ::= udp_instance_identifier [range]

A.6 Behavioral statements

A.6.1 Continuous assignment statements

continuous_assign ::= **assign** [drive_strength] [delay3] list_of_net_assignments **;**

list_of_net_assignments ::= net_assignment { **,** net_assignment }

net_assignment ::= net_lvalue **=** expression

A.6.2 Procedural blocks and assignments

analog_construct ::=
    analog analog_statement
  | **analog** **initial** analog_function_statement

analog_procedural_assignment ::= analog_variable_assignment **;**

analog_variable_assignment ::= analog_variable_lvalue **=** analog_expression

initial_construct ::= **initial** statement

always_construct ::= **always** statement

blocking_assignment ::= variable_lvalue **=** [delay_or_event_control] expression

nonblocking_assignment ::= variable_lvalue **<=** [delay_or_event_control] expression

procedural_continuous_assignments ::=
    assign variable_assignment
  | **deassign** variable_lvalue
  | **force** variable_assignment
  | **force** net_assignment
  | **release** variable_lvalue
  | **release** net_lvalue

variable_assignment ::= variable_lvalue **=** expression

A.6.3 Parallel and sequential blocks

analog_seq_block ::= **begin** [**:** analog_block_identifier { analog_block_item_declaration }]
    { analog_statement } **end**

analog_event_seq_block ::=
    begin [**:** analog_block_identifier { analog_block_item_declaration }]
    { analog_event_statement} **end**

analog_function_seq_block ::= **begin** [**:** analog_block_identifier { analog_block_item_declaration }]
    { analog_function_statement } **end**

par_block ::= **fork** [**:** block_identifier
    { block_item_declaration }] { statement } **join**

seq_block ::= **begin** [**:** block_identifier
    { block_item_declaration }] { statement } **end**

action_block ::=
    statement_or_null
  | [statement] **else** statement_or_null

A.6.4 Statements

analog_statement ::=
    { attribute_instance } analog_loop_generate_statement
  | { attribute_instance } analog_loop_statement
  | { attribute_instance } analog_case_statement
  | { attribute_instance } analog_conditional_statement

```
        | { attribute_instance } analog_procedural_assignment
        | { attribute_instance } analog_seq_block
        | { attribute_instance } analog_system_task_enable
        | { attribute_instance } contribution_statement
        | { attribute_instance } indirect_contribution_statement
        | { attribute_instance } analog_event_control_statement
analog_statement_or_null ::=
          analog_statement
        | { attribute_instance } ;
analog_event_statement ::=
          { attribute_instance } analog_loop_statement
        | { attribute_instance } analog_case_statement
        | { attribute_instance } analog_conditional_statement
        | { attribute_instance } analog_procedural_assignment
        | { attribute_instance } analog_event_seq_block
        | { attribute_instance } analog_system_task_enable
        | { attribute_instance } disable_statement
        | { attribute_instance } event_trigger
        | { attribute_instance } ;
analog_function_statement ::=
          { attribute_instance } analog_function_case_statement
        | { attribute_instance } analog_function_conditional_statement
        | { attribute_instance } analog_function_loop_statement
        | { attribute_instance } analog_function_seq_block
        | { attribute_instance } analog_procedural_assignment
        | { attribute_instance } analog_system_task_enable
analog_function_statement_or_null ::=
          analog_function_statement
        | { attribute_instance } ;
statement ::=
          { attribute_instance } blocking_assignment ;
        | { attribute_instance } case_statement
        | { attribute_instance } conditional_statement
        | { attribute_instance } disable_statement
        | { attribute_instance } event_trigger
        | { attribute_instance } loop_statement
        | { attribute_instance } nonblocking_assignment ;
        | { attribute_instance } par_block
        | { attribute_instance } procedural_continuous_assignments ;
        | { attribute_instance } procedural_timing_control_statement
        | { attribute_instance } seq_block
        | { attribute_instance } system_task_enable
        | { attribute_instance } task_enable
        | { attribute_instance } wait_statement
        | { attribute_instance } procedural_assertion_statement
        | { attribute_instance } expect_property_statement
        | { attribute_instance } clocking_drive
statement_or_null ::=
          statement
        | { attribute_instance } ;
function_statement[1] ::= statement
```

A.6.5 Timing control statements

```
analog_event_control_statement ::= analog_event_control analog_event_statement

analog_event_control ::=
        @ hierarchical_event_identifier
      | @ ( analog_event_expression )

analog_event_expression ::=
        expression
      | posedge expression
      | negedge expression
      | hierarchical_event_identifier
      | initial_step [ ( " analysis_identifier "{ , " analysis_identifier " } ) ]
      | final_step [ ( " analysis_identifier " { , " analysis_identifier " } ) ]
      | analog_event_functions
      | analog_event_expression or analog_event_expression
      | analog_event_expression , analog_event_expression

analog_event_functions ::=
        cross ( analog_expression [ , analog_expression_or_null
            [ , constant_expression_or_null [ , constant_expression_or_null [ , analog_expression ] ] ] ] )
      | above ( analog_expression [ , constant_expression_or_null
            [ , constant_expression_or_null [ , analog_expression ] ] ] )
      | timer ( analog_expression [ , analog_expression_or_null
            [ , constant_expression_or_null [ , analog_expression ] ] ] )

delay_control ::=
        # delay_value
      | # ( mintypmax_expression )

delay_or_event_control ::=
        delay_control
      | event_control
      | repeat ( expression ) event_control

disable_statement ::=
        disable hierarchical_task_identifier ;
      | disable hierarchical_block_identifier ;

event_control ::=
        @ hierarchical_event_identifier
      | @ ( event_expression )
      | @*
      | @ (*)

event_trigger ::=
        -> hierarchical_event_identifier { [ expression ] } ;

event_expression ::=
        expression
      | posedge expression
      | negedge expression
      | hierarchical_event_identifier
      | event_expression or event_expression
      | event_expression , event_expression
      | analog_event_functions
      | driver_update expression
      | analog_variable_lvalue

procedural_timing_control ::=
```

```
        delay_control
      | event_control
procedural_timing_control_statement ::=
        procedural_timing_control statement_or_null
wait_statement ::=
        wait ( expression ) statement_or_null
```

A.6.6 Conditional statements

```
analog_conditional_statement ::=
        if ( analog_expression ) analog_statement_or_null
        { else if ( analog_expression ) analog_statement_or_null }
        [ else analog_statement_or_null ]
analog_function_conditional_statement ::=
        if ( analog_expression ) analog_function_statement_or_null
        { else if ( analog_expression ) analog_function_statement_or_null }
        [ else analog_function_statement_or_null ]
conditional_statement ::=
        if ( expression )
          statement_or_null
        [ else statement_or_null ]
      | if_else_if_statement
if_else_if_statement ::=
        if ( expression ) statement_or_null
        { else if ( expression ) statement_or_null }
        [ else statement_or_null ]
```

A.6.7 Case statements

```
analog_case_statement ::=
        case ( analog_expression ) analog_case_item { analog_case_item } endcase
      | casex ( analog_expression ) analog_case_item { analog_case_item } endcase
      | casez ( analog_expression ) analog_case_item { analog_case_item } endcase
analog_case_item ::=
        analog_expression { , analog_expression } : analog_statement_or_null
      | default [ : ] analog_statement_or_null
analog_function_case_statement ::=
        case ( analog_expression ) analog_function_case_item {analog_function_case_item} endcase
analog_function_case_item ::=
        analog_expression { analog_expression } : analog_function_statement_or_null
      | default [ : ] analog_function_statement_or_null
case_statement ::=
        case ( expression ) case_item { case_item } endcase
      | casez ( expression ) case_item { case_item } endcase
      | casex ( expression ) case_item { case_item } endcase
case_item ::=
        expression { , expression } : statement_or_null
      | default [ : ] statement_or_null
```

A.6.8 Looping statements

analog_loop_statement ::=
    repeat **(** analog_expression **)** analog_statement
  | **while** **(** analog_expression **)** analog_statement
  | **for** **(** analog_variable_assignment **;** analog_expression **;** analog_variable_assignment **)**
      analog_statement

analog_function_loop_statement ::=
    repeat **(** analog_expression **)** analog_function_statement
  | **while** **(** analog_expression **)** analog_function_statement
  | **for** **(** analog_variable_assignment **;** analog_expression **;** analog_variable_assignment **)**
      analog_function_statement

loop_statement ::=
    forever statement
  | **repeat** **(** expression **)** statement
  | **while** **(** expression **)** statement
  | **for** **(** variable_assignment **;** expression **;** variable_assignment **)** statement

A.6.9 Task enable statements

analog_system_task_enable ::=
    analog_system_task_identifier [**(** [analog_expression] { **,** [analog_expression] } **)**] **;**

system_task_enable ::= system_task_identifier [**(** [expression] { **,** [expression] } **)**] **;**
    task_enable ::= hierarchical_task_identifier [**(** expression { **,** expression } **)**] **;**

A.6.10 Contribution statements

contribution_statement ::= branch_lvalue **<+** analog_expression **;**

indirect_contribution_statement ::= branch_lvalue **:** indirect_expression **==** analog_expression **;**

A.6.11 Assertion statements

assertion_item ::=
    concurrent_assertion_item
  | deferred_immediate_assertion_item

deferred_immediate_assertion_item ::= [block_identifier **:**] deferred_immediate_assertion_statement

procedural_assertion_statement ::=
    concurrent_assertion_statement
  | immediate_assertion_statement
  | checker_instantiation

immediate_assertion_statement ::=
    simple_immediate_assertion_statement
  | deferred_immediate_assertion_statement

simple_immediate_assertion_statement ::=
    simple_immediate_assert_statement
  | simple_immediate_assume_statement
  | simple_immediate_cover_statement

simple_immediate_assert_statement ::=
    assert **(** expression **)** action_block

```
simple_immediate_assume_statement ::=
      assume ( expression ) action_block
simple_immediate_cover_statement ::=
      cover ( expression ) statement_or_null
deferred_immediate_assertion_statement ::=
        deferred_immediate_assert_statement
      | deferred_immediate_assume_statement
      | deferred_immediate_cover_statement
deferred_immediate_assert_statement ::=
      assert #0 ( expression ) action_block
deferred_immediate_assume_statement ::=
      assume #0 ( expression ) action_block
deferred_immediate_cover_statement ::=
      cover #0 ( expression ) statement_or_null
```

A.6.12 Clocking block

```
clocking_declaration ::= [ default ] clocking [ clocking_identifier ] clocking_event ;
            { clocking_item }
        endclocking [ : clocking_identifier ]
      | global clocking [ clocking_identifier ] clocking_event ;
          endclocking [ : clocking_identifier ]
clocking_event ::=
        @ identifier
      | @ ( event_expression )
clocking_item ::=
        default default_skew ;
      | clocking_direction list_of_clocking_decl_assign ;
      | { attribute_instance } assertion_item_declaration
default_skew ::=
        input clocking_skew
      | output clocking_skew
      | input clocking_skew output clocking_skew
clocking_direction ::=
        input [ clocking_skew ]
      | output [ clocking_skew ]
      | input [ clocking_skew ] output [ clocking_skew ]
      | inout
list_of_clocking_decl_assign ::= clocking_decl_assign { , clocking_decl_assign }
clocking_decl_assign ::= signal_identifier [ = expression ]
clocking_skew ::=
        edge_identifier [ delay_control ]
      | delay_control
clocking_drive ::=
        clockvar_expression <= [ cycle_delay ] expression
cycle_delay ::=
        ## integral_number
      | ##  identifier
      | ## ( expression )
```

clockvar ::= hierarchical_identifier

clockvar_expression ::=
    clockvar hierarchical_identifier [{ **[** expression **]** } **[** range_expression **]**]

A.7 Specify section

A.7.1 Specify block declaration

specify_block ::= **specify** { specify_item } **endspecify**

specify_item ::=
    specparam_declaration
  | pulsestyle_declaration
  | showcancelled_declaration
  | path_declaration
  | system_timing_check

pulsestyle_declaration ::=
    pulsestyle_onevent list_of_path_outputs **;**
  | **pulsestyle_ondetect** list_of_path_outputs **;**

showcancelled_declaration ::=
    showcancelled list_of_path_outputs **;**
  | **noshowcancelled** list_of_path_outputs **;**

A.7.2 Specify path declarations

path_declaration ::=
    simple_path_declaration **;**
  | edge_sensitive_path_declaration **;**
  | state_dependent_path_declaration **;**

simple_path_declaration ::=
    parallel_path_description **=** path_delay_value
  | full_path_description **=** path_delay_value

parallel_path_description ::=
    (specify_input_terminal_descriptor [polarity_operator] **=>** specify_output_terminal_descriptor **)**

full_path_description ::=
    (list_of_path_inputs [polarity_operator] ***>** list_of_path_outputs **)**

list_of_path_inputs ::=
    specify_input_terminal_descriptor { **,** specify_input_terminal_descriptor }

list_of_path_outputs ::=
    specify_output_terminal_descriptor { **,** specify_output_terminal_descriptor }

A.7.3 Specify block terminals

specify_input_terminal_descriptor ::=
    input_identifier [**[** constant_range_expression **]**]

specify_output_terminal_descriptor ::=
    output_identifier [**[** constant_range_expression **]**]

input_identifier ::= input_port_identifier | inout_port_identifier

output_identifier ::= output_port_identifier | inout_port_identifier

A.7.4 Specify path delays

path_delay_value ::=
    list_of_path_delay_expressions
  | **(** list_of_path_delay_expressions **)**

list_of_path_delay_expressions ::=
    t_path_delay_expression
  | trise_path_delay_expression **,** tfall_path_delay_expression
  | trise_path_delay_expression **,** tfall_path_delay_expression **,** tz_path_delay_expression
  | t01_path_delay_expression **,** t10_path_delay_expression **,** t0z_path_delay_expression **,**
      tz1_path_delay_expression **,** t1z_path_delay_expression **,** tz0_path_delay_expression
  | t01_path_delay_expression **,** t10_path_delay_expression **,** t0z_path_delay_expression **,**
      tz1_path_delay_expression **,** t1z_path_delay_expression **,** tz0_path_delay_expression **,**
    t0x_path_delay_expression **,** tx1_path_delay_expression **,** t1x_path_delay_expression **,**
    tx0_path_delay_expression **,** txz_path_delay_expression **,** tzx_path_delay_expression

t_path_delay_expression ::= path_delay_expression

trise_path_delay_expression ::= path_delay_expression

tfall_path_delay_expression ::= path_delay_expression

tz_path_delay_expression ::= path_delay_expression

t01_path_delay_expression ::= path_delay_expression

t10_path_delay_expression ::= path_delay_expression

t0z_path_delay_expression ::= path_delay_expression

tz1_path_delay_expression ::= path_delay_expression

t1z_path_delay_expression ::= path_delay_expression

tz0_path_delay_expression ::= path_delay_expression

t0x_path_delay_expression ::= path_delay_expression

tx1_path_delay_expression ::= path_delay_expression

t1x_path_delay_expression ::= path_delay_expression

tx0_path_delay_expression ::= path_delay_expression

txz_path_delay_expression ::= path_delay_expression

tzx_path_delay_expression ::= path_delay_expression

path_delay_expression ::= constant_mintypmax_expression

edge_sensitive_path_declaration ::=
    parallel_edge_sensitive_path_description **=** path_delay_value
  | full_edge_sensitive_path_description **=** path_delay_value

parallel_edge_sensitive_path_description ::=
    ([edge_identifier] specify_input_terminal_descriptor **=>**
      (specify_output_terminal_descriptor [polarity_operator] **:** data_source_expression **))**

full_edge_sensitive_path_description ::=
    ([edge_identifier] list_of_path_inputs ***>**
      (list_of_path_outputs [polarity_operator] **:** data_source_expression **))**

data_source_expression ::= expression

edge_identifier ::= **posedge** | **negedge**

state_dependent_path_declaration ::=
    if (module_path_expression **)** simple_path_declaration
  | **if (** module_path_expression **)** edge_sensitive_path_declaration
  | **ifnone** simple_path_declaration

polarity_operator ::= **+** | **-**

A.7.5 System timing checks

A.7.5.1 System timing check commands

system_timing_check ::=
$setup_timing_check
| $hold_timing_check
| $setuphold_timing_check
| $recovery_timing_check
| $removal_timing_check
| $recrem_timing_check
| $skew_timing_check
| $timeskew_timing_check
| $fullskew_timing_check
| $period_timing_check
| $width_timing_check
| $nochange_timing_check

$setup_timing_check ::=
$setup (data_event **,** reference_event **,** timing_check_limit [**,** [notifier]] **) ;**

$hold_timing_check ::=
$hold (reference_event **,** data_event **,** timing_check_limit [**,** [notifier]] **) ;**

$setuphold_timing_check ::=
$setuphold (reference_event **,** data_event **,** timing_check_limit **,** timing_check_limit
[**,** [notifier] [**,** [stamptime_condition] [**,** [checktime_condition]
[**,** [delayed_reference] [**,** [delayed_data]]]]]] **) ;**

$recovery_timing_check ::=
$recovery (reference_event **,** data_event **,** timing_check_limit [**,** [notifier]] **) ;**

$removal_timing_check ::=
$removal (reference_event **,** data_event **,** timing_check_limit [**,** [notifier]] **) ;**

$recrem_timing_check ::=
$recrem (reference_event **,** data_event **,** timing_check_limit **,** timing_check_limit
[**,** [notifier] [**,** [stamptime_condition] [**,** [checktime_condition]
[**,** [delayed_reference] [**,** [delayed_data]]]]]] **) ;**

$skew_timing_check ::=
$skew (reference_event **,** data_event **,** timing_check_limit [**,** [notifier]] **) ;**

$timeskew_timing_check ::=
$timeskew (reference_event **,** data_event **,** timing_check_limit
[**,** [notifier] [**,** [event_based_flag] [**,** [remain_active_flag]]]] **) ;**

$fullskew_timing_check ::=
$fullskew (reference_event **,** data_event **,** timing_check_limit **,** timing_check_limit
[**,** [notifier] [**,** [event_based_flag] [**,** [remain_active_flag]]]] **) ;**

$period_timing_check ::=
$period (controlled_reference_event **,** timing_check_limit [**,** [notifier]] **) ;**

$width_timing_check ::=
$width (controlled_reference_event **,** timing_check_limit [**,** threshold [**,** notifier]] **) ;**

$nochange_timing_check ::=
$nochange (reference_event **,** data_event **,** start_edge_offset **,**
end_edge_offset [**,** [notifier]] **) ;**

A.7.5.2 System timing check command arguments

checktime_condition ::= mintypmax_expression

controlled_reference_event ::= controlled_timing_check_event

data_event ::= timing_check_event

delayed_data ::=
    terminal_identifier
  | terminal_identifier **[** constant_mintypmax_expression **]**

delayed_reference ::=
    terminal_identifier
  | terminal_identifier **[** constant_mintypmax_expression **]**

end_edge_offset ::= mintypmax_expression

event_based_flag ::= constant_expression

notifier ::= variable_identifier

reference_event ::= timing_check_event

remain_active_flag ::= constant_expression

stamptime_condition ::= mintypmax_expression

start_edge_offset ::= mintypmax_expression

threshold ::= constant_expression

timing_check_limit ::= expression

A.7.5.3 System timing check event definitions

timing_check_event ::=
    [timing_check_event_control] specify_terminal_descriptor [**&&&** timing_check_condition]

controlled_timing_check_event ::=
    timing_check_event_control specify_terminal_descriptor [**&&&** timing_check_condition]

timing_check_event_control ::=
    posedge
  | **negedge**
  | edge_control_specifier

specify_terminal_descriptor ::=
    specify_input_terminal_descriptor
  | specify_output_terminal_descriptor

edge_control_specifier ::= **edge** **[** edge_descriptor { **,** edge_descriptor } **]**

edge_descriptor[2] ::=
    01
  | **10**
  | z_or_x zero_or_one
  | zero_or_one z_or_x

zero_or_one ::= **0** | **1**

z_or_x ::= **x** | **X** | **z** | **Z**

timing_check_condition ::=
    scalar_timing_check_condition
  | **(** scalar_timing_check_condition **)**

scalar_timing_check_condition ::=
    expression
  | **~** expression
  | expression **==** scalar_constant

| expression **===** scalar_constant
| expression **!=** scalar_constant
| expression **!==** scalar_constant

scalar_constant ::=
1'b0 | **1'b1** | **1'B0** | **1'B1** | **'b0** | **'b1** | **'B0** | **'B1** | **1** | **0**

A.8 Expressions

A.8.1 Concatenations

analog_concatenation ::= **{** analog_expression { **,** analog_expression } **}**

analog_multiple_concatenation ::= **{** constant_expression analog_concatenation **}**

analog_filter_function_arg ::=
parameter_identifier
| parameter_identifier **[** msb_constant_expression **:** lsb_constant_expression **]**
| constant_optional_arrayinit

concatenation ::= **{** expression { **,** expression } **}**

constant_concatenation ::= **{** constant_expression { **,** constant_expression } **}**

constant_multiple_concatenation ::= **{** constant_expression constant_concatenation **}**

module_path_concatenation ::= **{** module_path_expression { **,** module_path_expression } **}**

module_path_multiple_concatenation ::= **{** constant_expression module_path_concatenation **}**

multiple_concatenation ::= **{** constant_expression concatenation **}**

constant_arrayinit ::= **'{** constant_expression { **,** constant_expression } **}**

constant_optional_arrayinit ::= **'{** [constant_expression] { **,** [constant_expression] } **}**

A.8.2 Function calls

analog_function_call ::=
analog_function_identifier { attribute_instance } **(** analog_expression { **,** analog_expression } **)**

analog_system_function_call ::=
analog_system_function_identifier [**(** [analog_expression] { **,** [analog_expression] } **)**]

analog_built_in_function_call ::=
analog_built_in_function_name **(** analog_expression [**,** analog_expression] **)**

analog_built_in_function_name ::=
ln | **log** | **exp** | **sqrt** | **min** | **max** | **abs** | **pow** | **floor** | **ceil**
| **sin** | **cos** | **tan** | **asin** | **acos** | **atan** | **atan2**
| **hypot** | **sinh** | **cosh** | **tanh** | **asinh** | **acosh** | **atanh**

analysis_function_call ::= **analysis (** **"** analysis_identifier **"** { **,** **"** analysis_identifier **"** } **)**

analog_filter_function_call ::=
ddt (analog_expression [**,** abstol_expression] **)**
| **ddx (** analog_expression **,** branch_probe_function_call **)**
| **idt (** analog_expression [**,** analog_expression [**,** analog_expression [**,** abstol_expression]]] **)**
| **idtmod (** analog_expression [**,** analog_expression [**,** analog_expression [**,** analog_expression [**,** abstol_expression]]]] **)**
| **absdelay (** analog_expression **,** analog_expression [**,** constant_expression] **)**
| **transition (** analog_expression [**,** analog_expression [**,** analog_expression [**,** analog_expression [**,** constant_expression]]]] **)**

```
    | slew ( analog_expression [ , analog_expression [ , analog_expression ] ] )
    | last_crossing ( analog_expression [ , analog_expression ] )
    | limexp ( analog_expression )
    | laplace_filter_name ( analog_expression , [ analog_filter_function_arg ] ,
        [ analog_filter_function_arg ] [ , constant_expression ] )
    | zi_filter_name ( analog_expression , [ analog_filter_function_arg ] ,
        [ analog_filter_function_arg ] , constant_expression
        [ , analog_expression [ , constant_expression ] ] )
analog_small_signal_function_call ::=
      ac_stim ( [ " analysis_identifier " [ , analog_expression [ , analog_expression ] ] ] )
    | white_noise ( analog_expression [ , string ] )
    | flicker_noise ( analog_expression , analog_expression [ , string ] )
    | noise_table ( noise_table_input_arg [ , string ] )
noise_table_input_arg ::=
      parameter_identifier
    | parameter_identifier [ msb_constant_expression : lsb_constant_expression ]
    | string
    | constant_arrayinit
laplace_filter_name ::= laplace_zd | laplace_zp | laplace_np | laplace_nd
zi_filter_name ::= zi_zp | zi_zd | zi_np | zi_nd
branch_probe_function_call ::=
      nature_attribute_identifier ( branch_reference )
    | nature_attribute_identifier ( analog_net_reference [ , analog_net_reference ] )
port_probe_function_call ::= nature_attribute_identifier ( < analog_port_reference > )
constant_analog_function_call ::=
      analog_function_identifier { attribute_instance } ( constant_expression { , constant_expression } )
constant_function_call ::= function_identifier { attribute_instance }
      ( constant_expression { , constant_expression } )
constant_system_function_call ::= system_function_identifier
      ( constant_expression { , constant_expression } )
constant_analog_built_in_function_call ::=
      analog_built_in_function_name ( constant_expression [ , constant_expression ] )
function_call ::= hierarchical_function_identifier{ attribute_instance }
      ( expression { , expression } )
system_function_call ::= system_function_identifier
      [ ( expression { , expression } ) ]
```

A.8.3 Expressions

```
abstol_expression ::=
      constant_expression
    | nature_identifier
analog_conditional_expression ::=
      analog_expression ? { attribute_instance } analog_expression : analog_expression
analog_range_expression ::=
      analog_expression
    | msb_constant_expression : lsb_constant_expression
analog_expression_or_null ::= [ analog_expression ]
analog_expression ::=
```

analog_primary
| unary_operator { attribute_instance } analog_primary
| analog_expression binary_operator { attribute_instance } analog_expression
| analog_conditional_expression
| string

base_expression ::= expression

conditional_expression ::= expression1 **?** { attribute_instance } expression2 **:** expression3

constant_base_expression ::= constant_expression

constant_expression_or_null ::= [constant_expression]

constant_expression ::=
constant_primary
| unary_operator { attribute_instance } constant_primary
| constant_expression binary_operator { attribute_instance } constant_expression
| constant_expression **?** { attribute_instance } constant_expression **:** constant_expression

analysis_or_constant_expression ::=
constant_primary
| analysis_function_call
| unary_operator { attribute_instance } analysis_or_constant_primary
| analysis_or_constant_expression binary_operator { attribute_instance }
analysis_constant_expression
| analysis_or_constant_expression **?** { attribute_instance } analysis_or_constant_expression **:**
analysis_or_constant_expression

constant_mintypmax_expression ::=
constant_expression
| constant_expression **:** constant_expression **:** constant_expression

constant_range_expression ::=
constant_expression
| msb_constant_expression **:** lsb_constant_expression
| constant_base_expression **+:** width_constant_expression
| constant_base_expression **-:** width_constant_expression

dimension_constant_expression ::= constant_expression

expression ::=
primary
| unary_operator { attribute_instance } primary
| expression binary_operator { attribute_instance } expression
| conditional_expression

expression1 ::= expression

expression2 ::= expression

expression3 ::= expression

indirect_expression ::=
branch_probe_function_call
| port_probe_function_call
| **ddt** **(** branch_probe_function_call [**,** abstol_expression] **)**
| **ddt** **(** port_probe_function_call [**,** abstol_expression] **)**
| **idt** **(** branch_probe_function_call [**,** analog_expression
[**,** analog_expression [**,** abstol_expression]]] **)**
| **idt** **(** port_probe_function_call [**,** analog_expression [**,** analog_expression
[**,** abstol_expression]]] **)**
| **idtmod** **(** branch_probe_function_call [**,** analog_expression [**,** analog_expression
[**,** analog_expression [**,** abstol_expression]]]] **)**

```
    | idtmod ( port_probe_function_call [ , analog_expression [ , analog_expression
        [ , analog_expression [ , abstol_expression ] ] ] ] )
lsb_constant_expression ::= constant_expression
mintypmax_expression ::=
        expression
    | expression : expression : expression
module_path_conditional_expression ::= module_path_expression ? { attribute_instance }
        module_path_expression : module_path_expression
module_path_expression ::=
        module_path_primary
    | unary_module_path_operator { attribute_instance } module_path_primary
    | module_path_expression binary_module_path_operator { attribute_instance }
        module_path_expression
    | module_path_conditional_expression
module_path_mintypmax_expression ::=
        module_path_expression
    | module_path_expression : module_path_expression : module_path_expression
msb_constant_expression ::= constant_expression
nature_attribute_expression ::=
        constant_expression
    | nature_identifier
    | nature_access_identifier
range_expression ::=
        expression
    | msb_constant_expression : lsb_constant_expression
    | base_expression +: width_constant_expression
    | base_expression -:  width_constant_expression
width_constant_expression ::= constant_expression
```

A.8.4 Primaries

```
analog_primary ::=
        number
    | analog_concatenation
    | analog_multiple_concatenation
    | variable_reference
    | net_reference
    | genvar_identifier
    | parameter_reference
    | nature_attribute_reference
    | branch_probe_function_call
    | port_probe_function_call
    | analog_function_call
    | analog_system_function_call
    | analog_built_in_function_call
    | analog_filter_function_call
    | analog_small_signal_function_call
    | analysis_function_call
    | ( analog_expression )
constant_primary ::=
        number
```

| parameter_identifier [[constant_range_expression]]
| specparam_identifier [[constant_range_expression]]
| constant_concatenation
| constant_multiple_concatenation
| constant_function_call
| constant_system_function_call
| constant_analog_built_in_function_call
| (constant_mintypmax_expression)
| string
| system_parameter_identifier
| nature_attribute_reference
| constant_analog_function_call
| constant_let_expression

module_path_primary ::=
number
| identifier
| module_path_concatenation
| module_path_multiple_concatenation
| function_call
| system_function_call
| (module_path_mintypmax_expression)

primary ::=
number
| hierarchical_identifier [{ [expression] } [range_expression]]
| concatenation
| multiple_concatenation
| function_call
| system_function_call
| (mintypmax_expression)
| string
| branch_probe_function_call
| port_probe_function_call
| nature_attribute_reference
| analog_function_call
| analog_built_in_function_call
| sequence_method_call
| let_expression

constant_let_expression ::= let_expression[1]

A.8.5 Expression left-side values

analog_variable_lvalue ::=
variable_identifier
| variable_identifier [analog_expression] { [analog_expression] }

branch_lvalue ::= branch_probe_function_call

net_lvalue ::=
hierarchical_net_identifier [{ [constant_expression] } [constant_range_expression]]
| { net_lvalue { , net_lvalue } }

variable_lvalue ::=

[1]In a constant_let_expression all arguments shall be constant_expressions and its right hand side shall be a constant_expression itself provided that its formal arguments are treated as constant_primary there.

hierarchical_variable_identifier [{ **[** expression **]** } **[** range_expression **]**]
| **{** variable_lvalue { **,** variable_lvalue } **}**

A.8.6 Operators

unary_operator ::=
+ | **-** | **!** | **~** | **&** | **~&** | **|** | **~|** | **^** | **~^** | **^~**

binary_operator ::=
+ | **-** | ***** | **/** | **%** | **==** | **!=** | **===** | **!==** | **&&** | **||** | ******
| **<** | **<=** | **>** | **>=** | **&** | **|** | **^** | **^~** | **~^** | **>>** | **<<** | **>>>** | **<<<**

unary_module_path_operator ::=
! | **~** | **&** | **~&** | **|** | **~|** | **^** | **~^** | **^~**

binary_module_path_operator ::=
== | **!=** | **&&** | **||** | **&** | **|** | **^** | **^~** | **~^**

A.8.7 Numbers

number ::=
decimal_number
| octal_number
| binary_number
| hex_number
| real_number

real_number[2] ::=
unsigned_number **.** unsigned_number
| unsigned_number [**.** unsigned_number] exp [sign] unsigned_number
| unsigned_number [**.** unsigned_number] scale_factor

exp ::= **e** | **E**

scale_factor ::= **T** | **G** | **M** | **K** | **k** | **m** | **u** | **n** | **p** | **f** | **a**

decimal_number ::=
unsigned_number
| [size] decimal_base unsigned_number
| [size] decimal_base x_digit { _ }
| [size] decimal_base z_digit { _ }

binary_number ::= [size] binary_base binary_value

octal_number ::= [size] octal_base octal_value

hex_number ::= [size] hex_base hex_value

sign ::= **+** | **-**

size ::= non_zero_unsigned_number

non_zero_unsigned_number[2] ::= non_zero_decimal_digit { _ | decimal_digit}

unsigned_number[2] ::= decimal_digit { _ | decimal_digit }

binary_value[2] ::= binary_digit { _ | binary_digit }

octal_value[2] ::= octal_digit { _ | octal_digit }

hex_value[2] ::= hex_digit { _ | hex_digit }

decimal_base[2] ::= **'**[**s**|**S**]**d** | **'**[**s**|**S**]**D**

binary_base[2] ::= **'**[**s**|**S**]**b** | **'**[**s**|**S**]**B**

octal_base[2] ::= '[s|S]o | '[s|S]O

hex_base[2] ::= '[s|S]h | '[s|S]H

non_zero_decimal_digit ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

decimal_digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

binary_digit ::= x_digit | z_digit | 0 | 1

octal_digit ::= x_digit | z_digit | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

hex_digit ::=
x_digit | z_digit | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
| a | b | c | d | e | f | A | B | C | D | E | F

x_digit ::= x | X

z_digit ::= z | Z | ?

A.8.8 Strings

string ::= " { Any_ASCII_Characters } "

A.8.9 Analog references

nature_attribute_reference ::= net_identifier . potential_or_flow . nature_attribute_identifier

analog_port_reference ::=
port_identifier
| port_identifier [constant_expression]

analog_net_reference ::=
port_identifier
| port_identifier [constant_expression]
| net_identifier
| net_identifier [constant_expression]

branch_reference ::=
hierarchical_branch_identifier
| hierarchical_branch_identifier [constant_expression]

parameter_reference ::=
parameter_identifier
| parameter_identifier [analog_expression]

variable_reference ::=
variable_identifier
| variable_identifier [analog_expression] { [analog_expression] }
| real_identifier
| real_identifier [analog_expression] { [analog_expression] }

net_reference ::=
hierarchical_net_identifier
| hierarchical_net_identifier [analog_range_expression]
| hierarchical_port_identifier
| hierarchical_port_identifier [analog_range_expression]

A.9 General

A.9.1 Attributes

attribute_instance ::= (* attr_spec { , attr_spec } *)
attr_spec ::= attr_name [= constant_expression]
attr_name ::= identifier

A.9.2 Comments

comment ::=
one_line_comment
| block_comment
one_line_comment ::= // comment_text \n
block_comment ::= /* comment_text */
comment_text ::= { Any_ASCII_character }

A.9.3 Identifiers

analog_block_identifier ::= block_identifier
analog_function_identifier ::= identifier
analog_system_task_identifier ::= system_task_identifier
analog_system_function_identifier ::= system_function_identifier
analysis_identifier ::= identifier
block_identifier ::= identifier
branch_identifier ::= identifier
cell_identifier ::= identifier
checker_identifier ::= identifier
clocking_identifier ::= identifier
config_identifier ::= identifier
connectmodule_identifier ::= module_identifier
connectrules_identifier ::= identifier
discipline_identifier ::= identifier
escaped_identifier ::= \ {Any_ASCII_character_except_white_space} white_space
event_identifier ::= identifier
function_identifier ::= identifier
gate_instance_identifier ::= identifier
generate_block_identifier ::= identifier
genvar_identifier ::= identifier
hierarchical_block_identifier ::= hierarchical_identifier
hierarchical_branch_identifier ::= hierarchical_identifier
hierarchical_event_identifier ::= hierarchical_identifier
hierarchical_function_identifier ::= hierarchical_identifier
hierarchical_identifier ::= { identifier [[constant_expression]] . } identifier
hierarchical_net_identifier ::= hierarchical_identifier
hierarchical_parameter_identifier ::= hierarchical_identifier
hierarchical_port_identifier ::= hierarchical_identifier
hierarchical_property_identifier ::= hierarchical_identifier

hierarchical_sequence_identifier :: hierarchical_identifier
hierarchical_task_identifier ::= hierarchical_identifier
hierarchical_variable_identifier ::= hierarchical_identifier
identifier ::=
simple_identifier
| escaped_identifier
inout_port_identifier ::= identifier
input_port_identifier ::= identifier
instance_identifier ::= identifier
library_identifier ::= identifier
module_identifier ::= identifier
module_instance_identifier ::= identifier
module_or_paramset_identifier ::=
module_identifier
| paramset_identifier
module_parameter_identifier ::= identifier
nature_identifier ::= identifier
nature_access_identifier ::= identifier
nature_attribute_identifier ::= **abstol** | **access** | **ddt_nature** | **idt_nature** | **units** | identifier
net_identifier ::= identifier
output_port_identifier ::= identifier
parameter_identifier ::= identifier
paramset_identifier ::= identifier
port_identifier ::= identifier
property_identifier ::= identifier
real_identifier ::= identifier
signal_identifier ::= identifier
simple_identifier[3] ::= [**a-zA-Z_**] { [**a-zA-Z0-9_$**] }
specparam_identifier ::= identifier
system_function_identifier[4] ::= **$** [**a-zA-Z0-9_$**] { [**a-zA-Z0-9_$**] }
system_parameter_identifier ::= **$** [**a-zA-Z0-9_$**] { [**a-zA-Z0-9_$**] }
system_task_identifier[4] ::= **$** [**a-zA-Z0-9_$**] { [**a-zA-Z0-9_$**] }
task_identifier ::= identifier
terminal_identifier ::= identifier
text_macro_identifier ::=
identifier
| **__VAMS_ENABLE__**
| **__VAMS_COMPACT_MODELING__**
topmodule_identifier ::= identifier
udp_identifier ::= identifier
udp_instance_identifier ::= identifier
variable_identifier ::= identifier

A.9.4 White space

white_space ::= space | tab | newline | eof[5]

A.10 Details

1) Function statements are limited by the rules of 4.7.1.
2) Embedded spaces are illegal.
3) A simple_identifier shall start with an alpha or underscore (_) character, shall have at least one character, and shall not have any spaces.
4) The $ character in a system_function_identifier or system_task_identifier shall not be followed by white_space. A system_function_identifier or system_task_identifier shall not be escaped.
5) End of file.