Annex A

(normative)

Formal syntax

The formal syntax of SystemVerilog is described using Backus-Naur Form (BNF). The syntax of SystemVerilog HDL source is derived from the starting symbol source_text. The syntax of a library map file is derived from the starting symbol library_text. The conventions used are as follows:

— Keywords and punctuation are in **bold-red** text.
— Syntactic categories are named in nonbold text.
— A vertical bar (|) separates alternatives.
— Square brackets ([ ]) enclose optional items.
— Braces ({ }) enclose items that can be repeated zero or more times.

The full syntax and semantics of Verilog and SystemVerilog are not described solely using BNF. The normative text description contained within the chapters of IEEE Std 1364 and this standard provide additional details on the syntax and semantics described in this BNF.

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
module_nonansi_header ::=
        { attribute_instance } module_keyword [ lifetime ] module_identifier [ parameter_port_list ]
```

```
        list_of_ports ;
module_ansi_header ::=
        { attribute_instance } module_keyword [ lifetime ] module_identifier [ 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
module_keyword ::= module | macromodule
interface_nonansi_header ::=
        { attribute_instance } interface [ lifetime ] interface_identifier
            [ parameter_port_list ] list_of_ports ;
interface_ansi_header ::=
        {attribute_instance } interface [ lifetime ] interface_identifier
            [ parameter_port_list ] [ list_of_port_declarations ] ;
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
program_nonansi_header ::=
        { attribute_instance } program [ lifetime ] program_identifier
            [ parameter_port_list ] list_of_ports ;
program_ansi_header ::=
        {attribute_instance } program [ lifetime ] program_identifier
            [ parameter_port_list ] [ list_of_port_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
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 package_identifier ;
            [ timeunits_declaration ] { { attribute_instance } package_item }
        endpackage [ : package_identifier ]
timeunits_declaration ::=
        timeunit 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
      | data_type list_of_param_assignments
      | type list_of_type_assignments
list_of_ports ::= ( port { , port } )
list_of_port_declarations[25] ::=
        ( [ { 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 }
      | [ variable_port_header ] port_identifier { variable_dimension } [ = constant_expression ]
      | [ net_port_header | variable_port_header ] . port_identifier ( [ expression ] )
```

A.1.4 Module items

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

module_item ::=
        port_declaration ;
      | non_port_module_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

module_or_generate_item_declaration ::=
        package_or_generate_item_declaration
      | genvar_declaration
      | clocking_declaration
      | default clocking clocking_identifier ;

non_port_module_item ::=
        generate_region
      | module_or_generate_item
      | specify_block
      | { attribute_instance } specparam_declaration
      | program_declaration
      | module_declaration
      | interface_declaration
      | timeunits_declaration[17]

parameter_override ::= defparam list_of_defparam_assignments ;

bind_directive ::=
        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
```

A.1.5 Configuration source text

```
config_declaration ::=
        config config_identifier ;
            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 ]
```

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 ;
interface_item ::=
        port_declaration ;
      | non_port_interface_item
non_port_interface_item ::=
        generate_region
      | interface_or_generate_item
      | program_declaration
      | interface_declaration
      | timeunits_declaration[17]
```

A.1.7 Program items

```
program_item ::=
        port_declaration ;
      | non_port_program_item
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[17]
      | program_generate_item

program_generate_item[37] ::=
        loop_generate_construct
      | conditional_generate_construct
      | generate_region
```

A.1.8 Class items

```
class_item ::=
        { attribute_instance } class_property
      | { attribute_instance } class_method
      | { attribute_instance } class_constraint
      | { attribute_instance } class_declaration
      | { attribute_instance } timeunits_declaration[17]
      | { attribute_instance } covergroup_declaration
      | ;

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[7] ::=
        static
      | protected
      | local

property_qualifier[7] ::=
        random_qualifier
      | class_item_qualifier

random_qualifier[7] ::=
        rand
      | randc

method_qualifier[7] ::=
        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.1.9 Constraints

```
constraint_declaration ::= [ static ] constraint constraint_identifier constraint_block
constraint_block ::= { { constraint_block_item } }
constraint_block_item ::=
        solve identifier_list before identifier_list ;
      | constraint_expression
constraint_expression ::=
        expression_or_dist ;
      | expression -> constraint_set
      | if ( expression ) constraint_set [ else constraint_set ]
      | foreach ( 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 ::= [ static ] constraint constraint_identifier ;
extern_constraint_declaration ::=
        [ static ] constraint class_scope constraint_identifier constraint_block
identifier_list ::= identifier { , identifier }
```

A.1.10 Package items

```
package_item ::=
        package_or_generate_item_declaration
      | anonymous_program
      | timeunits_declaration[17]
package_or_generate_item_declaration ::=
        net_declaration
      | data_declaration
      | task_declaration
      | function_declaration
      | dpi_import_export
      | extern_constraint_declaration
      | class_declaration
```

```
        | class_constructor_declaration
        | parameter_declaration ;
        | local_parameter_declaration
        | covergroup_declaration
        | overload_declaration
        | concurrent_assertion_item_declaration
        | ;
anonymous_program ::= program ; { anonymous_program_item } endprogram
anonymous_program_item ::=
          task_declaration
        | function_declaration
        | class_declaration
        | covergroup_declaration
        | class_constructor_declaration
        | ;
```

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 ;
```

A.2.1.2 Port declarations

```
inout_declaration ::=
          inout net_port_type list_of_port_identifiers
input_declaration ::=
          input net_port_type list_of_port_identifiers
        | input variable_port_type list_of_variable_identifiers
output_declaration ::=
          output net_port_type 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

```
data_declaration[14] ::=
          [ const ] [ var ] [ lifetime ] data_type_or_implicit list_of_variable_decl_assignments ;
```

```
    | type_declaration
    | package_import_declaration
    | virtual_interface_declaration

package_import_declaration ::=
    import package_import_item { , package_import_item } ;

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

genvar_declaration ::= genvar list_of_genvar_identifiers ;

net_declaration[13] ::=
    net_type [ drive_strength | charge_strength ] [ vectored | scalared ]
        data_type_or_implicit [ delay3 ] list_of_net_decl_assignments ;

type_declaration ::=
    typedef data_type type_identifier { variable_dimension } ;
  | typedef interface_instance_identifier . 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

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 } }
  | string
  | chandle
  | virtual [ interface ] interface_identifier
  | [ class_scope | package_scope ] type_identifier { packed_dimension }
  | class_type
  | event
  | ps_covergroup_identifier
  | type_reference[28]

data_type_or_implicit ::=
    data_type
  | [ signing ] { packed_dimension }

enum_base_type ::=
    integer_atom_type [ signing ]
  | integer_vector_type [ signing ] [ packed_dimension ]
  | type_identifier [ packed_dimension ][23]

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[33] ::=
[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[26] ::=
{ 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[27] **)**
| **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

A.2.3 Declaration lists

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_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

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

param_assignment ::= parameter_identifier { unpacked_dimension } = constant_param_expression

specparam_assignment ::=
specparam_identifier = constant_mintypmax_expression
| pulse_control_specparam

type_assignment ::=
type_identifier = data_type

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 [ ] [= dynamic_array_new]
| class_variable_identifier [= class_new]
| [covergroup_variable_identifier] = new [(list_of_arguments)][15]

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[11] ::=
[constant_range **]**
| unsized_dimension

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

variable_dimension ::=
unsized_dimension
| unpacked_dimension
| associative_dimension
| queue_dimension

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

unsized_dimension ::= **[ ]**

A.2.6 Function declarations

function_data_type ::= data_type | **void**

function_data_type_or_implicit ::=
function_data_type
| [signing] { packed_dimension }

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** function_data_type 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[8,9] ::= function_prototype

dpi_task_proto[9] ::= 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 }

tf_port_item[34] ::=
{ attribute_instance }
[tf_port_direction] [**var**] data_type_or_implicit
[port_identifier { variable_dimension } [**=** expression]]

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

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

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

A.2.8 Block item declarations

block_item_declaration ::=
{ attribute_instance } data_declaration
| { attribute_instance } local_parameter_declaration
| { attribute_instance } parameter_declaration **;**
| { attribute_instance } overload_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 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
concurrent_assertion_statement ::=
        assert_property_statement
      | assume_property_statement
      | cover_property_statement
assert_property_statement::=
        assert property ( property_spec ) action_block
assume_property_statement::=
        assume property ( property_spec ) ;
cover_property_statement::=
        cover property ( property_spec ) statement_or_null
expect_property_statement ::=
        expect ( property_spec ) action_block
property_instance ::=
        ps_property_identifier [ ( [ list_of_arguments ] ) ]
concurrent_assertion_item_declaration ::=
        property_declaration
      | sequence_declaration
property_declaration ::=
        property property_identifier [ ( [ tf_port_list ] ) ] ;
            { assertion_variable_declaration }
            property_spec ;
        endproperty [ : property_identifier ]
property_spec ::=
        [clocking_event ] [ disable iff ( expression_or_dist ) ] property_expr
property_expr ::=
        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
      | if ( expression_or_dist ) property_expr [ else property_expr ]
      | property_instance
      | clocking_event property_expr
```

```
sequence_declaration ::=
        sequence sequence_identifier [ ( [ tf_port_list ] ) ] ;
            { assertion_variable_declaration }
            sequence_expr ;
        endsequence [ : sequence_identifier ]
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 ]
      | ( expression_or_dist {, sequence_match_item } ) [ 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 ::=
        ## integral_number
      | ## identifier
      | ## ( constant_expression )
      | ## [ 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_sequence_identifier [ ( [ list_of_arguments ] ) ]
formal_list_item ::=
      formal_identifier [ = actual_arg_expr ]
list_of_formals ::= formal_list_item { , formal_list_item }
actual_arg_expr ::=
        event_expression
      | $
boolean_abbrev ::=
        consecutive_repetition
      | non_consecutive_repetition
      | goto_repetition
sequence_abbrev ::= consecutive_repetition
consecutive_repetition ::= [* const_or_range_expression ]
non_consecutive_repetition ::= [= const_or_range_expression ]
goto_repetition ::= [-> const_or_range_expression ]
const_or_range_expression ::=
        constant_expression
      | cycle_delay_const_range_expression
cycle_delay_const_range_expression ::=
```

```
        constant_expression : constant_expression
      | constant_expression : $
expression_or_dist ::= expression [ dist { dist_list } ]
assertion_variable_declaration ::=
        var_data_type list_of_variable_identifiers ;
```

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 = expression
coverage_spec ::=
        cover_point
      | cover_cross
coverage_event ::=
        clocking_event
      | @@( 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
cover_cross ::= [ cover_point_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 [ . bins_identifier ]
open_range_list ::= open_value_range { , open_value_range }
open_value_range ::= value_range[20]
```

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 Module, interface and generated instantiation

A.4.1 Instantiation

A.4.1.1 Module instantiation

```
module_instantiation ::=
        module_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 ] )
hierarchical_instance ::= name_of_instance ( [ list_of_port_connections ] )
name_of_instance ::= instance_identifier { unpacked_dimension }
list_of_port_connections[16] ::=
        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.4.2 Generated instantiation

```
module_or_interface_or_generate_item[31] ::=
        module_or_generate_item
      | interface_or_generate_item
generate_region ::=
        generate { module_or_interface_or_generate_item } endgenerate
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_or_null [ else generate_block_or_null ]
case_generate_construct ::=
        case ( constant_expression ) case_generate_item { case_generate_item } endcase
case_generate_item ::=
        constant_expression { , constant_expression } : generate_block_or_null
      | default [ : ] generate_block_or_null
generate_block ::=
        module_or_interface_or_generate_item
      | [ generate_block_identifier : ] begin [ : generate_block_identifier ]
            { module_or_interface_or_generate_item }
        end [ : generate_block_identifier ]
generate_block_or_null ::= generate_block | ;
```

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 reg** port_identifier [**=** constant_expression]

udp_input_declaration ::= { attribute_instance } **input** list_of_udp_port_identifiers

udp_reg_declaration ::= { attribute_instance } **reg** variable_identifier

A.5.3 UDP body

udp_body ::= combinational_body | sequential_body

combinational_body ::= **table** combinational_entry { combinational_entry } **endtable**

combinational_entry ::= level_input_list **:** output_symbol **;**

sequential_body ::= [udp_initial_statement] **table** sequential_entry { sequential_entry } **endtable**

udp_initial_statement ::= **initial** output_port_identifier **=** init_val **;**

init_val ::= **1'b0** | **1'b1** | **1'bx** | **1'bX** | **1'B0** | **1'B1** | **1'Bx** | **1'BX** | **1** | **0**

sequential_entry ::= seq_input_list **:** current_state **:** next_state **;**

seq_input_list ::= level_input_list | edge_input_list

level_input_list ::= level_symbol { level_symbol }

edge_input_list ::= { level_symbol } edge_indicator { level_symbol }

edge_indicator ::= (level_symbol level_symbol) | edge_symbol

current_state ::= level_symbol

next_state ::= output_symbol | **-**

output_symbol ::= **0** | **1** | **x** | **X**

level_symbol ::= **0** | **1** | **x** | **X** | **?** | **b** | **B**

edge_symbol ::= **r** | **R** | **f** | **F** | **p** | **P** | **n** | **N** | *****

A.5.4 UDP instantiation

udp_instantiation ::= udp_identifier [drive_strength] [delay2] udp_instance { **,** udp_instance } **;**

udp_instance ::= [name_of_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

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
| hierarchical_dynamic_array_variable_identifier **=** 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

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

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

```
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 )
      | @*
      | @ (*)
      | @ sequence_instance
event_expression ::=
        [ edge_identifier ] expression [ iff expression ]
      | sequence_instance [ iff expression ]
      | event_expression or event_expression
      | event_expression , event_expression
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

```
conditional_statement ::=
          if ( cond_predicate ) statement_or_null [ else statement_or_null ]
        | unique_priority_if_statement
unique_priority_if_statement ::=
          [ unique_priority ] if ( cond_predicate ) statement_or_null
              { else if ( cond_predicate ) statement_or_null }
              [ else statement_or_null ]
unique_priority ::= unique | 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

```
case_statement ::=
          [ unique_priority ] case_keyword ( expression ) case_item { case_item } endcase
        | [ unique_priority ] case_keyword ( expression ) matches case_pattern_item { case_pattern_item }
              endcase
        | [ unique_priority ] case ( expression ) inside case_inside_item { case_inside_item } endcase
case_keyword ::= case | casez | casex
case_item ::=
          expression { , 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
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

constant_assignment_pattern_expression[35] ::= assignment_pattern_expression

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

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

A.6.8 Looping statements

```
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 ( 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

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

A.6.10 Assertion statements

```
procedural_assertion_statement ::=
        concurrent_assertion_statement
      | immediate_assert_statement
immediate_assert_statement ::=
      assert ( expression ) action_block
```

A.6.11 Clocking block

```
clocking_declaration ::= [ default ] clocking [ clocking_identifier ] clocking_event ;
            { clocking_item }
        endclocking [ : clocking_identifier ]
clocking_event ::=
        @ identifier
      | @ ( event_expression )
clocking_item ::=
        default default_skew ;
      | clocking_direction list_of_clocking_decl_assign ;
      | { attribute_instance } concurrent_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 clockvar_expression <= expression
cycle_delay ::=
        ## integral_number
      | ## identifier
      | ## ( expression )
clockvar ::= hierarchical_identifier
clockvar_expression ::= clockvar select
```

A.6.12 Randsequence

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

production ::= [function_data_type] 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 (** expression **)** rs_case_item { rs_case_item } **endcase**

rs_case_item ::=
expression { **,** 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 =>
            ( specify_output_terminal_descriptor [ polarity_operator ] : data_source_expression ) )
full_edge_sensitive_path_description ::=
        ( [ edge_identifier ] list_of_path_inputs *>
            ( list_of_path_outputs [ polarity_operator ] : data_source_expression ) )
data_source_expression ::= expression
edge_identifier ::= posedge | negedge
state_dependent_path_declaration ::=
        if ( module_path_expression ) simple_path_declaration
      | if ( module_path_expression ) edge_sensitive_path_declaration
      | ifnone simple_path_declaration
polarity_operator ::= + | -
```

A.7.5 System timing checks

A.7.5.1 System timing check commands

```
system_timing_check ::=
        $setup_timing_check
      | $hold_timing_check
      | $setuphold_timing_check
      | $recovery_timing_check
      | $removal_timing_check
      | $recrem_timing_check
      | $skew_timing_check
      | $timeskew_timing_check
      | $fullskew_timing_check
      | $period_timing_check
      | $width_timing_check
      | $nochange_timing_check
$setup_timing_check ::=
        $setup ( data_event , reference_event , timing_check_limit [ , [ notifier ] ] ) ;
$hold_timing_check ::=
```

```
        $hold ( reference_event , data_event , timing_check_limit [ , [ notifier ] ] ) ;
$setuphold_timing_check ::=
        $setuphold ( reference_event , data_event , timing_check_limit , timing_check_limit
            [ , [ notifier ] [ , [ stamptime_condition ] [ , [ checktime_condition ]
            [ , [ delayed_reference ] [ , [ delayed_data ] ] ] ] ] ] ) ;
$recovery_timing_check ::=
        $recovery ( reference_event , data_event , timing_check_limit [ , [ notifier ] ] ) ;
$removal_timing_check ::=
        $removal ( reference_event , data_event , timing_check_limit [ , [ notifier ] ] ) ;
$recrem_timing_check ::=
        $recrem ( reference_event , data_event , timing_check_limit , timing_check_limit
            [ , [ notifier ] [ , [ stamptime_condition ] [ , [ checktime_condition ]
            [ , [ delayed_reference ] [ , [ delayed_data ] ] ] ] ] ] ) ;
$skew_timing_check ::=
        $skew ( reference_event , data_event , timing_check_limit [ , [ notifier ] ] ) ;
$timeskew_timing_check ::=
        $timeskew ( reference_event , data_event , timing_check_limit
            [ , [ notifier ] [ , [ event_based_flag ] [ , [ remain_active_flag ] ] ] ] ) ;
$fullskew_timing_check ::=
        $fullskew ( reference_event , data_event , timing_check_limit , timing_check_limit
            [ , [ notifier ] [ , [ event_based_flag ] [ , [ remain_active_flag ] ] ] ] ) ;
$period_timing_check ::=
        $period ( controlled_reference_event , timing_check_limit [ , [ notifier ] ] ) ;
$width_timing_check ::=
        $width ( controlled_reference_event , timing_check_limit , threshold [ , [ notifier ] ] ) ;
$nochange_timing_check ::=
        $nochange ( reference_event , data_event , start_edge_offset ,
            end_edge_offset [ , [ notifier ] ] ) ;
```

A.7.5.2 System timing check command arguments

```
checktime_condition ::= mintypmax_expression
controlled_reference_event ::= controlled_timing_check_event
data_event ::= timing_check_event
delayed_data ::=
        terminal_identifier
      | terminal_identifier [ constant_mintypmax_expression ]
delayed_reference ::=
        terminal_identifier
      | terminal_identifier [ constant_mintypmax_expression ]
end_edge_offset ::= mintypmax_expression
event_based_flag ::= constant_expression
notifier ::= variable_identifier
reference_event ::= timing_check_event
remain_active_flag ::= constant_mintypmax_expression
stamptime_condition ::= mintypmax_expression
start_edge_offset ::= mintypmax_expression
threshold ::=constant_expression
```

timing_check_limit ::= expression

A.7.5.3 System timing check event definitions

timing_check_event ::=
[timing_check_event_control] specify_terminal_descriptor [**&&&** timing_check_condition]
controlled_timing_check_event ::=
timing_check_event_control specify_terminal_descriptor [**&&&** timing_check_condition]
timing_check_event_control ::=
posedge
| **negedge**
| edge_control_specifier
specify_terminal_descriptor ::=
specify_input_terminal_descriptor
| specify_output_terminal_descriptor
edge_control_specifier ::= **edge [** edge_descriptor { **,** edge_descriptor } **]**
edge_descriptor[1] ::= **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

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 **}**[18]
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[21] ::= **{ }**

A.8.2 Subroutine calls

constant_function_call ::= function_subroutine_call[24]

tf_call[36] ::= 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
  | 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** constraint_block]

method_call_root ::= expression | implicit_class_handle

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

A.8.3 Expressions

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

part_select_range ::= constant_range | indexed_range

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

genvar_expression ::= constant_expression

A.8.4 Primaries

constant_primary ::=
primary_literal
| ps_parameter_identifier constant_select
| specparam_identifier [constant_range_expression]
| genvar_identifier[32]
| [package_scope | class_scope] enum_identifier
| constant_concatenation
| constant_multiple_concatenation
| constant_function_call
| (constant_mintypmax_expression)
| constant_cast
| constant_assignment_pattern_expression
| type_reference[29]

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

primary ::=
primary_literal
| [implicit_class_handle **.** | class_scope | package_scope] hierarchical_identifier select
| empty_queue
| concatenation
| multiple_concatenation
| function_subroutine_call
| **(** mintypmax_expression **)**
| cast
| assignment_pattern_expression
| streaming_concatenation
| sequence_method_call
| **this**[6]
| **$**[22]
| **null**

time_literal[5] ::=
unsigned_number time_unit
| fixed_point_number time_unit

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

implicit_class_handle[6] ::= **this** | **super** | **this . super**

bit_select ::= { **[** expression **]** }

```
select ::=
        [ { . member_identifier  bit_select } . member_identifier ] bit_select [ [ part_select_range ] ]
constant_bit_select ::= { [ constant_expression ] }
constant_select ::=
        [ { . member_identifier  constant_bit_select } . member_identifier ] constant_bit_select
            [ [ constant_part_select_range ] ]
primary_literal ::= number | time_literal | unbased_unsized_literal | string_literal
constant_cast ::=
        casting_type ' ( constant_expression )
cast ::=
        casting_type ' ( expression )
```

A.8.5 Expression left-side values

```
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
      | { variable_lvalue { , variable_lvalue } }
      | [ assignment_pattern_expression_type ] assignment_pattern_variable_lvalue
      | streaming_concatenation[30]
```

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

real_number[1] ::=
fixed_point_number
| unsigned_number [. unsigned_number] exp [sign] unsigned_number

fixed_point_number[1] ::= unsigned_number . unsigned_number

exp ::= e | E

unsigned_number[1] ::= decimal_digit { _ | decimal_digit }

binary_value[1] ::= binary_digit { _ | binary_digit }

octal_value[1] ::= octal_digit { _ | octal_digit }

hex_value[1] ::= hex_digit { _ | hex_digit }

decimal_base[1] ::= '[s|S]d | '[s|S]D

binary_base[1] ::= '[s|S]b | '[s|S]B

octal_base[1] ::= '[s|S]o | '[s|S]O

hex_base[1] ::= '[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 [10]

A.8.8 Strings

string_literal ::= " { Any_ASCII_Characters } "

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

array_identifier ::= identifier

block_identifier ::= identifier

bin_identifier ::= identifier

c_identifier[2] ::= [**a-zA-Z_**] { [**a-zA-Z0-9_**] }

cell_identifier ::= identifier

class_identifier ::= identifier

class_variable_identifier ::= variable_identifier

clocking_identifier ::= identifier

config_identifier ::= identifier

const_identifier ::= identifier

constraint_identifier ::= identifier

covergroup_identifier ::= identifier

covergroup_variable_identifier ::= variable_identifier

cover_point_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_block_identifier ::= hierarchical_identifier

hierarchical_dynamic_array_variable_identifier ::= hierarchical_variable_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_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

net_identifier ::= identifier

output_port_identifier ::= identifier

package_identifier ::= identifier

package_scope ::=
package_identifier **::**
| **$unit ::**

parameter_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_net_identifier ::= [package_scope] net_identifier | hierarchical_net_identifier

ps_or_hierarchical_tf_identifier ::= [package_scope] tf_identifier | hierarchical_tf_identifier

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

ps_property_identifier ::= [package_scope] property_identifier

ps_sequence_identifier ::= [package_scope] sequence_identifier

ps_type_identifier ::= [package_scope] type_identifier

sequence_identifier ::= identifier

signal_identifier ::= identifier

simple_identifier[2] ::= [**a-zA-Z_**] { [**a-zA-Z0-9_$**] }

specparam_identifier ::= identifier

system_tf_identifier[3] ::= **$**[**a-zA-Z0-9_$**]{ [**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[4]

A.10 Footnotes (normative)

1) Embedded spaces are illegal.
2) A simple_identifier, c_identifier, and arrayed_reference shall start with an alpha or underscore (_) character, shall have at least one character, and shall not have any spaces.
3) The $ character in a system_tf_identifier shall not be followed by white_space. A system_tf_identifier shall not be escaped.
4) End of file.
5) The unsigned number or fixed-point number in time_literal shall not be followed by a white_space.
6) implicit_class_handle shall only appear within the scope of a class_declaration or out-of-block method declaration.
7) 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.
8) dpi_function_proto return types are restricted to small values, per 26.4.5.
9) Formals of dpi_function_proto and dpi_task_proto cannot use pass-by-reference mode, and class types cannot be passed at all; for the complete set of restrictions, see 26.4.6.
10) The apostrophe (') in unbased_unsized_literal shall not be followed by white_space.
11) In packed_dimension, unsized_dimension is permitted only in declarations of import DPI functions; see dpi_function_proto.
12) When a packed dimension is used with the **struct** or **union** keyword, the **packed** keyword shall also be used.
13) 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.
14) In a data_declaration that is not within the 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.
15) It shall be legal to omit the covergroup_variable_identifier from a covergroup instantiation only if this implicit instantiation is within a class that has no other instantiation of the covergroup.
16) The **.*** token shall appear at most once in a list of port connections.
17) A timeunits_declaration shall be legal as a non_port_module_item, non_port_interface_item, non_port_program_item, package_item, or class_item only if it repeats and matches a previous timeunits_declaration within the same time scope.
18) 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.
19) In a shallow copy, the expression must evaluate to an object handle.
20) It shall be legal to use the **$** primary in an open_value_range of the form **[** expression **: $]** or **[$:** expression **]**.
21) **{ }** shall only be legal in the context of a queue.

22) The **$** primary shall be legal only in a select for a queue variable or in an open_value_range.

23) 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.

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

25) The list_of_port_declarations syntax is explained in 19.8, which also imposes various semantic restrictions, e.g., a **ref** port must be of a variable type and an **inout** port must not be. It shall be illegal to initialize a port that is not a variable **output** port.

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

27) 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.

28) 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.

29) 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.

30) 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.

31) Within an interface_declaration, it shall only be legal for a module_or_interface_or_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 module_or_interface_or_generate_item to be a module_or_generate_item.

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

33) 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.

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

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

36) 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.

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

Annex B

(normative)

Keywords

SystemVerilog reserves the keywords listed in Table B-1.

Legend:

— * indicates SystemVerilog reserved words that are not reserved in IEEE Std 1364.

Table B-1—Reserved keywords

alias*	endmodule	matches*	small
always	endpackage*	medium	solve*
always_comb*	endprimitive	modport*	specify
always_ff*	endprogram*	module	specparam
always_latch*	endproperty*	nand	static*
and	endspecify	negedge	string*
assert*	endsequence*	new*	strong0
assign	endtable	nmos	strong1
assume*	endtask	nor	struct*
automatic	enum*	noshowcancelled	super*
before*	event	not	supply0
begin	expect*	notif0	supply1
bind*	export*	notif1	table
bins*	extends*	null*	tagged*
binsof*	extern*	or	task
bit*	final*	output	this*
break*	first_match*	package*	throughout*
buf	for	packed*	time
bufif0	force	parameter	timeprecision*
bufif1	foreach*	pmos	timeunit*
byte*	forever	posedge	tran
case	fork	primitive	tranif0
casex	forkjoin*	priority*	tranif1
casez	function	program*	tri
cell	generate	property*	tri0
chandle*	genvar	protected*	tri1
class*	highz0	pull0	triand
clocking*	highz1	pull1	trior
cmos	if	pulldown	trireg
config	iff*	pullup	type*
const*	ifnone	pulsestyle_onevent	typedef*
constraint*	ignore_bins*	pulsestyle_ondetect	union*
context*	illegal_bins*	pure*	unique*
continue*	import*	rand*	unsigned
cover*	incdir	randc*	use
covergroup*	include	randcase*	uwire
coverpoint*	initial	randsequence*	var*
cross*	inout	rcmos	vectored
deassign	input	real	virtual*
default	inside*	realtime	void*
defparam	instance	ref*	wait
design	int*	reg	wait_order*
disable	integer	release	wand
dist*	interface*	repeat	weak0
do*	intersect*	return*	weak1
edge	join	rnmos	while
else	join_any*	rpmos	wildcard*
end	join_none*	rtran	wire
endcase	large	rtranif0	with*
endclass*	liblist	rtranif1	within*
endclocking*	library	scalared	wor
endconfig	local*	sequence*	xnor
endfunction	localparam	shortint*	xor
endgenerate	logic*	shortreal*	
endgroup*	longint*	showcancelled	
endinterface*	macromodule	signed	

Annex C

(normative)

Std package

C.1 General

The standard package contains system types (see 8.10.1). The following types are provided by the std built-in package. The descriptions of the semantics of these types are defined in the indicated subclauses.

C.2 Semaphore

The semaphore class is described in 14.2, and its prototype is as follows:

```
class semaphore;
   function new(int keyCount = 0);
   task put(int keyCount = 1);
   task get(int keyCount = 1);
   function int try_get(int keyCount = 1);
endclass
```

C.3 Mailbox

The mailbox class is described in 14.3, and its prototype is as follows:

The *dynamic_singular_type* below represents a special type that enables run-time type checking.

```
class mailbox #(type T = dynamic_singular_type) ;
   function new(int bound = 0);
   function int num();
   task put( T message);
   function int try_put( T message);
   task get( ref T message );
   function int try_get( ref T message );
   task peek( ref T message );
   function int try_peek( ref T message );
endclass
```

C.4 Randomize

The randomize function is described in 13.11, and its prototype is as follows:

```
function int randomize( ... );
```

The syntax for the randomize function is as follows:

```
randomize( variable_identifier {, variable_identifier } )
   [ with constraint_block ];
```

C.5 Process

The process class is described in 11.9, and its prototype is as follows:

```
class process;
   enum state { FINISHED, RUNNING, WAITING, SUSPENDED, KILLED };

   static function process self();
   function state status();
   task kill();
   task await();
   task suspend();
   task resume();
endclass
```