Annex A
(normative)
Formal syntax

The formal syntax of SystemVerilog is described using Backus-Naur Form (BNF). The syntax of SystemVerilog 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 conventions used are as follows:

- lKeywords and punctuation are in **bold-red** text.
- lSyntactic categories are named in nonbold text.
- lA vertical bar (|) separates alternatives.
- lSquare brackets ([ ]) enclose optional items.
- lBraces ({ }) enclose items that can be repeated zero or more times.

The full syntax and semantics of SystemVerilog are not described solely using BNF. The normative text description contained within the clauses and annexes of this standard provide additional details on the syntax and semantics described in this BNF.

A *qualified term* in the syntax is a term such as *array_identifier* for which the "array" portion represents some semantic intent and the "identifier" term indicates that the qualified term reduces to the "identifier" term in the syntax. The syntax does not completely define the semantics of such qualified terms; for example while an identifier which would qualify semantically as an array_identifier is created by a declaration, such declaration forms are not explicitly described using *array_identifier* in the syntax.

SystemVerilog-AMS merging notes:

- AMS syntax from the Verilog-AMS LRM version 2.3.1 will be merged into the SystemVerilog syntax.
- <text> is AMS syntax inserted into the existing SystemVerilog syntax.
- <text> is existing SystemVerilog syntax that is not present in the AMS syntax (version 2.3.1).
- <Text> is inserted comments, questions, and notes relating to the merger of AMS syntax into the SystemVerilog syntax.
- <Text> is inserted comments, changes added based on the AMS syntax merger review.

Editing syntax guidelines:

- multiple line syntax is created suing a newline break
- Keyword uses "Courier New" font of size 9, bold and red
- Other syntax uses "Time New Roman" font of size 10, black.

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 }] ;

include_statement ::= include file_path_spec ;

A.1.2 SystemVerilog source text

source_text ::= [timeunits_declaration] { description }

description ::=
module_declaration
| udp_declaration
| interface_declaration
| program_declaration
| package_declaration
| { attribute_instance } package_item
| { attribute_instance } bind_directive
| config_declaration
| paramset_declaration
| nature_declaration
| discipline_declaration
| connectrules_declaration

module_nonansi_header ::=
{ attribute_instance } module_keyword [lifetime] module_identifier
{ package_import_declaration } [parameter_port_list] list_of_ports ;

module_ansi_header ::=
{ attribute_instance } module_keyword [lifetime] module_identifier
{ package_import_declaration }[1] [parameter_port_list] [list_of_port_declarations] ;

module_declaration ::=
module_nonansi_header [timeunits_declaration] { module_item }
endmodule [: module_identifier]
| module_ansi_header [timeunits_declaration] { non_port_module_item }
endmodule [: module_identifier]
| { attribute_instance } module_keyword [lifetime] module_identifier (.*) ;
[timeunits_declaration] { module_item } endmodule [: module_identifier]
| extern module_nonansi_header
| extern module_ansi_header

REVIEW: Highlighted connectmodule keyword

module_keyword ::= module | macromodule | connectmodule

interface_declaration ::=
interface_nonansi_header [timeunits_declaration] { interface_item }
endinterface [: interface_identifier]
| interface_ansi_header [timeunits_declaration] { non_port_interface_item }

endinterface [**:** interface_identifier]

| { attribute_instance } **interface** interface_identifier **(.*) ;**
[timeunits_declaration] { interface_item }
endinterface [**:** interface_identifier]

| **extern** interface_nonansi_header

| **extern** interface_ansi_header

interface_nonansi_header ::=
{ attribute_instance } **interface** [lifetime] interface_identifier
{ package_import_declaration } [parameter_port_list] list_of_ports **;**

interface_ansi_header ::=
{attribute_instance } **interface** [lifetime] interface_identifier
{ package_import_declaration }[1] [parameter_port_list] [list_of_port_declarations] **;**

REVIEW: Add nature and discipline into packages, however packages do not contain global scope declarations.

program_declaration ::=
program_nonansi_header [timeunits_declaration] { program_item }
endprogram [**:** program_identifier]

| program_ansi_header [timeunits_declaration] { non_port_program_item }
endprogram [**:** program_identifier]

| { attribute_instance } **program** program_identifier **(.*) ;**
[timeunits_declaration] { program_item }
endprogram [**:** program_identifier]

| **extern** program_nonansi_header

| **extern** program_ansi_header

program_nonansi_header ::=
{ attribute_instance } **program** [lifetime] program_identifier
{ package_import_declaration } [parameter_port_list] list_of_ports **;**

program_ansi_header ::=
{attribute_instance } **program** [lifetime] program_identifier
{ package_import_declaration }[1] [parameter_port_list] [list_of_port_declarations] **;**

checker_declaration ::=
checker checker_identifier [**(** [checker_port_list] **)**] **;**
{ checker_or_generate_item }
endchecker [**:** checker_identifier]

class_declaration ::=
[**virtual**] **class** [lifetime] class_identifier [parameter_port_list]
[**extends** class_type [**(** list_of_arguments **)**]] **;**
{ class_item }
endclass [**:** class_identifier]

package_declaration ::=
{ attribute_instance } **package** [lifetime] package_identifier **;**
[timeunits_declaration] { { attribute_instance } package_item }
endpackage [**:** package_identifier]

timeunits_declaration ::=
timeunit time_literal [**/** time_literal] **;**

| **timeprecision** time_literal **;**
| **timeunit** time_literal **;** **timeprecision** time_literal **;**
| **timeprecision** time_literal **;** **timeunit** time_literal **;**

A.1.3 Module parameters and ports

parameter_port_list ::=
**(** list_of_param_assignments { **,** parameter_port_declaration } **)**
| **#** **(** parameter_port_declaration { **,** parameter_port_declaration } **)**
| **#(** **)**

parameter_port_declaration ::=
parameter_declaration
| local_parameter_declaration
| data_type list_of_param_assignments
| **type** list_of_type_assignments

list_of_ports ::= **(** port { **,** port } **)**

list_of_port_declarations[2] ::=
([{ attribute_instance} ansi_port_declaration { **,** { attribute_instance} ansi_port_declaration }] **)**

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

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

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

port_reference ::=
port_identifier constant_select

port_direction ::= **input** | **output** | **inout** | **ref**

net_port_header ::= [port_direction] net_port_type

variable_port_header ::= [port_direction] variable_port_type

interface_port_header ::=
interface_identifier [**.** modport_identifier]
| **interface** [**.** modport_identifier]

ansi_port_declaration ::=
[net_port_header | interface_port_header] port_identifier { unpacked_dimension }
[**=** constant_expression]
| [variable_port_header] port_identifier { variable_dimension } [**=** constant_expression]
| [port_direction] **.** port_identifier **(** [expression] **)**

A.1.4 Module items

elaboration_system_task ::=
    $fatal [**(** finish_number [**,** list_of_arguments] **)**] **;**
  | **$error** [**(** [list_of_arguments] **)**] **;**
  | **$warning** [**(** [list_of_arguments] **)**] **;**
  | **$info** [**(** [list_of_arguments] **)**] **;**

finish_number ::= **0** | **1** | **2**

NOTE: Separated analog and analog initial constructs to be consistent with other syntax constructs.

module_common_item ::=
    module_or_generate_item_declaration
  | interface_instantiation
  | program_instantiation
  | assertion_item
  | bind_directive
  | continuous_assign
  | net_alias
  | initial_construct
  | final_construct
  | always_construct
  | loop_generate_construct
  | conditional_generate_construct
  | elaboration_system_task
  | { attribute_instance } analog_construct
  | { attribute_instance } analog_initial_construct

module_item ::=
    port_declaration **;**
  | non_port_module_item

REVIEW: Moved branch_declaration from module_or_generate_item_declaration to module_or_generate_item

module_or_generate_item ::=
    { attribute_instance } parameter_override
  | { attribute_instance } gate_instantiation
  | { attribute_instance } udp_instantiation
  | { attribute_instance } module_instantiation
  | { attribute_instance } module_common_item
  | branch_declaration

module_or_generate_item_declaration ::=
    package_or_generate_item_declaration
  | genvar_declaration
  | clocking_declaration
  | **default clocking** clocking_identifier **;**
  | **default disable iff** expression_or_dist **;**
  | analog_function_declaration

non_port_module_item ::=

generate_region
| module_or_generate_item
| specify_block
| { attribute_instance } specparam_declaration
| program_declaration
| module_declaration
| interface_declaration
| timeunits_declaration[3]
| aliasparam_declaration

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

bind_directive[4] ::=
bind bind_target_scope [**:** bind_target_instance_list] bind_instantiation **;**
| **bind** bind_target_instance bind_instantiation **;**

bind_target_scope ::=
module_identifier
| interface_identifier

bind_target_instance ::=
hierarchical_identifier constant_bit_select

bind_target_instance_list ::=
bind_target_instance { **,** bind_target_instance }

bind_instantiation ::=
program_instantiation
| module_instantiation
| interface_instantiation
| checker_instantiation

A.1.5 Configuration source text

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

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**]
| **use** named_parameter_assignment { , named_parameter_assignment } [**: config**]
| **use** [library_identifier .] cell_identifier named_parameter_assignment
{ , named_parameter_assignment } [**: config**]

A.1.6 Interface items

interface_or_generate_item ::=
{ attribute_instance } module_common_item
| { attribute_instance } modport_declaration
| { attribute_instance } extern_tf_declaration

extern_tf_declaration ::=
extern method_prototype **;**
| **extern forkjoin** task_prototype **;**

REVIEW: Investigate supporting AMS syntax within interface_item syntax

interface_item ::=
port_declaration **;**
| non_port_interface_item

REVIEW: add aliasparam into interface declarations.

non_port_interface_item ::=
generate_region
| interface_or_generate_item
| program_declaration
| interface_declaration
| timeunits_declaration[3]
| aliasparam_declaration

A.1.7 Program items

program_item ::=
port_declaration **;**
| non_port_program_item

REVIEW: add aliasparam into program declarations.

non_port_program_item ::=
{ attribute_instance } continuous_assign
| { attribute_instance } module_or_generate_item_declaration
| { attribute_instance } initial_construct
| { attribute_instance } final_construct
| { attribute_instance } concurrent_assertion_item
| { attribute_instance } timeunits_declaration[3]

| program_generate_item
| aliasparam_declaration

program_generate_item[5] ::=
loop_generate_construct
| conditional_generate_construct
| generate_region
| elaboration_system_task

A.1.8 Checker items

checker_port_list ::=
checker_port_item {, checker_port_item}

checker_port_item ::=
{ attribute_instance } property_formal_type port_identifier {variable_dimension}
[= property_actual_arg]

checker_or_generate_item ::=
checker_or_generate_item_declaration
| initial_construct
| checker_always_construct
| final_construct
| assertion_item
| checker_generate_item

checker_or_generate_item_declaration ::=
[**rand**] data_declaration
| function_declaration
| assertion_item_declaration
| covergroup_declaration
| overload_declaration
| genvar_declaration
| clocking_declaration
| **default clocking** clocking_identifier **;**
| **default disable iff** expression_or_dist **;**
| **;**

checker_generate_item[6] ::=
loop_generate_construct
| conditional_generate_construct
| generate_region
| elaboration_system_task

checker_always_construct ::= **always** statement

A.1.9 Class items

class_item ::=
{ attribute_instance } class_property

| { attribute_instance } class_method
| { attribute_instance } class_constraint
| { attribute_instance } class_declaration
| { attribute_instance } covergroup_declaration
| local_parameter_declaration ;
| parameter_declaration[7] ;
| ;

class_property ::=
{ property_qualifier } data_declaration
| **const** { class_item_qualifier } data_type const_identifier [**=** constant_expression] ;

class_method ::=
{ method_qualifier } task_declaration
| { method_qualifier } function_declaration
| **extern** { method_qualifier } method_prototype ;
| { method_qualifier } class_constructor_declaration
| **extern** { method_qualifier } class_constructor_prototype

class_constructor_prototype ::=
function new ([tf_port_list] **)** ;

class_constraint ::=
constraint_prototype
| constraint_declaration

class_item_qualifier[8] ::=
static
| **protected**
| **local**

property_qualifier[8] ::=
random_qualifier
| class_item_qualifier

random_qualifier[8] ::=
rand
| **randc**

method_qualifier[8] ::=
[**pure**] **virtual**
| class_item_qualifier

method_prototype ::=
task_prototype
| function_prototype

class_constructor_declaration ::=
function [class_scope] **new** [**(** [tf_port_list] **)**] ;
{ block_item_declaration }
[**super . new** [**(** list_of_arguments **)**] ;]
{ function_statement_or_null }
endfunction [**: new**]

A.1.10 Constraints

constraint_declaration ::= [**static**] **constraint** constraint_identifier constraint_block

constraint_block ::= **{** { constraint_block_item } **}**

constraint_block_item ::=
    solve solve_before_list **before** solve_before_list **;**
  | constraint_expression

solve_before_list ::= solve_before_primary { **,** solve_before_primary }

solve_before_primary ::= [implicit_class_handle **.** | class_scope] hierarchical_identifier select

constraint_expression ::=
    expression_or_dist **;**
  | expression **->** constraint_set
  | **if (** expression **)** constraint_set [**else** constraint_set]
  | **foreach (** ps_or_hierarchical_array_identifier **[** loop_variables **])** constraint_set

constraint_set ::=
    constraint_expression
  | **{** { constraint_expression } **}**

dist_list ::= dist_item { **,** dist_item }

dist_item ::= value_range [dist_weight]

dist_weight ::=
    := expression
  | **:/** expression

constraint_prototype ::= [constraint_prototype_qualifier] [**static**] **constraint** constraint_identifier **;**

constraint_prototype_qualifier ::= **extern** | **pure**

extern_constraint_declaration ::=
    [**static**] **constraint** class_scope constraint_identifier constraint_block

identifier_list ::= identifier { **,** identifier }

A.1.11 Package items

package_item ::=
    package_or_generate_item_declaration
  | anonymous_program
  | package_export_declaration
  | timeunits_declaration[3]

REVIEW: add analog_function_declaration into package_or_generate_item_declaration

package_or_generate_item_declaration ::=
    net_declaration
  | data_declaration
  | task_declaration
  | function_declaration
  | checker_declaration
  | dpi_import_export

| extern_constraint_declaration
| class_declaration
| class_constructor_declaration
| local_parameter_declaration ;
| parameter_declaration ;
| covergroup_declaration
| overload_declaration
| assertion_item_declaration
| analog_function_declaration
| ;

anonymous_program ::= **program** ; { anonymous_program_item } **endprogram**

anonymous_program_item ::=
task_declaration
| function_declaration
| class_declaration
| covergroup_declaration
| class_constructor_declaration
| ;

A.1.12 Nature declaration

REVIEW: Make optional semicolon required in order to be consistent with P1800

REVIEW: Investigate adding natures into packages

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.1.13 Discipline declaration

REVIEW: Make optional semicolon required in order to be consistent with P1800

REVIEW: Investigate adding disciplines into packages

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.1.14 Connectrules declaration

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

connectrules_item ::=
        connect_insertion
    | connect_resolution

QUESTION: The expressions within the parameter_value_assignment has been expanded to include new systemVerilog constructs. Do we need to restrict what types of assignments are used here?

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

NOTE: Reordered *connect_resolution* syntax item to avoid inline | expression and improve readability.

connect_resolution ::= **connect** discipline_identifier { **,** discipline_identifier } **resolveto** resolved_discipline **;**

resolved_discipline ::= discipline_identifier | **exclude**

A.1.15 Paramset declaration

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

NOTE: The *integer_declaration* and *real_declaration* syntax item are not defined in SystemVerilog syntax, instead use *data_declaration* syntax item to access integer and real declarations. Need to identify what additional data declarations are to be supoorted and not supported in *paramset_item_declaration* syntax item.

paramset_item_declaration ::=
        { attribute_instance } parameter_declaration **;**
    | { attribute_instance } local_parameter_declaration **;**
    | aliasparam_declaration
    | { attribute_instance } data_declaration

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

QUESTION: Does the paramset support the new SystemVerilog binary operators?

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 data_type_or_implicit list_of_param_assignments
  | **localparam type** list_of_type_assignments

parameter_declaration ::=
    parameter data_type_or_implicit list_of_param_assignments
  | **parameter type** list_of_type_assignments

specparam_declaration ::=
    specparam [packed_dimension] list_of_specparam_assignments ;

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

A.2.1.2 Port declarations

QUESTION: Move the optional *discipline_identifier* and *wreal* keyword into the *net_port_type* SystemVerilog syntax item?

QUESTION: Syntactically real, integer and other variable types can be passed in and out of the module declaration through ports, how should the wreal be integrated into the existing the SystemVerilog syntax?

inout_declaration ::=
    inout [discipline_identifier] [net_port_type | **wreal**] list_of_port_identifiers

input_declaration ::=
    input [discipline_identifier] [net_port_type | **wreal**] list_of_port_identifiers
  | **input** variable_port_type list_of_variable_identifiers

output_declaration ::=
    output [discipline_identifier] [net_port_type | **wreal**] list_of_port_identifiers
  | **output** variable_port_type list_of_variable_port_identifiers

interface_port_declaration ::=
    interface_identifier list_of_interface_identifiers
  | interface_identifier . modport_identifier list_of_interface_identifiers

ref_declaration ::= **ref** variable_port_type list_of_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]

data_declaration[9] ::=
[const] [var] [lifetime] data_type_or_implicit list_of_variable_decl_assignments ;
| type_declaration
| package_import_declaration[10]
| virtual_interface_declaration

package_import_declaration ::=
import package_import_item { , package_import_item } ;

package_import_item ::=
package_identifier :: identifier
| package_identifier :: *

package_export_declaration ::=
export *::* ;
| export package_import_item { , package_import_item } ;

genvar_declaration ::= genvar list_of_genvar_identifiers ;

NOTE: Replaced *range* syntax item with *packed_dimension* syntax to be consistent with other SystemVerilog net type declarations.

QUESTION: Can existing systemVerilog real net type be used instead of the AMS wreal construct?

QUESTION: Can wreal and ground included as part of the net_type SystemVerilog syntax item?

QUESTION: The *reg_declaration* syntax item is not defined in SystemVerilog syntax. By adding the optional *discipline_identifier* into the *net_declaration* SystemVerilog syntax item, does the a reg keyword declaration use the optional *discipline_identifier* syntax?

net_declaration[11] ::=
net_type [discipline_identifier] [drive_strength | charge_strength] [vectored | scalared]
data_type_or_implicit [delay3] list_of_net_decl_assignments ;
| discipline_identifier [packed_dimension] list_of_net_identifiers ;
| discipline_identifier [packed_dimension] list_of_net_decl_assignments ;
| wreal [discipline_identifier] [packed_dimension] list_of_net_identifiers ;
| wreal [discipline_identifier] [packed_dimension] list_of_net_decl_assignments ;
| ground [discipline_identifier] [packed_dimension] list_of_net_identifiers ;

type_declaration ::=
typedef data_type type_identifier { variable_dimension } ;
| typedef interface_instance_identifier constant_bit_select . type_identifier type_identifier ;
| typedef [enum | struct | union | class] type_identifier ;

lifetime ::= static | automatic

A.2.2 Declaration data types

A.2.2.1 Net and variable types

casting_type ::= simple_type | constant_primary | signing | string | const

data_type ::=

integer_vector_type [signing] { packed_dimension }
| integer_atom_type [signing]
| non_integer_type
| struct_union [**packed** [signing]] **{** struct_union_member { struct_union_member } **}**
{ packed_dimension }[12]
| **enum** [enum_base_type] **{** enum_name_declaration { **,** enum_name_declaration } **}**
{ packed_dimension }
| **string**
| **chandle**
| **virtual** [**interface**] interface_identifier
| [class_scope | package_scope] type_identifier { packed_dimension }
| class_type
| **event**
| ps_covergroup_identifier
| type_reference[13]

data_type_or_implicit ::=
data_type
| implicit_data_type

implicit_data_type ::= [signing] { packed_dimension }

enum_base_type ::=
integer_atom_type [signing]
| integer_vector_type [signing] [packed_dimension]
| type_identifier [packed_dimension][14]

enum_name_declaration ::=
enum_identifier [**[** integral_number [**:** integral_number] **]**] [**=** constant_expression]

class_scope ::= class_type **::**

class_type ::=
ps_class_identifier [parameter_value_assignment]
{ **::** class_identifier [parameter_value_assignment] }

integer_type ::= integer_vector_type | integer_atom_type

integer_atom_type ::= **byte** | **shortint** | **int** | **longint** | **integer** | **time**

integer_vector_type ::= **bit** | **logic** | **reg**

non_integer_type ::= **shortreal** | **real** | **realtime**

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

net_port_type[15] ::=
[net_type] data_type_or_implicit

variable_port_type ::= var_data_type

var_data_type ::= data_type | **var** data_type_or_implicit

signing ::= **signed** | **unsigned**

simple_type ::= integer_type | non_integer_type | ps_type_identifier | ps_parameter_identifier

struct_union_member[16] ::=
{ attribute_instance } [random_qualifier] data_type_or_void list_of_variable_decl_assignments **;**

data_type_or_void ::= data_type | **void**

struct_union ::= **struct** | **union** [**tagged**]

type_reference ::=
type (expression[17])
| **type** (data_type)

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
| ps_identifier
| time_literal
| **1step**

A.2.3 Declaration lists

NOTE: Replaced optional *range* AMS syntax with zero-to-many list of *unpacked_dimension* SystemVerilog syntax item.

list_of_branch_identifiers ::= branch_identifier { unpacked_dimension } { **,** branch_identifier { unpacked_dimension } }

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

list_of_genvar_identifiers ::= genvar_identifier { **,** genvar_identifier }

list_of_interface_identifiers ::= interface_identifier { unpacked_dimension }
{ **,** interface_identifier { unpacked_dimension } }

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

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

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

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

list_of_udp_port_identifiers ::= port_identifier { **,** port_identifier }

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

list_of_tf_variable_identifiers ::= port_identifier { variable_dimension } [= expression]
{ , port_identifier { variable_dimension } [= expression] }

list_of_type_assignments ::= type_assignment { , type_assignment }

list_of_variable_decl_assignments ::= variable_decl_assignment { , variable_decl_assignment }

list_of_variable_identifiers ::= variable_identifier { variable_dimension }
{ , variable_identifier { variable_dimension } }

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

list_of_virtual_interface_decl ::=
variable_identifier [= interface_instance_identifier]
{ , variable_identifier [= interface_instance_identifier] }

A.2.4 Declaration assignments

defparam_assignment ::= hierarchical_parameter_identifier = constant_mintypmax_expression

NOTE: Changed *dimension* AMS syntax item to SystemVerilog *unpacked_dimension* syntax items.

net_decl_assignment ::=
net_identifier { unpacked_dimension } [= expression]
| net_identifier { unpacked_dimension } [= constant_arrayinit]

NOTE: Changed *dimension* AMS syntax item to SystemVerilog *unpacked_dimension* syntax items.

ams_net_identifier ::=
net_identifier { unpacked_dimension }
| hierarchical_net_identifier

NOTE: Replace *range* AMS syntax item with *unpacked_dimension* syntax item.

param_assignment ::=
parameter_identifier { unpacked_dimension } [= constant_param_expression][18] { param_value_range }
| parameter_identifier unpacked_dimension = constant_arrayinit { param_value_range }

specparam_assignment ::=
specparam_identifier = constant_mintypmax_expression
| pulse_control_specparam

type_assignment ::=
type_identifier [= data_type][18]

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

variable_decl_assignment ::=
variable_identifier { variable_dimension } [= expression]
| dynamic_array_variable_identifier unsized_dimension { variable_dimension }

[= dynamic_array_new]
| class_variable_identifier [= class_new]

class_new[19] ::= **new** [**(** list_of_arguments **)** | expression]

dynamic_array_new ::= **new** **[** expression **]** [**(** expression **)**]

A.2.5 Declaration ranges

unpacked_dimension ::=
[constant_range **]**
| **[** constant_expression **]**

packed_dimension[20] ::=
[constant_range **]**
| unsized_dimension

associative_dimension ::=
[data_type **]**
| **[*]**

variable_dimension ::=
unsized_dimension
| unpacked_dimension
| associative_dimension
| queue_dimension

queue_dimension ::= **[$** [**:** constant_expression] **]**

unsized_dimension ::= **[ ]**

NOTE: AMS *value_range* syntax item name is already used in SystemVerilog, changing to *param_value_range* syntax item name.

param_value_range ::=
param_value_range_type **(** param_value_range_expression **:** param_value_range_expression **)**
| param_value_range_type **(** param_value_range_expression **:** param_value_range_expression **]**
| param_value_range_type **[** param_value_range_expression **:** param_value_range_expression **)**
| param_value_range_type **[** param_value_range_expression **:** param_value_range_expression **]**
| param_value_range_type **'{** string { **,** string } **}**
| **exclude** constant_expression

param_value_range_type :: = **from** | **exclude**

param_value_range_expression ::= constant_expression | **-inf** | **inf**

A.2.6 Function declarations

REVIEW: Add mantis item for function_declaration. Call digital function from analog, etc

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_data_type_or_implicit ::=
    data_type_or_void
  | implicit_data_type

function_declaration ::= **function** [lifetime] function_body_declaration

function_body_declaration ::=
    function_data_type_or_implicit
      [interface_identifier . | class_scope] function_identifier ;
    { tf_item_declaration }
    { function_statement_or_null }
    endfunction [: function_identifier]
  | function_data_type_or_implicit
      [interface_identifier . | class_scope] function_identifier ([tf_port_list]) ;
    { block_item_declaration }
    { function_statement_or_null }
    endfunction [: function_identifier]

function_prototype ::= **function** data_type_or_void function_identifier ([tf_port_list])

dpi_import_export ::=
    import dpi_spec_string [dpi_function_import_property] [c_identifier =] dpi_function_proto ;
  | **import** dpi_spec_string [dpi_task_import_property] [c_identifier =] dpi_task_proto ;
  | **export** dpi_spec_string [c_identifier =] **function** function_identifier ;
  | **export** dpi_spec_string [c_identifier =] **task** task_identifier ;

dpi_spec_string ::= **"DPI-C"** | **"DPI"**

dpi_function_import_property ::= **context** | **pure**

dpi_task_import_property ::= **context**

dpi_function_proto[21,22] ::= function_prototype

dpi_task_proto[22] ::= task_prototype

A.2.7 Task declarations

task_declaration ::= **task** [lifetime] task_body_declaration

task_body_declaration ::=
    [interface_identifier . | class_scope] task_identifier ;
    { tf_item_declaration }
    { statement_or_null }
    endtask [: task_identifier]
  | [interface_identifier . | class_scope] task_identifier ([tf_port_list]) ;
    { block_item_declaration }
    { statement_or_null }
    endtask [: task_identifier]

tf_item_declaration ::=
block_item_declaration
| tf_port_declaration

tf_port_list ::=
tf_port_item { , tf_port_item }

QUESTION: Move the optional discipline_identifier into data_type_or_implicit syntax item to be consistent with net_declarations?

tf_port_item[23] ::=
{ attribute_instance }
[tf_port_direction] [discipline_identifier] [**var**] data_type_or_implicit
[port_identifier { variable_dimension } [= expression]]

tf_port_direction ::= port_direction | **const ref**

QUESTION: Move the optional discipline_identifier into data_type_or_implicit syntax item to be consistent with net_declarations?

tf_port_declaration ::=
{ attribute_instance } tf_port_direction [discipline_identifier] [**var**] data_type_or_implicit
list_of_tf_variable_identifiers ;

task_prototype ::= **task** task_identifier ([tf_port_list])

A.2.8 Block item declarations

NOTE: The *integer_declaration* and *real_declaration* syntax items are not defined in SystemVerilog syntax, instead use data_declaration syntax item. Need to constrain what SystemVerilog declaration other than real and integer declaration are to be support in analog_block_item_declaration syntax item

analog_block_item_declaration ::=
{ attribute_instance } parameter_declaration ;
| { attribute_instance } data_declaration

block_item_declaration ::=
{ attribute_instance } data_declaration
| { attribute_instance } local_parameter_declaration ;
| { attribute_instance } parameter_declaration ;
| { attribute_instance } overload_declaration
| { attribute_instance } let_declaration

overload_declaration ::=
bind overload_operator **function** data_type function_identifier (overload_proto_formals) ;

overload_operator ::= **+** | **++** | **-** | **- -** | ***** | ****** | **/** | **%** | **==** | **!=** | **<** | **<=** | **>** | **>=** | **=**

overload_proto_formals ::= data_type {, data_type}

A.2.9 Interface declarations

virtual_interface_declaration ::=
virtual [**interface**] interface_identifier [parameter_value_assignment] [. modport_identifier]
list_of_virtual_interface_decl ;

modport_declaration ::= **modport** modport_item { , modport_item } ;

modport_item ::= modport_identifier (modport_ports_declaration { , modport_ports_declaration })

modport_ports_declaration ::=
{ attribute_instance } modport_simple_ports_declaration
| { attribute_instance } modport_tf_ports_declaration
| { attribute_instance } modport_clocking_declaration

modport_clocking_declaration ::= **clocking** clocking_identifier

modport_simple_ports_declaration ::=
port_direction modport_simple_port { **,** modport_simple_port }

modport_simple_port ::=
port_identifier
| **.** port_identifier **(** [expression] **)**

modport_tf_ports_declaration ::=
import_export modport_tf_port { **,** modport_tf_port }

modport_tf_port ::=
method_prototype
| tf_identifier

import_export ::= **import** | **export**

A.2.10 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 ::=
ps_or_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 ::=
operator_assignment
| inc_or_dec_expression
| subroutine_call

sequence_instance ::=
ps_or_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_decl_assignments ;

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.2.11 Covergroup declarations

covergroup_declaration ::=
covergroup covergroup_identifier [([tf_port_list])] [coverage_event] ;
{ coverage_spec_or_option }
endgroup [: covergroup_identifier]

coverage_spec_or_option ::=
{attribute_instance} coverage_spec
| {attribute_instance} coverage_option ;

coverage_option ::=
option.member_identifier = expression
| type_option.member_identifier = constant_expression

coverage_spec ::=
cover_point
| cover_cross

coverage_event ::=
clocking_event

| **with function sample (** [tf_port_list] **)**
| **@@(** block_event_expression **)**

block_event_expression ::=
block_event_expression **or** block_event_expression
| **begin** hierarchical_btf_identifier
| **end** hierarchical_btf_identifier

hierarchical_btf_identifier ::=
hierarchical_tf_identifier
| hierarchical_block_identifier
| hierarchical_identifier [class_scope] method_identifier

cover_point ::= [cover_point_identifier **:**] **coverpoint** expression [**iff (** expression **)**] bins_or_empty

bins_or_empty ::=
{ {attribute_instance} { bins_or_options **;** } **}**
| **;**

bins_or_options ::=
coverage_option
| [**wildcard**] bins_keyword bin_identifier [**[** [expression] **]**] **=** **{** open_range_list **}** [**iff (** expression **)**]
| [**wildcard**] bins_keyword bin_identifier [**[ ]**] **=** trans_list [**iff (** expression **)**]
| bins_keyword bin_identifier [**[** [expression] **]**] **=** **default** [**iff (** expression **)**]
| bins_keyword bin_identifier **=** **default sequence** [**iff (** expression **)**]

bins_keyword::= **bins** | **illegal_bins** | **ignore_bins**

range_list ::= value_range { **,** value_range }

trans_list ::= **(** trans_set **)** { **,** **(** trans_set **)** }

trans_set ::= trans_range_list { **=>** trans_range_list }

trans_range_list ::=
trans_item
| trans_item **[*** repeat_range **]**
| trans_item **[->** repeat_range **]**
| trans_item **[=** repeat_range **]**

trans_item ::= range_list

repeat_range ::=
expression
| expression **:** expression

REVIEW: Investigate is AMS cross keyword is conflict with P1800 cross keyword

cover_cross ::=
[cross_identifier **:**] **cross** list_of_coverpoints [**iff (** expression **)**] select_bins_or_empty

list_of_coverpoints ::= cross_item **,** cross_item { **,** cross_item }

cross_item ::=
cover_point_identifier
| variable_identifier

select_bins_or_empty ::=
{ { bins_selection_or_option **;** } **}**

| ;

bins_selection_or_option ::=
{ attribute_instance } coverage_option
| { attribute_instance } bins_selection

bins_selection ::= bins_keyword bin_identifier = select_expression [iff (expression)]

select_expression ::=
select_condition
| ! select_condition
| select_expression && select_expression
| select_expression || select_expression
| (select_expression)

select_condition ::= binsof (bins_expression) [intersect { open_range_list }]

bins_expression ::=
variable_identifier
| cover_point_identifier [. bin_identifier]

open_range_list ::= open_value_range { , open_value_range }

open_value_range ::= value_range[24]

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_instance] (output_terminal , input_terminal ,
ncontrol_terminal , pcontrol_terminal)

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

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

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

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

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

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

pull_gate_instance ::= [name_of_instance] (output_terminal)

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 Instantiations

A.4.1 Instantiation

A.4.1.1 Module instantiation

module_instantiation ::=
module_or_paramset_identifier [parameter_value_assignment] hierarchical_instance { , hierarchical_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 ::= param_expression

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

hierarchical_instance ::= name_of_instance ([list_of_port_connections])

name_of_instance ::= instance_identifier { unpacked_dimension }

list_of_port_connections[25] ::=
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])]
| { attribute_instance } .*

A.4.1.2 Interface instantiation

interface_instantiation ::=
interface_identifier [parameter_value_assignment] hierarchical_instance { , hierarchical_instance } ;

A.4.1.3 Program instantiation

program_instantiation ::=
program_identifier [parameter_value_assignment] hierarchical_instance { , hierarchical_instance } ;

A.4.1.4 Checker instantiation

checker_instantiation ::=
checker_identifier name_of_instance ([list_of_checker_port_connections]) ;

list_of_checker_port_connections[25] ::=
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.4.2 Generated instantiation

generate_region ::=
generate { generate_item } endgenerate

QUESTION: Does *analog_loop_generate_statement* syntax item support the SystemVerilog *genvar_iteration* syntax item?

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**] genvar_identifier **=** constant_expression

genvar_iteration ::=
    genvar_identifier assignment_operator genvar_expression
  | inc_or_dec_operator genvar_identifier
  | genvar_identifier inc_or_dec_operator

conditional_generate_construct ::=
    if_generate_construct
  | case_generate_construct

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

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

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

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

generate_item[26] ::=
    module_or_generate_item
  | interface_or_generate_item
  | checker_or_generate_item

A.5 UDP declaration and instantiation

A.5.1 UDP declaration

udp_nonansi_declaration ::=
    { attribute_instance } **primitive** udp_identifier **(** udp_port_list **)** **;**

udp_ansi_declaration ::=
    { attribute_instance } **primitive** udp_identifier **(** udp_declaration_port_list **)** **;**

udp_declaration ::=
    udp_nonansi_declaration udp_port_declaration { udp_port_declaration }
        udp_body
    endprimitive [**:** udp_identifier]
  | udp_ansi_declaration

udp_body
endprimitive [**:** udp_identifier]
| **extern** udp_nonansi_declaration
| **extern** udp_ansi_declaration
| { attribute_instance } **primitive** udp_identifier **(.*) ;**
{ udp_port_declaration }
udp_body
endprimitive [**:** udp_identifier]

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]

REVIEW: add optional discipline_identifier syntax item

udp_input_declaration ::= { attribute_instance } **input** [discipline_identifier] list_of_udp_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 ::= **x** | **X** | **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_instance] **(** output_terminal **,** input_terminal { **,** input_terminal } **)**

A.6 Behavioral statements

A.6.1 Continuous assignment and net alias statements

continuous_assign ::=

assign [drive_strength] [delay3] list_of_net_assignments **;**

| **assign** [delay_control] list_of_variable_assignments **;**

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

list_of_variable_assignments ::= variable_assignment { **,** variable_assignment }

net_alias ::= **alias** net_lvalue **=** net_lvalue { **=** net_lvalue } **;**

net_assignment ::= net_lvalue **=** expression

A.6.2 Procedural blocks and assignments

analog_construct ::= **analog** analog_statement

analog_initial_construct ::= **analog initial** analog_function_statement

analog_procedural_assignment ::= analog_variable_assignment **;**

analog_variable_assignment ::= analog_variable_lvalue **=** analog_expression

initial_construct ::= **initial** statement_or_null

always_construct ::= always_keyword statement

always_keyword ::= **always** | **always_comb** | **always_latch** | **always_ff**

final_construct ::= **final** function_statement

blocking_assignment ::=

variable_lvalue **=** delay_or_event_control expression

| nonrange_variable_lvalue **=** dynamic_array_new

| [implicit_class_handle **.** | class_scope | package_scope] hierarchical_variable_identifier

select **=** class_new

| operator_assignment

operator_assignment ::= variable_lvalue assignment_operator expression

assignment_operator ::=

= | **+=** | **-=** | ***=** | **/=** | **%=** | **&=** | **|=** | **^=** | **<<=** | **>>=** | **<<<=** | **>>>=**

nonblocking_assignment ::=

variable_lvalue **<=** [delay_or_event_control] expression

procedural_continuous_assignment ::=

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

NOTE: To be consistent with SystemVerilog *seq_block* syntax item, moved *analog_block_item_declaration* syntax item out of the optional condition containing the *analog_block_identifier* syntax item.

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

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

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

par_block ::=
fork [**:** block_identifier] { block_item_declaration } { statement_or_null }
join_keyword [**:** block_identifier]

join_keyword ::= **join** | **join_any** | **join_none**

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_or_null ::=
statement
| { attribute_instance } ;

statement ::= [block_identifier :] { attribute_instance } statement_item

statement_item ::=
blocking_assignment ;
| nonblocking_assignment ;
| procedural_continuous_assignment ;
| case_statement
| conditional_statement
| inc_or_dec_expression ;
| subroutine_call_statement
| disable_statement
| event_trigger
| loop_statement
| jump_statement
| par_block
| procedural_timing_control_statement
| seq_block
| wait_statement
| procedural_assertion_statement
| clocking_drive ;
| randsequence_statement
| randcase_statement

| expect_property_statement

function_statement ::= statement

function_statement_or_null ::=
function_statement
| { attribute_instance } ;

variable_identifier_list ::= variable_identifier { , variable_identifier }

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

REVIEW: Investigate is AMS cross keyword is conflict with P1800 cross keyword

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]]])

procedural_timing_control_statement ::=
procedural_timing_control statement_or_null

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

delay_control ::=
delay_value
| # (mintypmax_expression)

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

event_expression[27] ::=
[edge_identifier] expression [**iff** expression]
| sequence_instance [**iff** expression]
| event_expression **or** event_expression
| event_expression **,** event_expression
| **(** event_expression **)**
| analog_event_functions
| driver_update expression
| analog_variable_lvalue

procedural_timing_control ::=
delay_control
| event_control
| cycle_delay

jump_statement ::=
return [expression] **;**
| **break ;**
| **continue ;**

wait_statement ::=
wait (expression **)** statement_or_null
| **wait fork ;**
| **wait_order (** hierarchical_identifier { **,** hierarchical_identifier } **)** action_block

event_trigger ::=
-> hierarchical_event_identifier **;**
| **->>** [delay_or_event_control] hierarchical_event_identifier **;**

disable_statement ::=
disable hierarchical_task_identifier **;**
| **disable** hierarchical_block_identifier **;**
| **disable fork ;**

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 ::=
[unique_priority] **if (** cond_predicate **)** statement_or_null
{ **else if (** cond_predicate **)** statement_or_null }
[**else** statement_or_null]

unique_priority ::= **unique** | **unique0** | **priority**

cond_predicate ::=

expression_or_cond_pattern { **&&&** expression_or_cond_pattern }

expression_or_cond_pattern ::=
expression | cond_pattern

cond_pattern ::= expression **matches** pattern

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 ::=
[unique_priority] case_keyword (case_expression)
case_item { case_item } **endcase**
| [unique_priority] case_keyword (case_expression)**matches**
case_pattern_item { case_pattern_item } **endcase**
| [unique_priority] **case** (case_expression) **inside**
case_inside_item { case_inside_item } **endcase**

case_keyword ::= **case** | **casez** | **casex**

case_expression ::= expression

case_item ::=
case_item_expression { , case_item_expression } : statement_or_null
| **default** [:] statement_or_null

case_pattern_item ::=
pattern [**&&&** expression] : statement_or_null
| **default** [:] statement_or_null

case_inside_item ::=
open_range_list : statement_or_null
| **default** [:] statement_or_null

case_item_expression ::= expression

randcase_statement ::=
randcase randcase_item { randcase_item } **endcase**

randcase_item ::= expression : statement_or_null

A.6.7.1 Patterns

pattern ::=

. variable_identifier
| .*
| constant_expression
| tagged member_identifier [pattern]
| '{ pattern { , pattern } }
| '{ member_identifier : pattern { , member_identifier : pattern } }

assignment_pattern ::=
'{ expression { , expression } }
| '{ structure_pattern_key : expression { , structure_pattern_key : expression } }
| '{ array_pattern_key : expression { , array_pattern_key : expression } }
| '{ constant_expression { expression { , expression } } }

structure_pattern_key ::= member_identifier | assignment_pattern_key

array_pattern_key ::= constant_expression | assignment_pattern_key

assignment_pattern_key ::= simple_type | default

assignment_pattern_expression ::=
[assignment_pattern_expression_type] assignment_pattern

assignment_pattern_expression_type ::=
ps_type_identifier
| ps_parameter_identifier
| integer_atom_type
| type_reference

constant_assignment_pattern_expression[28] ::= assignment_pattern_expression

assignment_pattern_net_lvalue ::=
'{ net_lvalue { , net_lvalue } }

assignment_pattern_variable_lvalue ::=
'{ variable_lvalue { , variable_lvalue } }

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_or_null
| repeat (expression) statement_or_null
| while (expression) statement_or_null
| for (for_initialization ; expression ; for_step)
statement_or_null

| do statement_or_null while (expression) ;
| foreach (ps_or_hierarchical_array_identifier [loop_variables]) statement

for_initialization ::=
list_of_variable_assignments
| for_variable_declaration { , for_variable_declaration }

for_variable_declaration ::=
data_type variable_identifier = expression { , variable_identifier = expression }

for_step ::= for_step_assignment { , for_step_assignment }

for_step_assignment ::=
operator_assignment
| inc_or_dec_expression
| function_subroutine_call

loop_variables ::= [index_variable_identifier] { , [index_variable_identifier] }

A.6.9 Subroutine call statements

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

subroutine_call_statement ::=
subroutine_call ;
| void ' (function_subroutine_call) ;

A.6.10 Contribution statement

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 select

A.6.13 Randsequence

randsequence_statement ::= randsequence ([production_identifier])
production { production }
endsequence

production ::= [data_type_or_void] production_identifier [(tf_port_list)] : rs_rule { | rs_rule } ;

rs_rule ::= rs_production_list [:= weight_specification [rs_code_block]]

rs_production_list ::=
rs_prod { rs_prod }
| rand join [(expression)] production_item production_item { production_item }

weight_specification ::=
integral_number
| ps_identifier
| (expression)

rs_code_block ::= { { data_declaration } { statement_or_null } }

rs_prod ::=
production_item
| rs_code_block
| rs_if_else
| rs_repeat
| rs_case

production_item ::= production_identifier [(list_of_arguments)]

rs_if_else ::= if (expression) production_item [else production_item]

rs_repeat ::= repeat (expression) production_item

rs_case ::= case (case_expression) rs_case_item { rs_case_item } endcase

rs_case_item ::=
case_item_expression { , case_item_expression } : production_item ;
| default [:] production_item ;

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 | interface_identifier **.** port_identifier

output_identifier ::= output_port_identifier | inout_port_identifier | interface_identifier **.** 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 [polarity_operator] =>
      (specify_output_terminal_descriptor [polarity_operator] : data_source_expression))

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

data_source_expression ::= expression

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

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] [**,** [timestamp_condition] [**,** [timecheck_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] [**,** [timestamp_condition] [**,** [timecheck_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

timecheck_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_mintypmax_expression

timestamp_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**

| edge_control_specifier

specify_terminal_descriptor ::=
    specify_input_terminal_descriptor
| specify_output_terminal_descriptor

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

edge_descriptor[29] ::= **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 **}**

NOTE: The *msb_constant_expression* and *lsb_constant_expression* syntax item is not present within SystemVerilog syntax, replace with *constant_range* syntax item.

analog_filter_function_arg ::=
    parameter_identifier
| parameter_identifier **[** constant_range **]**
| 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 ::= **{** expression concatenation **}**[30]

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

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

streaming_concatenation ::= **{** stream_operator [slice_size] stream_concatenation **}**

stream_operator ::= >> | <<

slice_size ::= simple_type | constant_expression

stream_concatenation ::= { stream_expression { , stream_expression } }

stream_expression ::= expression [with [array_range_expression]]

array_range_expression ::=
    expression
  | expression : expression
  | expression +: expression
  | expression -: expression

empty_queue[31] ::= { }

A.8.2 Subroutine 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**

NOTE: Changed the *nature_attribute_identifier* to *nature_access_identifier* to be more accurate.

branch_probe_function_call ::=
    nature_access_identifier **(** branch_reference **)**
  | nature_access_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_analog_built_in_function_call ::=
    analog_built_in_function_name **(** constant_expression [**,** constant_expression] **)**

constant_function_call ::= function_subroutine_call[32]

tf_call[33] ::= ps_or_hierarchical_tf_identifier { attribute_instance } [**(** list_of_arguments **)**]

system_tf_call ::=
    system_tf_identifier [**(** list_of_arguments **)**]
  | system_tf_identifier **(** data_type [**,** expression] **)**

subroutine_call ::=
    tf_call
  | system_tf_call
  | method_call
  | [**std::**] randomize_call

function_subroutine_call ::= subroutine_call

list_of_arguments ::=
    [expression] { **,** [expression] } { **,** **.** identifier **(** [expression] **)** }
  | **.** identifier **(** [expression] **)** { **,** **.** identifier **(** [expression] **)** }

method_call ::= method_call_root **.** method_call_body

method_call_body ::=
    method_identifier { attribute_instance } [**(** list_of_arguments **)**]
  | built_in_method_call

built_in_method_call ::=
    array_manipulation_call
  | randomize_call

array_manipulation_call ::=
    array_method_name { attribute_instance }
      [**(** list_of_arguments **)**]
      [**with** **(** expression **)**]

randomize_call ::=
    randomize { attribute_instance }
      [**(** [variable_identifier_list | **null**] **)**]
      [**with** [**(** [identifier_list] **)**] constraint_block][34]

method_call_root ::= primary | implicit_class_handle

array_method_name ::=
    method_identifier | **unique** | **and** | **or** | **xor**

A.8.3 Expressions

abstol_expression ::=
    constant_expression
  | nature_identifier

analog_conditional_expression ::= analog_expression ? { attribute_instance } analog_expression : analog_expression

NOTE: The *msb_constant_expression* and *lsb_constant_expression* syntax items are not defined in SystemVerilog, replace with *constant_range* syntax item.

analog_range_expression ::=
    analog_expression
  | constant_range

analog_expression_or_null ::= [analog_expression]

QUESTION: Are the new SystemVerilog operators support in analog expressions?

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

inc_or_dec_expression ::=
    inc_or_dec_operator { attribute_instance } variable_lvalue
  | variable_lvalue { attribute_instance } inc_or_dec_operator

conditional_expression ::= cond_predicate ? { attribute_instance } expression : 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

QUESTION: Are the new SystemVerilog operators support in *analysis_or_constant_expression* syntax item?

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_param_expression ::=
    constant_mintypmax_expression | data_type | $

param_expression ::= mintypmax_expression | data_type

constant_range_expression ::=
    constant_expression
  | constant_part_select_range

constant_part_select_range ::=
    constant_range
  | constant_indexed_range

constant_range ::= constant_expression : constant_expression

constant_indexed_range ::=
    constant_expression +: constant_expression
  | constant_expression -: constant_expression

expression ::=
    primary
  | unary_operator { attribute_instance } primary
  | inc_or_dec_expression
  | (operator_assignment)
  | expression binary_operator { attribute_instance } expression
  | conditional_expression
  | inside_expression
  | tagged_union_expression

tagged_union_expression ::=
    tagged member_identifier [expression]

inside_expression ::= expression inside { open_range_list }

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]]]])

value_range ::=
    expression
  | [expression : 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

nature_attribute_expression ::=
constant_expression
| nature_identifier
| nature_access_identifier

part_select_range ::= constant_range | indexed_range

indexed_range ::=
expression +: constant_expression
| expression -: constant_expression

genvar_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 ::=
primary_literal
| ps_parameter_identifier constant_select
| specparam_identifier [[constant_range_expression]]
| genvar_identifier[35]
| [package_scope | class_scope] enum_identifier
| constant_concatenation [[constant_range_expression]]
| constant_multiple_concatenation [[constant_range_expression]]
| constant_function_call
| constant_let_expression
| constant_analog_built_in_function_call
| (constant_mintypmax_expression)
| constant_cast
| constant_assignment_pattern_expression

| type_reference[36]
| system_parameter_identifier
| nature_attribute_reference
| constant_analog_function_call

module_path_primary ::=
number
| identifier
| module_path_concatenation
| module_path_multiple_concatenation
| function_subroutine_call
| (module_path_mintypmax_expression)

primary ::=
primary_literal
| [class_qualifier | package_scope] hierarchical_identifier select
| empty_queue
| concatenation [[range_expression]]
| multiple_concatenation [[range_expression]]
| function_subroutine_call
| let_expression
| (mintypmax_expression)
| cast
| assignment_pattern_expression
| streaming_concatenation
| sequence_method_call
| **this**[37]
| **$**[38]
| **null**
| branch_probe_function_call
| port_probe_function_call
| nature_attribute_reference
| analog_function_call
| analog_built_in_function_call

class_qualifier := [**local::**[39]] [implicit_class_handle **.** | class_scope]

range_expression ::=
expression
| part_select_range

primary_literal ::= number | time_literal | unbased_unsized_literal | string_literal

time_literal[40] ::=
unsigned_number time_unit
| fixed_point_number time_unit

time_unit ::= **s** | **ms** | **us** | **ns** | **ps** | **fs**

implicit_class_handle[37] ::= **this** | **super** | **this . super**

bit_select ::= { [expression] }

select ::=

[{ . member_identifier bit_select } . member_identifier] bit_select [[part_select_range]]

nonrange_select ::=
[{ . member_identifier bit_select } . member_identifier] bit_select

constant_bit_select ::= { [constant_expression] }

constant_select ::=
[{ . member_identifier constant_bit_select } . member_identifier] constant_bit_select
[[constant_part_select_range]]

constant_cast ::=
casting_type ' (constant_expression)

constant_let_expression ::= let_expression[41]

cast ::=
casting_type ' (expression)

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 ::=
ps_or_hierarchical_net_identifier constant_select
| { net_lvalue { , net_lvalue } }
| [assignment_pattern_expression_type] assignment_pattern_net_lvalue

variable_lvalue ::=
[implicit_class_handle . | package_scope] hierarchical_variable_identifier select[42]
| { variable_lvalue { , variable_lvalue } }
| [assignment_pattern_expression_type] assignment_pattern_variable_lvalue
| streaming_concatenation[43]

nonrange_variable_lvalue ::=
[implicit_class_handle . | package_scope] hierarchical_variable_identifier nonrange_select

A.8.6 Operators

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

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

inc_or_dec_operator ::= ++ | --

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

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

A.8.7 Numbers

number ::=
integral_number
| real_number

integral_number ::=
decimal_number
| octal_number
| binary_number
| hex_number

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[29] ::= non_zero_decimal_digit { _ | decimal_digit}

real_number[29] ::=
fixed_point_number
| unsigned_number [. unsigned_number] exp [sign] unsigned_number
| unsigned_number [. unsigned_number] scale_factor

fixed_point_number[29] ::= unsigned_number . unsigned_number

exp ::= e | E

unsigned_number[29] ::= decimal_digit { _ | decimal_digit }

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

binary_value[29] ::= binary_digit { _ | binary_digit }

octal_value[29] ::= octal_digit { _ | octal_digit }

hex_value[29] ::= hex_digit { _ | hex_digit }

decimal_base[29] ::= '[s|S]d | '[s|S]D

binary_base[29] ::= '[s|S]b | '[s|S]B

octal_base[29] ::= '[s|S]o | '[s|S]O

hex_base[29] ::= '[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 | ?

unbased_unsized_literal ::= '0 | '1 | 'z_or_x [44]

A.8.8 Strings

string_literal ::= " { 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]

NOTE: The *real_identifier* syntax item is no longer separate from the *variable_identifier* syntax item in SystemVerilog, remove *real_identifier* variable reference.

variable_reference ::=
    variable_identifier
  | variable_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

array_identifier ::= identifier

block_identifier ::= identifier

bin_identifier ::= identifier

branch_identifier ::= identifier

c_identifier[45] ::= [a-zA-Z_] { [a-zA-Z0-9_] }

cell_identifier ::= identifier

checker_identifier ::= identifier

class_identifier ::= identifier

class_variable_identifier ::= variable_identifier

clocking_identifier ::= identifier

config_identifier ::= identifier

connectmodule_identifier ::= identifier

connectrules_identifier ::= identifier

const_identifier ::= identifier

constraint_identifier ::= identifier

covergroup_identifier ::= identifier

covergroup_variable_identifier ::= variable_identifier

cover_point_identifier ::= identifier

cross_identifier ::= identifier

discipline_identifier ::= identifier

dynamic_array_variable_identifier ::= variable_identifier

enum_identifier ::= identifier

escaped_identifier ::= \ {any_ASCII_character_except_white_space} white_space

formal_identifier ::= identifier

function_identifier ::= identifier

generate_block_identifier ::= identifier

genvar_identifier ::= identifier

hierarchical_array_identifier ::= hierarchical_identifier

hierarchical_block_identifier ::= hierarchical_identifier

hierarchical_branch_identifier ::= hierarchical_identifier

hierarchical_event_identifier ::= hierarchical_identifier

hierarchical_identifier ::= [**$root .**] { identifier constant_bit_select **.** } 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_tf_identifier ::= hierarchical_identifier

hierarchical_variable_identifier ::= hierarchical_identifier

identifier ::=
simple_identifier
| escaped_identifier

index_variable_identifier ::= identifier

interface_identifier ::= identifier

interface_instance_identifier ::= identifier

inout_port_identifier ::= identifier

input_port_identifier ::= identifier

instance_identifier ::= identifier

library_identifier ::= identifier

member_identifier ::= identifier

method_identifier ::= identifier

modport_identifier ::= identifier

module_identifier ::= identifier

module_or_paramset_identifier ::=
module_identifier
| paramset_identifier

module_parameter_identifier ::= identifier

nature_access_identifier ::= identifier

nature_attribute_identifier ::= **abstol** | **access** | **ddt_nature** | **idt_nature** | **units** | identifier

nature_identifier ::= identifier

net_identifier ::= identifier

output_port_identifier ::= identifier

package_identifier ::= identifier

package_scope ::=
    package_identifier `::`
  | `$unit ::`

parameter_identifier ::= identifier

paramset_identifier ::= identifier

port_identifier ::= identifier

production_identifier ::= identifier

program_identifier ::= identifier

property_identifier ::= identifier

ps_class_identifier ::= [package_scope] class_identifier

ps_covergroup_identifier ::= [package_scope] covergroup_identifier

ps_identifier ::= [package_scope] identifier

ps_or_hierarchical_array_identifier ::=
    [implicit_class_handle `.` | class_scope | package_scope] hierarchical_array_identifier

ps_or_hierarchical_net_identifier ::= [package_scope] net_identifier | hierarchical_net_identifier

ps_or_hierarchical_property_identifier ::=
    [package_scope] property_identifier
  | hierarchical_property_identifier

ps_or_hierarchical_sequence_identifier ::=
    [package_scope] sequence_identifier
  | hierarchical_sequence_identifier

ps_or_hierarchical_tf_identifier ::= [package_scope] tf_identifier | hierarchical_tf_identifier

ps_parameter_identifier ::=
    [package_scope | class_scope] parameter_identifier
  | { generate_block_identifier [`[` constant_expression `]`] `.` } parameter_identifier

ps_type_identifier ::= [`local::`[39] | package_scope] type_identifier

sequence_identifier ::= identifier

signal_identifier ::= identifier

simple_identifier[45] ::= [`a-zA-Z_`] { [`a-zA-Z0-9_$`] }

specparam_identifier ::= identifier

system_parameter_identifier ::= `$` [`a-zA-Z0-9_$`] { [`a-zA-Z0-9_$`] }

system_tf_identifier[46] ::= `$`[`a-zA-Z0-9_$`]{ [`a-zA-Z0-9_$`] }

task_identifier ::= identifier

tf_identifier ::= identifier

terminal_identifier ::= identifier

topmodule_identifier ::= identifier

type_identifier ::= identifier

udp_identifier ::= identifier

variable_identifier ::= identifier

A.9.4 White space

white_space ::= space | tab | newline | eof[47]

A.10 Footnotes (normative)

1) A package_import_declaration in a module_ansi_header, interface_ansi_header, or program_ansi_header shall be followed by a parameter_port_list or list_of_port_declarations, or both.

2) The list_of_port_declarations syntax is explained in CrossRefparanumonlyefault ? Font, which also imposes various semantic restrictions, e.g., a **ref** port shall be of a variable type and an **inout** port shall not be. It shall be illegal to initialize a port that is not a variable **output** port or to specify a default value for a port that is not an **input** port.

3) A timeunits_declaration shall be legal as a non_port_module_item, non_port_interface_item, non_port_program_item, or package_item only if it repeats and matches a previous timeunits_declaration within the same time scope.

4) If the bind_target_scope is an interface_identifier or the bind_target_instance is an interface_instance_identifier, then the bind_instantiation shall be an interface_instantiation or a checker_instantiation.

5) It shall be illegal for a program_generate_item to include any item that would be illegal in a program_declaration outside a program_generate_item.

6) It shall be illegal for a checker_generate_item to include any item that would be illegal in a checker_declaration outside a checker_generate_item.

7) In a parameter_declaration that is a class_item, the **parameter** keyword shall be a synonym for the **localparam** keyword.

8) In any one declaration, only one of **protected** or **local** is allowed, only one of **rand** or **randc** is allowed, and **static** and/or **virtual** can appear only once.

9) In a data_declaration that is not within a procedural context, it shall be illegal to use the **automatic** keyword. In a data_declaration, it shall be illegal to omit the explicit data_type before a list_of_variable_decl_assignments unless the **var** keyword is used.

10) It shall be illegal to have an import statement directly within a class scope.

11) A charge strength shall only be used with the **trireg** keyword. When the **vectored** or **scalared** keyword is used, there shall be at least one packed dimension.

12) When a packed dimension is used with the **struct** or **union** keyword, the **packed** keyword shall also be used.

13) When a type_reference is used in a net declaration, it shall be preceded by a net type keyword; and when it is used in a variable declaration, it shall be preceded by the **var** keyword.

14) A type_identifier shall be legal as an enum_base_type if it denotes an integer_atom_type, with which an additional packed dimension is not permitted, or an integer_vector_type.

15) When a net_port_type contains a data_type, it shall only be legal to omit the explicit net_type when declaring an **inout** port.

16) It shall be legal to declare a **void** struct_union_member only within tagged unions.

17) An expression that is used as the argument in a type_reference shall not contain any hierarchical references or references to elements of dynamic objects.

18) In a param_assignment it shall be illegal to omit the constant_param_expression except within a parameter_declaration in a parameter_port_list. In a type_assignment it shall be illegal to omit the data_type except within a parameter_declaration in a parameter_port_list.

19) In a shallow copy, the expression shall evaluate to an object handle.

20) In packed_dimension, unsized_dimension is permitted only as the sole packed dimension in a DPI import declaration; see dpi_function_proto and dpi_task_proto.

21) dpi_function_proto return types are restricted to small values, per CrossRefparanumonlyefault ? Font.

22) Formals of dpi_function_proto and dpi_task_proto cannot use pass-by-reference mode, and class types cannot be passed at all; see CrossRefparanumonlyefault ? Font for a description of allowed types for DPI formal arguments.

23) In a tf_port_item, it shall be illegal to omit the explicit port_identifier except within a function_prototype or task_prototype.

24) It shall be legal to use the **`$`** primary in an open_value_range of the form **`[`** expression **`: $ ]`** or **`[ $ :`** expression **`]`**.

25) The **`.*`** token shall appear at most once in a list of port connections.

26) Within an interface_declaration, it shall only be legal for a generate_item to be an interface_or_generate_item. Within a module_declaration, except when also within an interface_declaration, it shall only be legal for a generate_item to be a module_or_generate_item. Within a checker_declaration, it shall only be legal for a generate_item to be a checker_or_generate_item.

27) Parentheses are required when an event expression that contains comma-separated event expressions is passed as an actual argument using positional binding.

28) In a constant_assignment_pattern_expression, all member expressions shall be constant expressions.

29) Embedded spaces are illegal.

30) In a multiple_concatenation, it shall be illegal for the multiplier not to be a constant_expression unless the type of the concatenation is **`string`**.

31) **`{ }`** shall only be legal in the context of a queue.

32) In a constant_function_call, all arguments shall be constant_expressions.

33) It shall be illegal to omit the parentheses in a tf_call unless the subroutine is a task, void function, or class method. If the subroutine is a nonvoid class function method, it shall be illegal to omit the parentheses if the call is directly recursive.

34) In a randomize_call that is not a method call of an object of class type (i.e. a scope randomize), the optional parenthesized *identifier_list* after the keyword **`with`** shall be illegal, and the use of **`null`** shall be illegal.

35) A genvar_identifier shall be legal in a constant_primary only within a genvar_expression.

36) It shall be legal to use a type_reference constant_primary as the casting_type in a static cast. It shall be illegal for a type_reference constant_primary to be used with any operators except the equality/inequality and case equality/inequality operators.

37) implicit_class_handle shall only appear within the scope of a class_declaration or out-of-block method declaration.

38) The **`$`** primary shall be legal only in a select for a queue variable, in an open_value_range, or as an entire sequence_actual_arg or property_actual_arg.

39) The **`local::`** qualifier shall only appear within the scope of an inline constraint block.

40) The unsigned number or fixed-point number in time_literal shall not be followed by white_space.

41) 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.

42) In a variable_lvalue that is assigned within a sequence_match_item any select shall also be a constant_select.

43) A streaming_concatenation expression shall not be nested within another variable_lvalue. A streaming_concatenation shall not be the target of the increment or decrement operator nor the target of any assignment operator except the simple (**`=`**) or nonblocking assignment (**`<=`**) operator.

44) The apostrophe (**`'`**) in unbased_unsized_literal shall not be followed by white_space.

45) A simple_identifier or c_identifier shall start with an alpha or underscore (**`_`**) character, shall have at least one character, and shall not

have any spaces.

46) The **$** character in a system_tf_identifier shall not be followed by white_space. A system_tf_identifier shall not be escaped.

47) End of file.