

Annex A

(informative)

Syntax summary

This annex summarizes, using Backus-Naur Form (BNF), the syntax that is supported for RTL synthesis.

A.1 Source text

A.1.1 Library source text

```
library_text ::= { library_descriptions }
```

```
library_descriptions ::=  
    library_declaration  
    | include_statement  
    | config_declaration
```

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

```
file_path_spec ::= file_path
```

```
include_statement ::= include < file_path_spec >;
```

A.1.2 Configuration source text

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

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

```
config_rule_statement ::=  
    default_clause liblist_clause  
    | inst_clause liblist_clause  
    | inst_clause use_clause  
    | cell_clause liblist_clause  
    | cell_clause use_clause
```

```
default_clause ::= default
```

```
inst_clause ::= instance inst_name
```

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

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

```
liblist_clause ::= liblist [ { library_identifier } ]
```

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

A.1.3 Module and primitive source text

source_text ::= { description }

description ::=

module_declaration

+ ~~udp_declaration~~

module_declaration ::=

{ [attribute_instance](#) } module_keyword module_identifier [module_parameter_port_list]
[list_of_ports] ; { module_item }

endmodule

| { [attribute_instance](#) } module_keyword module_identifier [module_parameter_port_list]
[list_of_port_declarations] ; { non_port_module_item }

endmodule

module_keyword ::= **module** | **macromodule**

A.1.4 Module parameters and ports

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

list_of_ports ::= (port { , port })

list_of_port_declarations ::=

(port_declaration { , port_declaration })

| 0

port ::=

[port_expression]

| . port_identifier ([port_expression])

port_expression ::=

port_reference

| { port_reference { , port_reference } }

port_reference ::= port_identifier

port_identifier [constant_expression]

| port_identifier [range_expression]

port_declaration ::=

{ [attribute_instance](#) } inout_declaration

| { [attribute_instance](#) } input_declaration

| { [attribute_instance](#) } output_declaration

A.1.5 Module items

module_item ::=

module_or_generate_item

| port_declaration ;

| { [attribute_instance](#) } generated_instantiation

| { [attribute_instance](#) } local_parameter_declaration

| { [attribute_instance](#) } parameter_declaration

| { [attribute_instance](#) } [specify_block](#)
| { [attribute_instance](#) } [specparam_declaration](#)

module_or_generate_item ::=
 { [attribute_instance](#) } module_or_generate_item_declaration
| { [attribute_instance](#) } parameter_override
| { [attribute_instance](#) } continuous_assign
| { [attribute_instance](#) } gate_instantiation
| { [attribute_instance](#) } **udp_instantiation**
| { [attribute_instance](#) } module_instantiation
| { [attribute_instance](#) } [initial_construct](#)
| { [attribute_instance](#) } [always_construct](#)

module_or_generate_item_declaration ::=
 net_declaration
| reg_declaration
| integer_declaration
~~+real_declaration~~
~~+time_declaration~~
~~+realtime_declaration~~
~~+event_declaration~~
| genvar_declaration
| task_declaration
| function_declaration

non_port_module_item ::=
 { [attribute_instance](#) } generated_instantiation
| { [attribute_instance](#) } local_parameter_declaration
| { [attribute_instance](#) } module_or_generate_item
| { [attribute_instance](#) } parameter_declaration
| { [attribute_instance](#) } [specify_block](#)
| { [attribute_instance](#) } [specparam_declaration](#)

parameter_override ::= **defparam** list_of_param_assignments ;

A.2 Declarations

A.2.1 Declaration types

A.2.1.1 Module parameter declarations

local_parameter_declaration ::=
 localparam [**signed**] [range] list_of_param_assignments ;
| **localparam integer** list_of_param_assignments ;
~~+localparam real list_of_param_assignments ;~~
~~+localparam realtime list_of_param_assignments ;~~
~~+localparam time list_of_param_assignments ;~~

parameter_declaration ::=
 parameter [**signed**] [range] list_of_param_assignments ;
| **parameter integer** list_of_param_assignments ;
~~+parameter real list_of_param_assignments ;~~
~~+parameter realtime list_of_param_assignments ;~~
~~+parameter time list_of_param_assignments ;~~

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

A.2.1.2 Port declarations

inout_declaration ::=
inout [net_type] [**signed**] [range] list_of_port_identifiers

input_declaration ::=
input [net_type] [signed] [range] list_of_port_identifiers

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

A.2.1.3 Type declarations

event_declaration ::= **event** list_of_event_identifiers ;

genvar_declaration ::= **genvar** list_of_genvar_identifiers ;

integer_declaration ::= **integer** list_of_variable_identifiers ;

net_declaration ::=
net_type [**signed**] [[delay3](#)] list_of_net_identifiers ;
| net_type [[drive_strength](#)] [**signed**] [[delay3](#)]
list_of_net_decl_assignments ;
| net_type [**vectored** | **scalared**] [**signed**] range [[delay3](#)]
list_of_net_identifiers ;
| net_type [[drive_strength](#)] [**vectored** | **scalared**] [**signed**] range
[[delay3](#)] list_of_net_decl_assignments ;
~~| **tri**reg [charge_strength] [**signed**] [[delay3](#)] list_of_net_identifiers ;~~
~~| **tri**reg [drive_strength] [**signed**] [[delay3](#)]
list_of_net_decl_assignments ;~~
~~| **tri**reg [charge_strength] [**vectored** | **scalared**] [**signed**] range
[[delay3](#)] list_of_net_identifiers ;~~
~~| **tri**reg [drive_strength] [**vectored** | **scalared**] [**signed**] range
[[delay3](#)] list_of_net_decl_assignments ;~~

real_declaration ::= **real** list_of_real_identifiers ;

realtime_declaration ::= **realtime** list_of_real_identifiers ;

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

time_declaration ::= **time** list_of_variable_identifiers ;

A.2.2 Declaration data types

A.2.2.1 Net and variable types

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

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

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

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

A.2.2.2 Strengths

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

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

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

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

A.2.2.3 Delays

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

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

delay_value ::=
unsigned_number
| parameter_identifier
| specparam_identifier
| mintypmax_expression

A.2.3 Declaration lists

list_of_event_identifiers ::= event_identifier [dimension { dimension }]
{ , event_identifier [dimension { dimension }] }

list_of_genvar_identifiers ::= genvar_identifier { , genvar_identifier }

list_of_net_decl_assignments ::= net_decl_assignment { , net_decl_assignment }

list_of_net_identifiers ::= net_identifier [dimension { dimension }]
{ , net_identifier [dimension { dimension }] }

list_of_param_assignments ::= param_assignment { , param_assignment }

list_of_port_identifiers ::= port_identifier { , port_identifier }

list_of_real_identifiers ::= real_type { , real_type }

```

list_of_specparam_assignments ::=
    specparam_assignment { , specparam_assignment }

list_of_variable_identifiers ::= variable_type { , variable_type }

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

```

A.2.4 Declaration assignments

```

net_decl_assignment ::= net_identifier = expression

param_assignment ::= parameter_identifier = constant_expression

specparam_assignment ::=
    specparam_identifier = constant_mintypmax_expression
    | pulse_control_specparam

pulse_control_specparam ::=
    PATHPULSE$ = ( reject_limit_value [ , error_limit_value ] );
    | PATHPULSE$specify_input_terminal_descriptor$specify_output_terminal_descriptor
      = ( reject_limit_value [ , error_limit_value ] );

error_limit_value ::= limit_value

reject_limit_value ::= limit_value

limit_value ::= constant_mintypmax_expression

```

A.2.5 Declaration ranges

```

dimension ::= [ dimension_constant_expression : dimension_constant_expression ]

range ::= [ msb_constant_expression : lsb_constant_expression ]

```

A.2.6 Function declarations

```

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

function_item_declaration ::=
    block_item_declaration
    | tf_input_declaration ;

function_port_list ::=
    { attribute\_instance } tf_input_declaration { , { attribute\_instance } tf_input_declaration }

```

range_or_type ::= range | **integer** | **real** | **realtime** | **time**

A.2.7 Task declarations

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

```
task_item_declaration ::=
    block_item_declaration
| { attribute\_instance } tf_input_declaration ;
| { attribute\_instance } tf_output_declaration ;
| { attribute\_instance } tf_inout_declaration ;
```

```
task_port_list ::= task_port_item { , task_port_item }
```

```
task_port_item ::=
    { attribute\_instance } tf_input_declaration
| { attribute\_instance } tf_output_declaration
| { attribute\_instance } tf_inout_declaration
```

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

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

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

```
task_port_type ::=
    time | real | realtime | integer
```

A.2.8 Block item declarations

```
block_item_declaration ::=
    { attribute\_instance } block_reg_declaration
+ { attribute\_instance } event_declaration
| { attribute\_instance } integer_declaration
| { attribute\_instance } local_parameter_declaration
| { attribute\_instance } parameter_declaration
+ { attribute\_instance } real_declaration
+ { attribute\_instance } realtime_declaration
+ { attribute\_instance } time_declaration
```

```
block_reg_declaration ::= reg [ signed ] [ range ]
    list_of_block_variable_identifiers ;
```

```
list_of_block_variable_identifiers ::=
    block_variable_type { , block_variable_type }

block_variable_type ::=
    variable_identifier
    | variable_identifier dimension { dimension }
```

A.3 Primitive instances

A.3.1 Primitive instantiation and instances

```
gate_instantiation ::=
    emos_swichtype [delay3] emos_switch_instance { , emos_switch_instance } ;
    | enable_gatetype [drive_strength] [delay3] enable_gate_instance
      { , enable_gate_instance } ;
    mos_swichtype [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_swichtype [delay3] pass_enable_switch_instance
      { , pass_enable_switch_instance } ;
    pass_swichtype pass_switch_instance { , pass_switch_instance } ;
    pulldown [pulldown_strength] pull_gate_instance { , pull_gate_instance } ;
    pullup [pullup_strength] pull_gate_instance { , pull_gate_instance } ;
```

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

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

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

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

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

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

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

```
pull_gate_instance ::= [name_of_gate_instance] ( output_terminal )
```

```
name_of_gate_instance ::= gate_instance_identifier [ range ]
```

A.3.2 Primitive strengths

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

```
| ( strength1 , strength0 )  
| ( strength0 )
```

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

A.3.3 Primitive terminals

enable_terminal ::= expression

inout_terminal ::= net_lvalue

input_terminal ::= expression

necontrol_terminal ::= expression

output_terminal ::= net_lvalue

pecontrol_terminal ::= expression

A.3.4 Primitive gate and switch types

emos_swichtype ::= **cmos** | **rcmos**

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

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

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

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

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

pass_swichtype ::= **tran** | **rtran**

A.4 Module and generated instantiation

A.4.1 Module instantiation

```
module_instantiation ::=  
    module_identifier [ parameter_value_assignment ] module_instance { ,  
        module_instance } ;
```

parameter_value_assignment ::= # (list_of_parameter_assignments)

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

ordered_parameter_assignment ::= expression

```
named_parameter_assignment ::= . parameter_identifier ( [ expression ] )
```

```

module_instance ::= name_of_instance ( [ list_of_port_connections ] )
name_of_instance ::= module_instance_identifier [ range ]
list_of_port_connections ::=
    ordered_port_connection { , ordered_port_connection }
    | named_port_connection { , named_port_connection }
ordered_port_connection ::= { attribute\_instance } [ expression ]
named_port_connection ::= { attribute\_instance } . port_identifier ( [ expression ] )

```

A.4.2 Generated instantiation

```

generated_instantiation ::= generate { generate_item } endgenerate
generate_item_or_null ::= generate_item | ;
generate_item ::=
    generate_conditional_statement
    | generate_case_statement
    | generate_loop_statement
    | generate_block
    | module_or_generate_item
generate_conditional_statement ::=
    if ( constant_expression ) generate_item_or_null
    [ else generate_item_or_null ]
generate_case_statement ::= case ( constant_expression )
    genvar_case_item { genvar_case_item } endcase
genvar_case_item ::= constant_expression { , constant_expression } ;
    generate_item_or_null | default [ : ] generate_item_or_null
generate_loop_statement ::=
    for ( genvar_assignment ; constant_expression ; genvar_assignment )
    begin : generate_block_identifier { generate_item } end
genvar_assignment ::= genvar_identifier = constant_expression
generate_block ::= begin [ : generate_block_identifier ] { generate_item } end

```

A.5 UDP declaration and instantiation

A.5.1 UDP declaration

```

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

```

A.5.2 UDP ports

```
udp_port_list ::=
    output_port_identifier , input_port_identifier { , input_port_identifier }

udp_declaration_port_list ::=
    udp_output_declaration , udp_input_declaration { , udp_input_declaration }

udp_port_declaration ::=
    udp_output_declaration ;
    | udp_input_declaration ;
    | udp_reg_declaration ;

udp_output_declaration ::=
    { attribute_instance } output port_identifier
    | { attribute_instance } output reg port_identifier [ = constant_expression ]

udp_input_declaration ::= { attribute_instance } input list_of_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_udp_instance] (output_terminal , input_terminal
{ , input_terminal })

`name_of_udp_instance` ::= udp_instance_identifier [range]

A.6 Behavioral statements

A.6.1 Continuous assignment statements

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

list_of_net_assignments ::= net_assignment { , net_assignment }

net_assignment ::= net_lvalue = expression

A.6.2 Procedural blocks and assignments

`initial_construct` ::= **initial** statement

always_construct ::= **always** statement

blocking assignment ::=
variable_lvalue = [`delay_or_event_control`] expression

nonblocking assignment ::=
variable_lvalue <= [`delay_or_event_control`] expression

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

function_blocking_assignment ::= variable_lvalue = expression

function_statement_or_null ::=
function_statement
| { `attribute_instance` } ;

A.6.3 Parallel and sequential blocks

function_seq_block ::= **begin** [: block_identifier
{ block_item_declaration }] { function_statement } **end**

variable_assignment ::= variable_lvalue = expression

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

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

A.6.4 Statements

```
statement ::=  
    { attribute\_instance } blocking_assignment ;  
    | { attribute\_instance } case_statement  
    | { attribute\_instance } conditional_statement  
    | { attribute\_instance } disable_statement  
    | { attribute\_instance } event_trigger  
    | { attribute\_instance } loop_statement  
    | { attribute\_instance } nonblocking_assignment ;  
    | { attribute\_instance } par_block  
    | { attribute\_instance } procedural_continuous_assignments ;  
    | { attribute\_instance } procedural_timing_control_statement  
    | { attribute\_instance } seq_block  
    | { attribute\_instance } system\_task\_enable  
    | { attribute\_instance } task_enable  
    | { attribute\_instance } wait_statement
```

```
statement_or_null ::=  
    statement  
    | { attribute\_instance } ;
```

```
function_statement ::=  
    { attribute\_instance } function_blocking_assignment ;  
    | { attribute\_instance } function_case_statement  
    | { attribute\_instance } function_conditional_statement  
    | { attribute\_instance } function_loop_statement  
    | { attribute\_instance } function_seq_block  
    | { attribute\_instance } disable_statement  
    | { attribute\_instance } system\_task\_enable
```

A.6.5 Timing control statements

```
delay\_control ::=  
    # delay_value  
    | # ( mintypmax_expression )
```

```
delay_or_event_control ::=  
    delay\_control  
    | event_control  
    | repeat ( expression ) event_control
```

```
disable_statement ::=  
    | disable hierarchical_task_identifier ;  
    | disable hierarchical_block_identifier ;
```

```
event_control ::=  
    @ event_identifier  
    | @ ( event_expression )  
    | @ *  
    | @ ( * )
```

```

event_trigger ::=
    -> hierarchical_event_identifier ;

event_expression ::=
    expression
    | hierarchical_identifier
    | posedge expression
    | negedge expression
    | event_expression or event_expression
    | event_expression , event_expression

procedural_timing_control_statement ::=
    delay_or_event_control statement_or_null

wait_statement ::=
    wait ( expression ) statement_or_null

```

A.6.6 Conditional statements

```

conditional_statement ::=
    if ( expression ) statement_or_null [ else statement_or_null ]
    | if_else_if_statement

if_else_if_statement ::=
    if ( expression ) statement_or_null
    { else if ( expression ) statement_or_null }
    [ else statement_or_null ]

function_conditional_statement ::=
    if ( expression ) function_statement_or_null
    [ else function_statement_or_null ]
    | function_if_else_if_statement

function_if_else_if_statement ::=
    if ( expression ) function_statement_or_null
    { else if ( expression ) function_statement_or_null }
    [ else function_statement_or_null ]

```

A.6.7 Case statements

```

case_statement ::=
    case ( expression ) case_item { case_item } endcase
    | casez ( expression ) case_item { case_item } endcase
    | casex ( expression ) case_item { case_item } endcase

case_item ::=
    expression { , expression } : statement_or_null
    | default [ : ] statement_or_null

function_case_statement ::=
    case ( expression ) function_case_item { function_case_item } endcase
    | casez ( expression ) function_case_item { function_case_item } endcase
    | casex ( expression ) function_case_item { function_case_item } endcase

```

```
function_case_item ::=  
    expression { , expression } : function_statement_or_null  
    | default [ : ] function_statement_or_null
```

A.6.8 Looping statements

```
function_loop_statement ::=  
    forever function_statement  
    | repeat ( expression ) function_statement  
    | while ( expression ) function_statement  
    | for ( variable_assignment ; expression ; variable_assignment )  
        function_statement
```

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

A.6.9 Task enable statements

```
system_task_enable ::=  
    system_task_identifier [ ( expression { , expression } ) ] ;
```

```
task_enable ::=  
    hierarchical_task_identifier [ ( expression { , expression } ) ] ;
```

A.7 Specify section

A.7.1 Specify block declaration

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

```
specify_item ::=  
    specparam_declaration  
    | pulsestyle_declaration  
    | showcanceled_declaration  
    | path_declaration  
    | system_timing_check
```

```
pulsestyle_declaration ::=  
    pulsestyle_onevent list_of_path_output ;  
    | 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
| input_identifier [ constant_expression ]
| input_identifier [ range_expression ]
```

```
specify_output_terminal_descriptor ::=
  output_identifier
| output_identifier [ constant_expression ]
| output_identifier [ range_expression ]
```

```
input_identifier ::= input_port_identifier | inout_port_identifier
```

```
output_identifier ::= output_port_identifier | inout_port_identifier
```

A.7.4 Specify path delays

```
path_delay_value ::=
  list_of_path_delay_expressions
| ( list_of_path_delay_expressions )
```

```
list_of_path_delay_expressions ::=
  t_path_delay_expression
| trise_path_delay_expression , tfall_path_delay_expression
| trise_path_delay_expression , tfall_path_delay_expression ,
  tz_path_delay_expression
| t01_path_delay_expression , t10_path_delay_expression ,
  t0z_path_delay_expression ,
  tz1_path_delay_expression , t1z_path_delay_expression ,
  tz0_path_delay_expression
| t01_path_delay_expression , t10_path_delay_expression ,
  t0z_path_delay_expression ,
  tz1_path_delay_expression , t1z_path_delay_expression ,
  tz0_path_delay_expression ,
  t0x_path_delay_expression , tx1_path_delay_expression ,
```

`tlx_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 [, [notify_reg]]);

[\\$hold_timing_check](#) ::=
\$hold (reference_event , data_event , timing_check_limit [, [notify_reg]]);

[\\$setuphold_timing_check](#) ::=
\$setuphold (reference_event , data_event , timing_check_limit , timing_check_limit
 [, [notify_reg] [, [stamptime_condition] [, [checktime_condition]
 [, [delayed_reference] [, [delayed_data]]]]]);

[\\$recovery_timing_check](#) ::=
\$recovery (reference_event , data_event , timing_check_limit [, [notify_reg]]);

[\\$removal_timing_check](#) ::=
\$removal (reference_event , data_event , timing_check_limit [, [notify_reg]]);

[\\$recrem_timing_check](#) ::=
\$recrem (reference_event , data_event , timing_check_limit , timing_check_limit
 [, [notify_reg] [, [stamptime_condition] [, [checktime_condition]
 [, [delayed_reference] [, [delayed_data]]]]]);

[\\$skew_timing_check](#) ::=
\$skew (reference_event , data_event , timing_check_limit [, [notify_reg]]);

[\\$timeskew_timing_check](#) ::=
\$timeskew (reference_event , data_event , timing_check_limit
 [, [notify_reg] [, [event_based_flag] [, [remain_active_flag]]]]);

[\\$fullskew_timing_check](#) ::=
\$fullskew (reference_event , data_event , timing_check_limit , timing_check_limit
 [, [notify_reg] [, [event_based_flag] [, [remain_active_flag]]]]);

[\\$period_timing_check](#) ::=
\$period (controlled_reference_event , timing_check_limit [, [notify_reg]]);

\$width timing check ::=
 \$width (controlled_reference_event , timing_check_limit , threshold [, [notify_reg]]);

\$nochange timing check ::=
 \$nochange (reference_event , data_event , start_edge_offset ,
 end_edge_offset [, [notify_reg]]);

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

notify_reg ::= 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 ::=

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 ::= { constant_expression concatenation }

net_concatenation ::= { net_concatenation_value { , net_concatenation_value } }

net_concatenation_value ::=

hierarchical_net_identifier
| hierarchical_net_identifier [expression] { [expression] }
| hierarchical_net_identifier [expression] { [expression] } [range_expression]
| hierarchical_net_identifier [range_expression]
| net_concatenation

variable_concatenation ::= { variable_concatenation_value { , variable_concatenation_value } }

variable_concatenation_value ::=

hierarchical_variable_identifier

| hierarchical_variable_identifier [expression] { [expression] }
| hierarchical_variable_identifier [expression] { [expression] } [range_expression]
| hierarchical_variable_identifier [range_expression]
| variable_concatenation

A.8.2 Function calls

constant_function_call ::= function_identifier { [attribute_instance](#) }
 (constant_expression { , constant_expression })

function_call ::= hierarchical_function_identifier { [attribute_instance](#) }
 (expression { , expression })

genvar_function_call ::= genvar_function_identifier { [attribute_instance](#) }
 (constant_expression { , constant_expression })

~~system_function_call~~ ::= system_function_identifier
 [(expression { , expression })]

A.8.3 Expressions

base_expression ::= expression

conditional_expression ::= expression1 ? { [attribute_instance](#) } expression2 : expression3

constant_base_expression ::= constant_expression

constant_expression ::=
 constant_primary
 | unary_operator { [attribute_instance](#) } constant_primary
 | constant_expression binary_operator { [attribute_instance](#) } constant_expression
 | constant_expression ? { [attribute_instance](#) } constant_expression : constant_expression
 | string

~~constant_mintypmax_expression~~ ::=
 constant_expression
 | constant_expression : constant_expression : constant_expression

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

dimension_constant_expression ::= constant_expression

expression1 ::= expression

expression2 ::= expression

expression3 ::= expression

expression ::=
 primary
 | unary_operator { [attribute_instance](#) } primary
 | expression binary_operator { [attribute_instance](#) } expression

| conditional_expression
| string

lsb_constant_expression ::= constant_expression

[mintypmax_expression](#) ::=
expression
| expression : expression : expression

module_path_conditional_expression ::=
module_path_expression ? { [attribute_instance](#) }
module_path_expression : module_path_expression

module_path_expression ::=
module_path_primary
| unary_module_path_operator { [attribute_instance](#) } module_path_primary
| module_path_expression binary_module_path_operator
{ [attribute_instance](#) } module_path_expression
| module_path_conditional_expression

module_path_mintypmax_expression ::=
module_path_expression
| module_path_expression : module_path_expression : module_path_expression

msb_constant_expression ::= constant_expression

range_expression ::=
expression
| msb_constant_expression : lsb_constant_expression
| base_expression +: width_constant_expression
| base_expression -: width_constant_expression

width_constant_expression ::= constant_expression

A.8.4 Primaries

constant_primary ::=
constant_concatenation
| constant_function_call
| (constant_mintypmax_expression)
| constant_multiple_concatenation
| genvar_identifier
| number
| parameter_identifier
| specparam_identifier

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

```

primary ::=
    number
  | hierarchical_identifier
  | hierarchical_identifier [ expression ] { [ expression ] }
  | hierarchical_identifier [ expression ] { [ expression ] }
    [ range_expression ]
  | hierarchical_identifier [ range_expression ]
  | concatenation
  | multiple_concatenation
  | function_call
  | system_function_call
  | constant_function_call
  | ( mintypmax_expression )
    
```

A.8.5 Expression left-side values

```

net_lvalue ::=
    hierarchical_net_identifier
  | hierarchical_net_identifier [ constant_expression ] { [ constant_expression ] }
  | hierarchical_net_identifier [ constant_expression ] { [ constant_expression ] }
    [ constant_range_expression ]
  | hierarchical_net_identifier [ constant_range_expression ]
  | net_concatenation
    
```

```

variable_lvalue ::=
    hierarchical_variable_identifier
  | hierarchical_variable_identifier [ expression ] { [ expression ] }
  | hierarchical_variable_identifier [ expression ] { [ expression ] }
    [ range_expression ]
  | hierarchical_variable_identifier [ range_expression ]
  | variable_concatenation
    
```

A.8.6 Operators

```

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

```

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

```

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

```

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

A.8.7 Numbers

```

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

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

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

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 ::= non_zero_decimal_digit { _ | decimal_digit }

unsigned_number ::= decimal_digit { _ | decimal_digit }

binary_value ::= binary_digit { _ | binary_digit }

octal_value ::= octal_digit { _ | octal_digit }

hex_value ::= hex_digit { _ | hex_digit }

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

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

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

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

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

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

binary_digit ::= x_digit | z_digit | **0** | **1**

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

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

x_digit ::= **x** | **X**

z_digit ::= **z** | **Z** | **?**

A.8.8 Strings

string ::= " { Any_ASCII_Characters_except_new_line } "

A.9 General

A.9.1 Attributes

`attribute_instance ::= (* attr_spec { , attr_spec } *)`

`attr_spec ::=
 attr_name = constant_expression
 | attr_name`

`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

`arrayed_identifier ::=
 simple_arrayed_identifier
 | escaped_arrayed_identifier`

`block_identifier ::= identifier`

`cell_identifier ::= identifier`

`config_identifier ::= identifier`

`escaped_arrayed_identifier ::= escaped_identifier [range]`

`escaped_hierarchical_identifier ::=
 escaped_hierarchical_branch (simple_hierarchical_branch
 escaped_hierarchical_branch)`

`escaped_identifier ::= \ { Any_ASCII_character_except_white_space } white_space`

`event_identifier ::= identifier`

`function_identifier ::= identifier`

`gate_instance_identifier ::= arrayed_identifier`

`generate_block_identifier ::= identifier`

`genvar_function_identifier ::= identifier /* hierarchy disallowed */`

`genvar_identifier ::= identifier`

```

hierarchical_block_identifier ::= hierarchical_identifier
hierarchical_event_identifier ::= hierarchical_identifier
hierarchical_function_identifier ::= hierarchical_identifier

hierarchical_identifier ::=
    simple_hierarchical_identifier
    | escaped_hierarchical_identifier

hierarchical_net_identifier ::= hierarchical_identifier
hierarchical_variable_identifier ::= hierarchical_identifier
hierarchical_task_identifier ::= hierarchical_identifier

identifier ::=
    simple_identifier
    | escaped_identifier

inout_port_identifier ::= identifier
input_port_identifier ::= identifier
instance_identifier ::= identifier
library_identifier ::= identifier
memory_identifier ::= identifier
module_identifier ::= identifier
module_instance_identifier ::= arrayed_identifier
net_identifier ::= identifier
output_port_identifier ::= identifier
parameter_identifier ::= identifier
port_identifier ::= identifier
real_identifier ::= identifier
simple_arrayed_identifier ::= simple_identifier [ range ]

simple_hierarchical_identifier ::=
    simple_hierarchical_branch { escaped_identifier }
simple_identifier ::= [ a-zA-Z_ ] { [ a-zA-Z0-9_$ ] }

specparam_identifier ::= identifier

system_function_identifier ::= $[ a-zA-Z0-9_$ ] { [ a-zA-Z0-9_$ ] }
system_task_identifier ::= $[ a-zA-Z0-9_$ ] { [ a-zA-Z0-9_$ ] }

task_identifier ::= identifier
terminal_identifier ::= identifier

```

text_macro_identifier ::= simple_identifier

topmodule_identifier ::= identifier

udp_identifier ::= identifier

udp_instance_identifier ::= arrayed_identifier

variable_identifier ::= identifier

A.9.4 Identifier branches

simple_hierarchical_branch ::=
simple_identifier~~[[unsigned_number]]~~
~~[[.simple_identifier [[unsigned_number]]]]~~

escaped_hierarchical_branch ::=
escaped_identifier~~[[unsigned_number]]~~
~~[[.escaped_identifier [[unsigned_number]]]]~~

A.9.5 White space

white_space ::= space | tab | newline | eof