

Annex A

Syntax

This annex contains the formal syntax definition of Verilog-AMS HDL. The conventions used are described in Section 1. The Verilog-AMS HDL syntax extends and modifies the *IEEE 1364-2001 Verilog HDL* syntax. This syntax should be read in conjunction with the *IEEE 1364-2001 Verilog HDL* syntax. The following type of syntax categories are present in this annex:

1. *IEEE 1364-2001 Verilog HDL* syntax categories are represented in blue with cyan keywords and punctuation. This syntax is present for readability.
2. Verilog-AMS HDL syntax categories are represented in black with red keywords and punctuation. These syntax categories are extensions to the *IEEE 1364-2001 Verilog HDL* syntax.
3. Modified *IEEE 1364-2001 Verilog HDL* syntax categories are displayed blue with cyan keywords and punctuation with the Verilog-AMS extensions represented in black with red keywords and punctuation. The categories in this annex supersedes the same categories in the *IEEE 1364-2001 Verilog HDL* annex A.

A.1 Source text

A.1.1 Library source text

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.1.2 Configuration source text

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.1.3 Module and primitive source text

The syntax below extends and modifies the syntax found in the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

```
description ::=  
  module_declaration  
  | udp_declaration  
  | discipline_declaration
```

```
| nature_declaration
| connectrules_specification
```

```
module_keyword ::= module | macromodule | connectmodule
```

NOTE: Semantically restrict connectmodule port list to be ordered only

NOTE: Semantically restrict connectmodule port list to be size 2 only

NOTE: Semantically restrict connectmodule ports to be scalar.

NOTE: Semantically restrict a connectmodule ports to be non-empty

NOTE: removed duplicate "source_text" 2001 syntax item

A.1.4 Module parameters and ports

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.1.5 Module Items

The syntax below extends and modifies the syntax found in the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

```
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
  | analog_block
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
  | ground_declaration
  | net_discipline_declaration
  | branch_declaration
  | analog_function_declaration
non_port_module_item ::=
  module_or_generate_item
  | port_declaration ;
  | { 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  
| analog_block
```

NOTE: removed "module_items" syntax item

NOTE: Semantically restrict analog block to appear zero or one time in a module declaration.

NOTE: removed duplicate "parameter_override" syntax.

NOTE: AMS declarations now can be prefixed with attributes.

A.1.6 Natures

```
nature_declaration ::=  
    nature nature_identifier [ : parent_nature ]  
    { nature_item }  
    endnature  
parent_nature ::=  
    nature_identifier  
    | discipline_identifier . potential_or_flow  
nature_item ::= nature_attribute  
nature_attribute ::= nature_attribute_identifier = nature_attribute_expression ;  
nature_attribute_expression ::=  
    constant_expression  
    | nature_identifier  
    | nature_access_identifier
```

NOTE: natures should be terminated with a ';' to be consistent with connectrules, module and udp parameter declarations.

NOTE: nature attribute identifiers move to section A.11.3

NOTE: nature attribute expression syntax item defined since constant expression cannot handle nature identifiers and nature access identifiers.

A.1.7 Disciplines

```
discipline_declaration ::=  
    discipline discipline_identifier  
    { discipline_item }  
    enddiscipline  
discipline_item ::=  
    nature_binding  
    | discipline_domain_binding  
    | nature_attribute_override  
nature_binding ::= potential_or_flow nature_identifier ;  
potential_or_flow ::= potential | flow  
discipline_domain_binding ::= domain discrete_or_continuous ;  
discrete_or_continuous ::= discrete | continuous  
nature_attribute_override ::= potential_or_flow . nature_attribute
```

NOTE: disciplines should be terminated with a ';' to be consistent with connectrules, module and udp parameter declarations.

A.1.8 Connectrule Specifications

```
connectrules_specification ::=
    connectrules connectrules_identifier ;
    { connectrules_item }
endconnectrules
connectrules_item ::=
    connect_insertion
    | connect_resolution
connect_insertion ::=
    connect connectmodule_identifier [ connect_mode ] [ parameter_value_assignment ] [ connect_port_overrides ] ;
connect_mode ::= merged | split
connect_port_overrides ::= [ port_direction ] discipline_identifier , [ port_direction ] discipline_identifier
port_direction ::= input | output | inout
connect_resolution ::= connect discipline_identifier { , discipline_identifier } resolveto discipline_identifier ;
```

NOTE: connect_insertion syntax reusing digital parameter override syntax. Now can have order and named overrides.

NOTE: changed keyword "merge" to "merged".

A.2 Declarations

A.2.1 Declaration types

A.2.1.1 Module parameter declarations

The syntax below extends and modifies the syntax found in the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

```
parameter_declaration ::=
    parameter [opt_type] list_of_param_assignments ;
opt_type ::=
    real
    | integer
declarator_init ::=
    parameter_identifier = constant_expression { opt_value_range }
    | parameter_array_identifier range = constant_param_arrayinit { opt_value_range }
opt_value_range ::=
    from value_range_specifier
    | exclude value_range_specifier
    | exclude value_constant_expression
value_range_specifier ::=
    start_range_spec expression1 : expression2 end_range_spec
value_range_specifier ::=
    start_paren expression1 : expression2 end_paren
start_paren ::=
    [ (
end_paren ::=
    ] )
```

```

expression1 ::=
    constant_expression | -inf
expression2 ::=
    constant_expression | inf
constant_param_arrayinit ::=
    { param_arrayinit_element_list }
param_arrayinit_element_list
    param_arrayinit_element { , param_arrayinit_element }
param_arrayinit_element ::=
    constant_expression
    | { replicator_constant_expression {constant_expression} }

```

A.2.1.2 Port declarations

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.2.1.3 Type declarations

The syntax below extends and modifies the syntax found in the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

```

integer_declaration ::=
    integer list_of_identifiers ;
real_declaration ::=
    real list_of_identifiers ;
var_name ::=
    variable_identifier
    | array_identifier array_range
ground_declaration ::=
    ground [ range ] list_of_nets ;
net_discipline_declaration ::=
    discipline_identifier [range] list_of_nets ;
list_of_nets ::=
    net_identifier [ range ]
    | net_identifier [ range ] , list_of_nets
digital_net_declaration ::=
    digital_net_declaration
    | wreal [ list_of_identifiers ] ;
branch_declaration ::=
    branch list_of_branches ;
list_of_branches ::=
    terminals list_of_branch_identifiers
terminals ::=
    ( net_or_port_scalar_expression )
    | ( net_or_port_scalar_expression , net_or_port_scalar_expression )
genvar_declaration ::=
    genvar list_of_genvar_identifiers ;

```

A.2.2 Declaration data types

A.2.2.4 Net and variable types

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.2.2.5 Strengths

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.2.2.6 Delays

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.2.3 Declaration lists

The syntax below extends and modifies the syntax found in the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

```
list_of_genvar_identifiers ::=  
    genvar_identifier { , genvar_identifier }  
list_of_branch_identifiers ::=  
    branch_identifier [ range ]  
    | branch_identifier [ range ] , list_of_branch_identifiers  
list_of_identifiers ::=  
    var_name { , var_name }  
list_of_param_assignments ::=  
    declarator_init { , declarator_init }
```

A.2.4 Declaration assignments

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.2.5 Declaration ranges

The syntax below extends and modifies the syntax found in the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

```
array_range ::=  
    [ upper_limit_constant_expression : lower_limit_constant_expression ]  
range ::=  
    [ constant_expression : constant_expression ]  
range ::=  
    [ constant_expression : constant_expression ]
```

A.2.6 Function declarations

The syntax below extends and modifies the syntax found in the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.2.7 Analog function declarations

```
analog_function_declaration ::=  
    analog function [ type ] function_identifier ;  
    function_item_declaration { function_item_declaration }  
    statement  
    endfunction  
type ::=  
    integer  
    | real  
function_item_declaration ::=  
    block_item_declaration  
    | input_declaration
```

A.2.8 Task declarations

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.2.9 Block item declarations

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.2.10 Analog block item declarations

```
block_item_declaration ::=  
    parameter_declaration  
    | integer_declaration  
    | real_declaration
```

A.3 Primitive instances

A.3.1 Primitive instantiation and instances

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.3.2 Primitive strengths

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.3.3 Primitive terminals

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.3.4 Primitive gate and switch types

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.4 Module and generated instantiation

A.4.1 Module instantiation

The syntax below extends and modifies the syntax found in the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

```
module_instantiation ::=
    module_identifier [ parameter_value_assignment ] instance_list
instance_list ::=
    module_instance { , module_instance } ;
module_instance ::=
    name_of_instance ( [ list_of_module_connections ] )
name_of_instance ::=
    module_instance_identifier [ range ]
list_of_module_connections ::=
    ordered_port_connection { , ordered_port_connection }
    | named_port_connection { , named_port_connection }
ordered_port_connection ::=
    net_expression
named_port_connection ::=
    .port_identifier ( net_expression )
parameter_value_assignment ::=
    # ( ordered_param_override_list )
    | # ( named_param_override_list )
ordered_param_override_list ::=
    constant_or_constant_array_expression { , constant_or_constant_array_expression }
named_param_override_list ::=
    named_param_override { , named_param_override }
named_param_override ::=
    .parameter_identifier ( constant_or_constant_array_expression )
constant_or_constant_array_expression ::=
    constant_expression
    | constant_array_expression
net_expression ::=
    net_identifier
    | net_identifier [ expression ]
    | net_identifier [ msb_constant_expression : lsb_constant_expression ]
    | net_concatenation
net_concatenation ::=
    { net_expression_list }
net_expression_list ::=
    net_expression { , net_expression }
```


A.4.2 Generated instantiation

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.5 UDP declaration and instantiation

A.5.1 UDP declaration

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.5.2 UDP ports

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.5.3 UDP body

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.5.4 UDP instantiation

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.6 Behavioural statements

A.6.1 Continuous assignment statements

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.6.2 Procedural blocks and assignments

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.6.3 Parallel and sequential blocks

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.6.4 Statements

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.6.5 Timing control statements

The syntax below extends and modifies the syntax found in the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

```
digital_event_expression ::=  
    digital_expression  
    | event_identifier  
    | posedge digital_expression  
    | negedge digital_expression  
    | event_function  
    | digital_event_expression or digital_event_expression
```

A.6.6 Conditional statements

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.6.7 Case statements

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.6.8 Looping statements

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.6.9 Task enable statements

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.7 Analog behavioral statements

A.7.1 Procedural blocks and assignments

```
analog_block ::=  
    analog analog_statement  
analog_procedural_assignment ::=  
    lexpr = analog_expression ;  
lexpr ::=  
    integer_identifier  
    | real_identifier  
    | array_element  
array_element ::=  
    integer_identifier [ expression ]  
    | real_identifier [ expression ]  
procedural_assignment ::=  
    lexpr = expression ;
```

A.7.2 Sequential blocks

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

A.7.3 Analog statements

```
analog_statement ::=
    analog_seq_block
    | analog_branch_contribution
    | analog_indirect_branch_assignment
    | analog_procedural_assignment
    | analog_conditional_statement
    | analog_for_statement
    | analog_case_statement
    | analog_event_controlled_statement
    | system_task_enable
    | statement
statement ::=
    seq_block
    | procedural_assignment
    | conditional_statement
    | loop_statement
    | case_statement
analog_statement_or_null ::=
    analog_statement | ;
statement_or_null ::=
    statement | ;
```

A.7.4 Contribution statements

```
analog_branch_contribution ::=
    bvalue <+ analog_expression ;
analog_indirect_branch_assignment ::=
    bvalue : nexpr == analog_expression ;
nexpr ::=
    bvalue
    | pvalue
    | ddt ( bvalue | pvalue )
    | idt ( bvalue | pvalue )
```

A.7.5 Genvar loop statement

```
analog_for_statement ::=  
    for ( genvar_assignment ; genvar_expression ;  
        genvar_assignment ) analog_statement  
genvar_assignment ::=  
    genvar_identifier = genvar_expression
```

A.7.6 Event control statements

```
event_control_statement ::=  
    event_control statement_or_null  
event_control ::=  
    @ event_identifier  
    | @ ( event_expression )  
analog_event_expression ::=  
    global_event  
    | event_function  
    | digital_expression  
    | event_identifier  
    | posedge digital_expression  
    | negedge digital_expression  
    | event_expression or event_expression  
global_event ::=  
    initial_step [ ( analysis_list ) ]  
    | final_step [ ( analysis_list ) ]  
analysis_list ::=  
    analysis_name { , analysis_name }  
analysis_name ::=  
    " analysis_identifier "  
event_function ::=  
    cross_function  
    | timer_function  
cross_function ::=  
    cross ( arg_list )  
timer_function ::=  
    timer ( arg_list )  
driver_access_function ::=  
    driver_update | net_resolution
```

A.7.7 Conditional statements

```
conditional_statement ::=  
    if ( expression ) statement_or_null  
    [ else statement_or_null ]  
analog_conditional_statement ::=  
    if ( genvar_expression ) analog_statement_or_null  
    [ else analog_statement_or_null ]
```

A.7.8 Case statements

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

A.7.9 Analog loop statements

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

A.7.10 Analog task enable statements

```
system_task_enable ::=
    system_task_name [ ( expression { , expression } ) ] ;
system_task_name ::=
    $identifier
```

Note: The \$ may not be followed by a space.

A.8 Specify section

A.8.1 Specify block declaration

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.8.2 Specify path declarations

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.8.3 Specify block terminals

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.8.4 Specify path delays

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.8.5 System timing checks

A.8.5.1 System timing check commands

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.8.5.2 System timing check command arguments

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.8.5.3 System timing check event definitions

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.9 Expressions

A.9.1 Concatenations

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.9.2 Function calls

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.9.3 Expressions

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.9.4 Primaries

The syntax below extends and modifies the syntax found in the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

```
digital_primary ::=  
    primary
```

A.9.5 Expression left-side values

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.9.6 Operators

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.9.7 Numbers

The syntax below extends and modifies the syntax found in the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

```
digital_number ::=  
    number
```

A.9.8 Strings

The syntax below extends and modifies the syntax found in the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

```
string ::=  
    " { Any_ASCII_character_except_newline } "
```

A.10 AMS expressions

A.10.1 AMS concatenations

```
constant_array_expression ::=  
    { constant_arrayinit_element { , constant_arrayinit_element } }  
constant_arrayinit_element ::=  
    constant_expression  
    | integer_constant_expression { constant_expression }  
constant_array_expression ::=  
    { constant_array_init_element { , constant_array_init_element } }  
constant_array_init_element ::=  
    constant_expression  
    | integer_constant_expression { , constant_expression }  
concatenation ::=  
    { expression { , expression } }
```

A.10.2 AMS function calls

```
function_call ::=  
    function_identifier ( expression { , expression } )  
system_function ::=  
    $abstime | $realtime | $temperature | $vt | $bound_step | $discontinuity  
    | $driver_count | $driver_state | $driver_strength
```

A.10.3 AMS expressions

```
analog_expression ::=
    expression
    | analog_operator ( analog_operator_arg_list )
genvar_expression ::=
    genvar_primary
    | unary_operator genvar_primary
    | genvar_expression binary_operator genvar_primary
    | genvar_expression ? genvar_expression : genvar_expression
    | string
constant_expression ::=
    constant_primary
    | string
    | unary_operator constant_primary
    | constant_expression binary_operator constant_expression
    | constant_expression ? constant_expression : constant_expression
    | constant_array_expression
    | attribute_reference
    | built_in_function ( const_arg_list )
const_arg_list ::=
    constant_expression { , constant_expression }
expression ::=
    primary
    | unary_operator primary
    | expression binary_operator expression
    | expression ? expression : expression
    | function_call
    | access_function_reference
    | built_in_function ( arg_list )
    | system_function ( arg_list )
```

A.10.4 Analog operators

```
analog_operator_arg_list ::=
    analog_operator_argument { , analog_operator_argument }
analog_operator_argument ::=
    expression
    | constant_array_expression
    | analog_operator ( analog_operator_arg_list )
analog_operator ::=
    ddt | idt | idtmod | absdelay | transition | slew |
    | laplace_zd | laplace_zp | laplace_np | laplace_nd
    | zi_zp | zi_zd | zi_np | zi_nd | last_crossing | ac_stim
    | limexp | white_noise | flicker_noise | noise_table
```

A.10.5 AMS primaries

```
genvar_primary ::=
    constant_primary
```



```

    | genvar_identifier
    | genvar_identifier [ genvar_expression ]
    | analysis ( arg_list )
arg_list ::=
    argument { , argument }
argument ::=
    expression
    | constant_array_expression
constant_primary ::=
    number
    | parameter_identifier
    | constant_concatenation
primary ::=
    number
    | identifier
    | identifier [ expression ]
    | identifier [ digital_msb_constant_expression : digital_lsb_constant_expression ]
    | digital_concatenation
    | digital_multiple_concatenation
    | digital_function_call
    | (digital_mintypmax_expression )
    | string
    | nexpr
    | ( expression )

```

A.10.6 Nature attribute accessing

```

attribute_reference ::=
    net_identifier . pot_or_flow . attribute_identifier

```

A.10.7 Analog Probes

```

access_function_reference ::=
    bvalue
    | pvalue
bvalue ::=
    access_identifier ( analog_signal_list )
analog_signal_list ::=
    branch_identifier
    | array_branch_identifier [ genvar_expression ]
    | net_or_port_scalar_expression
    | net_or_port_scalar_expression , net_or_port_scalar_expression
net_or_port_scalar_expression ::=
    net_or_port_identifier
    | array_net_or_port_identifier [ genvar_expression ]
    | vector_net_or_port_identifier [ genvar_expression ]
pvalue ::=
    flow_access_identifier ( < port_scalar_expression > )
port_scalar_expression ::=
    port_identifier

```

```
| array_port_identifier [ genvar_expression ]  
| vector_port_identifier [ genvar_expression ]
```

A.10.8 AMS operators

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

A.10.9 Mathematical functions

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

A.10.10 AMS numbers

```
number ::=  
decimal_number  
| digital_octal_number  
| digital_binary_number  
| digital_hex_number  
| real_number  
decimal_number ::=  
[ sign ] unsigned_num  
real_number ::=  
[ sign ] unsigned_num . unsigned_num  
| [ sign ] unsigned_num [ . unsigned_num ] e [ sign ] unsigned_num  
| [ sign ] unsigned_num [ . unsigned_num ] E [ sign ] unsigned_num  
| [ sign ] unsigned_num [ . unsigned_num ] scale_factor  
sign ::=  
+  
|-  
unsigned_num ::=  
decimal_digit { _ | decimal_digit }  
decimal_digit ::=  
0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9  
scale_factor ::=  
T | G | M | K | k | m | u | n | p | f | a
```

A.11 General

A.11.1 Attributes

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.11.2 Comments

The syntax below extends and modifies the syntax found in the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

```
comment ::=
    short_comment
  | long_comment
short_comment ::=
    // comment_text \n
long_comment ::=
    /* comment_text */
comment_text ::=
    { Any_ASCII_character }
```

A.11.3 Identifiers

The syntax below extends and modifies the syntax found in the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

```
connectmodule_identifier ::= module_identifier
connectrules_identifier ::= identifier
nature_identifier ::= identifier
nature_attribute_identifier ::=
    abstol
  | access
  | ddt_nature
  | idt_nature
  | units
  | identifier
```

```
discipline_identifier ::= identifier
```

```
identifier ::=
    IDENTIFIER [ { . IDENTIFIER } ]
```

NOTE: The period in identifier may not be preceded or followed by a space.

```
IDENTIFIER ::=
```

```
    simple_identifier
  | escaped_identifier
```

```
simple_identifier ::=
    [ a-zA-Z_ ] { a-zA-Z_$0-9 }
```

```
escaped_identifier ::=
    \ { Any_ASCII_character_except_white_space } white_space
```

A.11.4 Identifier branches

Refer to the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

A.11.5 White space

The syntax below extends and modifies the syntax found in the equivalent section in *IEEE 1364-2001 Verilog HDL* annex A.

```
white_space ::=  
    space  
    | tab  
    | newline  
    | formfeeds
```