**// 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

{ 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

module\_keyword ::= module | macromodule

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 ] ;

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 ] ) ] ;

{ { attribute\_instance } checker\_or\_generate\_item } endchecker [ : checker\_identifier ]

class\_declaration ::=

[ virtual ] class [ lifetime ] class\_identifier [ parameter\_port\_list ]

[ extends class\_type [ ( list\_of\_arguments ) ] ]

[ implements interface\_class\_type { , interface\_class\_type } ] ; { class\_item }

endclass [ : class\_identifier]

interface\_class\_type ::= ps\_class\_identifier [ parameter\_value\_assignment ]

interface\_class\_declaration ::=

interface class class\_identifier [ parameter\_port\_list ]

[ extends interface\_class\_type { , interface\_class\_type } ] ;

{ interface\_class\_item } endclass [ : class\_identifier]

interface\_class\_item ::= type\_declaration

| { attribute\_instance } interface\_class\_method

| local\_parameter\_declaration ;

| parameter\_declaration7 ; |;

interface\_class\_method ::=

pure virtual method\_prototype ;

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\_declarations2 ::=

( [ { 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

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

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 ;

| default disable iff expression\_or\_dist ;

non\_port\_module\_item ::= generate\_region

| module\_or\_generate\_item

| specify\_block

| { attribute\_instance } specparam\_declaration

| program\_declaration

| module\_declaration

| interface\_declaration

| timeunits\_declaration3

parameter\_override ::= defparam list\_of\_defparam\_assignments ;

bind\_directive4 ::=

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 } 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

| modport\_declaration

| interface\_declaration

| timeunits\_declaration3  **// 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

| timeunits\_declaration3

| program\_generate\_item

program\_generate\_item5 ::= 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 } [ checker\_port\_direction ] property\_formal\_type formal\_port\_identifier

{variable\_dimension} [ = property\_actual\_arg ] checker\_port\_direction ::=

input | output

checker\_or\_generate\_item ::= checker\_or\_generate\_item\_declaration

| initial\_construct

| always\_construct

| final\_construct

| assertion\_item

| continuous\_assign

| checker\_generate\_item

checker\_or\_generate\_item\_declaration ::= [ rand ] data\_declaration

| function\_declaration

| checker\_declaration

| assertion\_item\_declaration

| covergroup\_declaration

| genvar\_declaration

| clocking\_declaration

| default clocking clocking\_identifier ;

| default disable iff expression\_or\_dist ; |;

checker\_generate\_item6 ::= loop\_generate\_construct

| conditional\_generate\_construct

| generate\_region

| elaboration\_system\_task

**// 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\_declaration7 ; |;

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

| pure virtual { class\_item\_qualifier } method\_prototype ;

| 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\_qualifier8 ::= static

| protected | local

property\_qualifier8 ::= random\_qualifier

| class\_item\_qualifier

random\_qualifier8 ::= rand

| randc

method\_qualifier8 ::=

[ 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 ::= constraint\_primary { , constraint\_primary }

constraint\_primary ::= [ implicit\_class\_handle . | class\_scope ] hierarchical\_identifier select

constraint\_expression ::=

[ soft ] expression\_or\_dist ;

| uniqueness\_constraint ;

| expression –> constraint\_set

| if ( expression ) constraint\_set [ else constraint\_set ]

| foreach ( ps\_or\_hierarchical\_array\_identifier [ loop\_variables ] ) constraint\_set | disable soft constraint\_primary ;

uniqueness\_constraint ::=

unique { open\_range\_list9 }

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\_declaration3

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

| 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\_variable\_identifiers

**// A.2.1.3 Type declarations**

data\_declaration ::=

[ const ] [ var ] [ lifetime ] data\_type\_or\_implicit list\_of\_variable\_decl\_assignments ;10

| type\_declaration

| package\_import\_declaration11

| net\_type\_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 ;

net\_declaration12 ::=

net\_type [ drive\_strength | charge\_strength ] [ vectored | scalared ]

data\_type\_or\_implicit [ delay3 ] list\_of\_net\_decl\_assignments ;

| net\_type\_identifier [ delay\_control ]

list\_of\_net\_decl\_assignments ;

| interconnect implicit\_data\_type [ # delay\_value ]

net\_identifier { unpacked\_dimension }

[ , net\_identifier { unpacked\_dimension }] ;

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 | interface class ] type\_identifier ;

net\_type\_declaration ::=

nettype data\_type net\_type\_identifier

[ with [ package\_scope | class\_scope ] tf\_identifier ] ;

| nettype [ package\_scope | class\_scope ] net\_type\_identifier net\_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 }13

| enum [ enum\_base\_type ] { enum\_name\_declaration { , enum\_name\_declaration } }

{ packed\_dimension }

| string

| chandle

| virtual [ interface ] interface\_identifier [ parameter\_value\_assignment ] [ . modport\_identifier ]

| [ class\_scope | package\_scope ] type\_identifier { packed\_dimension }

| class\_type

| event

| ps\_covergroup\_identifier

| type\_reference14

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 ]15 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 ::=

[ net\_type ] data\_type\_or\_implicit

| net\_type\_identifier

| interconnect implicit\_data\_type

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\_member16 ::=

{ 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 ( expression17 ) | 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**

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 ] }

**// 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 ]18

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\_new19 ::=

[ class\_scope ] new [ ( list\_of\_arguments ) ]

| new expression

dynamic\_array\_new::=new [expression][(expression)]

**// A.2.5 Declaration ranges**

unpacked\_dimension ::= [ constant\_range ]

| [ constant\_expression ]

packed\_dimension20 ::= [ 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\_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\_proto21,22 ::= function\_prototype dpi\_task\_proto22 ::= 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\_item23 ::=

{ 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 } let\_declaration

**// A.2.9 Interface declarations**

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\_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

formal\_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\_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

| if ( expression\_or\_dist ) property\_expr [ else property\_expr ]

| case ( expression\_or\_dist ) property\_case\_item { property\_case\_item } endcase | 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

property\_case\_item ::=

expression\_or\_dist { , expression\_or\_dist } : property\_expr ;

| default [ : ] 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

formal\_port\_identifier {variable\_dimension} [ = sequence\_actual\_arg ] sequence\_lvar\_port\_direction ::= input | inout | output

sequence\_formal\_type ::= data\_type\_or\_implicit

| sequence | 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 ;

**// 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 ::=

[ [ data\_type\_or\_implicit ] 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 [ [ [ covergroup\_expression ] ] ] = { covergroup\_range\_list } [ with ( with\_covergroup\_expression ) ]

[ iff ( expression ) ]

| [ wildcard ] bins\_keyword bin\_identifier [ [ [ covergroup\_expression ] ] ] = cover\_point\_identifier with ( with\_covergroup\_expression ) [ iff ( expression ) ]

| [ wildcard ] bins\_keyword bin\_identifier [ [ [ covergroup\_expression ] ] ] = set\_covergroup\_expression [ iff ( expression ) ]

| [ wildcard] bins\_keyword bin\_identifier [ [ ] ] = trans\_list [ iff ( expression ) ]

| bins\_keyword bin\_identifier [ [ [ covergroup\_expression ] ] ] = default [ iff ( expression ) ] | bins\_keyword bin\_identifier = default sequence [ iff ( expression ) ]

bins\_keyword::= bins | illegal\_bins | ignore\_bins 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 ::= covergroup\_range\_list

repeat\_range ::= covergroup\_expression

| covergroup\_expression : covergroup\_expression cover\_cross ::=

[cross\_identifier: ]crosslist\_of\_cross\_items[iff(expression)]cross\_body

list\_of\_cross\_items ::= cross\_item , cross\_item { , cross\_item }

cross\_item ::= cover\_point\_identifier

| variable\_identifier

cross\_body ::=

{ { cross\_body\_item ; } }

|;

cross\_body\_item ::= function\_declaraton

| 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\_expression24 ::= select\_condition

| ! select\_condition

| select\_expression && select\_expression

| select\_expression || select\_expression

| ( select\_expression )

| select\_expression with ( with\_covergroup\_expression ) [ matches integer\_covergroup\_expression ]

| cross\_identifier

| cross\_set\_expression [ matches integer\_covergroup\_expression ]

select\_condition::=binsof (bins\_expression)[intersect{covergroup\_range\_list}]

bins\_expression ::= variable\_identifier

| cover\_point\_identifier [ . bin\_identifier ]

covergroup\_range\_list ::= covergroup\_value\_range { , covergroup\_value\_range }

covergroup\_value\_range ::= covergroup\_expression

| [ covergroup\_expression : covergroup\_expression ]25 with\_covergroup\_expression ::= covergroup\_expression26

set\_covergroup\_expression ::= covergroup\_expression27 integer\_covergroup\_expression ::= covergroup\_expression cross\_set\_expression ::= covergroup\_expression

covergroup\_expression ::= expression28  **// A.2.12 Let declarations**

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 formal\_port\_identifier { variable\_dimension } [ = expression ]

let\_formal\_type ::= data\_type\_or\_implicit

| untyped let\_expression ::=

[ package\_scope ] let\_identifier [ ( [ let\_list\_of\_arguments ] ) ]

let\_list\_of\_arguments ::=

[ let\_actual\_arg ] {, [ let\_actual\_arg ] } {, . identifier ( [ let\_actual\_arg ] ) }

| . identifier ( [ let\_actual\_arg ] ) { , . identifier ( [ let\_actual\_arg ] ) } let\_actual\_arg ::=

expression

**// A.3 Primitive instances**

**// A.3.1 Primitive instantiation and instances**

gate\_instantiation ::=

cmos\_switchtype [delay3] cmos\_switch\_instance { , cmos\_switch\_instance } ;

| enable\_gatetype [drive\_strength] [delay3] enable\_gate\_instance { , enable\_gate\_instance } ;

| mos\_switchtype [delay3] mos\_switch\_instance { , mos\_switch\_instance } ;

| n\_input\_gatetype [drive\_strength] [delay2] n\_input\_gate\_instance { , n\_input\_gate\_instance } ;

| n\_output\_gatetype [drive\_strength] [delay2] n\_output\_gate\_instance

{ , n\_output\_gate\_instance } ;

| pass\_en\_switchtype [delay2] pass\_enable\_switch\_instance { , pass\_enable\_switch\_instance } ;

| pass\_switchtype pass\_switch\_instance { , pass\_switch\_instance } ;

| pulldown [pulldown\_strength] pull\_gate\_instance { , pull\_gate\_instance } ;

| pullup [pullup\_strength] pull\_gate\_instance { , pull\_gate\_instance } ;

cmos\_switch\_instance ::= [ name\_of\_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\_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\_connections29 ::=

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 ::=

ps\_checker\_identifier name\_of\_instance ( [list\_of\_checker\_port\_connections] ) ;

list\_of\_checker\_port\_connections29 ::=

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 } . formal\_port\_identifier [ ( [ property\_actual\_arg ] ) ]

| { attribute\_instance } .\*

**// A.4.2 Generated instantiation**

generate\_region ::=

generate { 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 [ 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\_item30 ::= 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}primitiveudp\_identifier(udp\_port\_list) ;

udp\_ansi\_declaration ::= {attribute\_instance}primitiveudp\_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

| 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**

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 )

| @\*

| @ (\*)

| @ ps\_or\_hierarchical\_sequence\_identifier

event\_expression31 ::=

[ edge\_identifier ] expression [ iff expression ]

| sequence\_instance [ iff expression ]

| event\_expression or event\_expression

| 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 ::=

[ 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**

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 open\_range\_list ::= open\_value\_range { , open\_value\_range }

open\_value\_range ::= value\_range25  **// 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\_expression32 ::= 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 ( ps\_or\_hierarchical\_array\_identifier [ loop\_variables ] ) statement

for\_initialization ::= list\_of\_variable\_assignments

| for\_variable\_declaration { , for\_variable\_declaration } for\_variable\_declaration ::=

[ var ] data\_type variable\_identifier = expression { , variable\_identifier = expression }14

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**

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

| assert final ( expression ) action\_block

deferred\_immediate\_assume\_statement ::= assume #0 ( expression ) action\_block

| assume final ( expression ) action\_block

deferred\_immediate\_cover\_statement ::=

cover #0 ( expression ) statement\_or\_null

| cover final ( expression ) statement\_or\_null

**// A.6.11 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.6.12 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[elseproduction\_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\_descriptor33 ::= 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 }34

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\_unpacked\_array\_concatenation35 ::= { }

**// A.8.2 Subroutine calls**

constant\_function\_call ::= function\_subroutine\_call36

tf\_call37 ::= 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 ] )

| system\_tf\_identifier ( expression { , [ expression ] } [ , [ clocking\_event ] ] )

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 ]38

method\_call\_root ::= primary | 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\_identifier39

| formal\_port\_identifier constant\_select

| [ package\_scope | class\_scope ] enum\_identifier

| constant\_concatenation [ [ constant\_range\_expression ] ]

| constant\_multiple\_concatenation [ [ constant\_range\_expression ] ] | constant\_function\_call

| constant\_let\_expression

| ( constant\_mintypmax\_expression )

| constant\_cast

| constant\_assignment\_pattern\_expression

| type\_reference40 | null

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\_unpacked\_array\_concatenation

| concatenation [ [ range\_expression ] ]

| multiple\_concatenation [ [ range\_expression ] ]

| function\_subroutine\_call

| let\_expression

| ( mintypmax\_expression )

| cast

| assignment\_pattern\_expression

| streaming\_concatenation

| sequence\_method\_call

| this41

| $42

| null

class\_qualifier := [ local ::43 ] [ implicit\_class\_handle . | class\_scope ]

range\_expression ::= expression

| part\_select\_range

primary\_literal ::= number | time\_literal | unbased\_unsized\_literal | string\_literal

time\_literal44 ::= unsigned\_number time\_unit

| fixed\_point\_number time\_unit time\_unit ::= s | ms | us | ns | ps | fs

implicit\_class\_handle41 ::= 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\_expression45

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\_identifierselect46

| { variable\_lvalue { , variable\_lvalue } }

| [ assignment\_pattern\_expression\_type ] assignment\_pattern\_variable\_lvalue

| streaming\_concatenation47 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\_number33 ::= non\_zero\_decimal\_digit { \_ | decimal\_digit}

real\_number33 ::= fixed\_point\_number

| unsigned\_number [ . unsigned\_number ] exp [ sign ] unsigned\_number fixed\_point\_number33 ::= unsigned\_number . unsigned\_number

exp ::= e | E

unsigned\_number33 ::= decimal\_digit { \_ | decimal\_digit } binary\_value33 ::= binary\_digit { \_ | binary\_digit } octal\_value33 ::= octal\_digit { \_ | octal\_digit } hex\_value33 ::= hex\_digit { \_ | hex\_digit }

decimal\_base33 ::= '[s|S]d | '[s|S]D

binary\_base33 ::= '[s|S]b | '[s|S]B

octal\_base33 ::= '[s|S]o | '[s|S]O

hex\_base33 ::= '[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 48

**// 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\_identifier49 ::= [ 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

const\_identifier ::= identifier

constraint\_identifier ::= identifier covergroup\_identifier ::= identifier covergroup\_variable\_identifier ::= variable\_identifier cover\_point\_identifier ::= identifier

cross\_identifier ::= identifier

dynamic\_array\_variable\_identifier ::= variable\_identifier

enum\_identifier ::= identifier

escaped\_identifier ::= \ {any\_printable\_ASCII\_character\_except\_white\_space} white\_space formal\_identifier ::= identifier

formal\_port\_identifier ::= identifier

function\_identifier ::= identifier

generate\_block\_identifier ::= identifier

genvar\_identifier ::= identifier

hierarchical\_array\_identifier ::= hierarchical\_identifier

hierarchical\_block\_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\_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 net\_identifier ::= identifier net\_type\_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\_checker\_identifier ::= [ package\_scope ] checker\_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 ::43 | package\_scope | class\_scope ] type\_identifier sequence\_identifier ::= identifier

signal\_identifier ::= identifier

simple\_identifier49 ::= [ a-zA-Z\_ ] { [ a-zA-Z0-9\_$ ] } specparam\_identifier ::= identifier

system\_tf\_identifier50 ::= $[ a-zA-Z0-9\_$ ]{ [ 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

**// 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 23.2.2.2, 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) The *open\_range\_list* in a *uniqueness\_constraint* shall contain only expressions that denote scalar or array variables, as described in 18.5.5.

10) 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.

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

12) 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.

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

14) When a type\_reference is used in a net declaration, it shall be preceded by a net type keyword; and when it is used

packed dimension is not permitted, or an integer\_vector\_type.

16) It shall be legal to declare a void struct\_union\_member only within tagged unions. It shall be legal to declare a random\_qualifier only within unpacked structures.

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) It shall be legal to omit the constant\_param\_expression from a param\_assignment or the data\_type from a type\_assignment only within a parameter\_port\_list. However, it shall not be legal to omit them from localparam declarations 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 35.5.5.

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 35.5.6 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) The matches operator shall have higher precedence than the && and || operators.

25) It shall be legal to use the $ primary in an open\_value\_range or covergroup\_value\_range of the form [ expression : $ ] or [ $ : expression ].

26) The result of this expression shall be assignment compatible with an integral type.

27) This expression is restricted as described in 19.5.1.2.

28) This expression is restricted as described in 19.5.

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

30) 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.

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

32) In a constant\_assignment\_pattern\_expression, all member expressions shall be constant expressions.

33) Embedded spaces are illegal.

34) 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.

in a variable declaration, it shall be preceded by the var keyword.

15) A type\_identifier shall be legal as an enum\_base\_type if it denotes an integer\_atom\_type, with which an additional

35) { } shall denote an empty unpacked array concatenation, as described in 10.10, and shall not be used in any other form of concatenation.

36) In a constant\_function\_call, all arguments shall be constant\_expressions.

37) 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.

38) 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.

39) A genvar\_identifier shall be legal in a constant\_primary only within a genvar\_expression.

40) 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.

41) implicit\_class\_handle shall only appear within the scope of a class\_declaration or out-of-block method declaration.

42) The $ primary shall be legal only in a select for a queue variable, in an open\_value\_range, covergroup\_val- ue\_range, integer\_covergroup\_expression, or as an entire sequence\_actual\_arg or property\_actual\_arg.

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

44) The unsigned number or fixed-point number in time\_literal shall not be followed by white\_space.

45) 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.

46) In a variable\_lvalue that is assigned within a sequence\_match\_item any select shall also be a constant\_select.

47) 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.

48) The apostrophe ( ' ) in unbased\_unsized\_literal shall not be followed by white\_space.

49) 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.

50) The $ character in a system\_tf\_identifier shall not be followed by white\_space. A system\_tf\_identifier shall not be escaped.

End of file.