

Annex A

Syntax

Issue 96 is about supporting attributes for disciplines --Martin

This annex contains the formal syntax definition of Verilog-AMS HDL. The conventions used are described in Section 1. Any category whose name begins with the italicized word *digital_* should be interpreted by its definition in the grammar given in *IEEE 1364-1995 Verilog HDL Annex A*, and not by the local definition given herein. When such a category is defined herein (e.g., *digital_primary ::=*), that definition should be taken to supersede the definition in *IEEE 1364-1995 Verilog HDL* when used for Verilog-AMS HDL. If any category is defined here and also in *IEEE 1364-1995 Verilog HDL*, it is a repetition of *IEEE 1364-1995 Verilog HDL*. This is done for the convenience of readers of this LRM.

A.1 Source text

```

source_text ::=
    { description }
description ::=
    module_declaration
    | discipline_definition
    | nature_definition
    | connect_specification
    | digital_udp_declaration
module_declaration ::=
    module_keyword module_identifier [ digital_list_of_ports ] ;
    [ module_items ]
    endmodule
module_keyword ::=
    module
    | macromodule
module_items ::=
    { module_item }
    | analog_block
module_item ::=
    module_item_declaration
    | parameter_override
    | module_instantiation
    | digital_continuous_assignment
    | digital_gate_instantiation
    | digital_udp_instantiation
    | digital_specify_block

```

```

| digital_initial_construct
| digital_always_construct
module_item_declaration ::=
  parameter_declaration
| digital_input_declaration
| digital_output_declaration
| digital_inout_declaration
| ground_declaration
| integer_declaration
| real_declaration
| net_discipline_declaration
| genvar_declaration
| branch_declaration
| analog_function_declaration
| digital_function_declaration
| digital_net_declaration
| digital_reg_declaration
| digital_time_declaration
| digital_realtime_declaration
| digital_event_declaration
| digital_task_declaration
parameter_override ::=
  defparam list_of_param_assignments ;

```

A.2 Natures

```

nature_declaration ::=
  nature nature_name
  [ nature_descriptions ]
  endnature
nature_name ::=
  nature_identifier
| nature_identifier : parent_identifier
parent_identifier ::=
  nature_identifier
| discipline_identifier.flow
| discipline_identifier.potential
nature_descriptions ::=
  nature_description { nature_description }
nature_description ::=
  attribute = constant_expression ;
attribute ::=
  abstol
| access
| ddt_nature
| idt_nature
| units
| attribute_identifier

```

A.3 Disciplines

```

discipline_declaration ::=
    discipline discipline_identifier
    [ discipline_descriptions ]
    enddiscipline
discipline_descriptions ::=
    discipline_description { discipline_description }
discipline_description ::=
    nature_binding
    | attr_override
    | domain_binding
    | attribute_identifier = constant_expression;
nature_binding ::=
    pot_or_flow nature_identifier ;
attr_override ::=
    pot_or_flow . attribute_identifier = constant_expression ;
pot_or_flow ::=
    potential
    | flow

domain_binding ::=
    domain discrete
    | domain continuous

```

A.4 Declarations

```

digital_net_declaration ::=
    digital_net_declaration
    | wreal [ list_of_identifiers ] ;
parameter_declaration ::=
    parameter [ opt_type ] list_of_param_assignments ;
opt_type ::=
    real
    | integer
list_of_param_assignments ::=
    declarator_init { , declarator_init }
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} }
integer_declaration ::=
    integer list_of_identifiers ;
real_declaration ::=
    real list_of_identifiers ;
list_of_identifiers ::=
    var_name { , var_name }
var_name ::=
    variable_identifier
    | array_identifier array_range
array_range ::=
    [ upper_limit_constant_expression : lower_limit_constant_expression ]
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
range ::=
    [ constant_expression : constant_expression ]
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 )
list_of_branch_identifiers ::=
    branch_identifier [ range ]
    | branch_identifier [ range ] , list_of_branch_identifiers
genvar_declaration ::=
    genvar list_of_genvar_identifiers ;
list_of_genvar_identifiers ::=
    genvar_identifier { , genvar_identifier }
analog_function_declaration ::=
    analog function [ type ] function_identifier ;
    function_item_declaration { function_item_declaration }
    statement
    endfunction

```

```

type ::=
    integer
    | real
function_item_declaration ::=
    input_declaration
    | block_item_declaration
block_item_declaration ::=
    parameter_declaration
    | integer_declaration
    | real_declaration

```

A.5 Module instantiation

```

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.6 Mixed-signal

```

connectmodule_declaration ::=
    connectmodule module_identifier ( connectmod_port , connectmod_port ) ;
    [ module_items ]
    endmodule
connectmod_port ::=
    connectmod_port_identifier
connect_specification ::=
    connectrules connectrule_identifier ;
    { connect_spec_item }
    endconnectrules
connect_spec_item ::=
    connect_insertion
    | connect_resolution
connect_insertion ::=
    connect connect_module_identifier connect_attributes
    [ [ direction ] discipline_identifier , [ direction ] discipline_identifier ] ;
connect_attributes ::=
    [ connect_mode ] [ #( attribute_list ) ]
connect_mode ::=
    merge
    | split
attribute_list ::=
    attribute
    | attribute_list , attribute
attribute ::=
    ,parameter_identifier ( expression )
direction ::=
    input
    | output
    | inout
discipline_list ::=
    discipline_identifier
    | discipline_list , discipline_identifier
connect_resolution ::=
    connect discipline_list resolveto discipline_identifier ;

discipline_list ::=
    discipline_identifier
    | discipline_list , discipline_identifier

```

A.7 Behavioral statements

```

analog_block ::=
    analog analog_statement
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_seq_block ::=
    begin [ : block_identifier { block_item_declaration } ]
    { analog_statement }
    end
analog_statement_or_null ::=
    analog_statement | ;
analog_branch_contribution ::=
    bvalue <+ analog_expression ;
analog_indirect_branch_assignment ::=
    bvalue : nexpr == analog_expression ;
nexpr ::=
    bvalue
    | pvalue
    | ddt ( bvalue | pvalue )
    | idt ( bvalue | pvalue )
analog_procedural_assignment ::=
    lexpr = analog_expression ;
lexpr ::=
    integer_identifier
    | real_identifier
    | array_element
array_element ::=
    integer_identifier [ expression ]
    | real_identifier [ expression ]
analog_conditional_statement ::=
    if ( genvar_expression ) analog_statement_or_null
    [ else analog_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
analog_for_statement ::=
    for ( genvar_assignment ; genvar_expression ;
    genvar_assignment ) analog_statement
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
digital_event_expression ::=
    digital_expression
  | event_identifier
  | posedge digital_expression
  | negedge digital_expression
  | event_function
  | digital_event_expression or digital_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 )
statement_or_null ::=
    statement | ;
system_task_enable ::=
    system_task_name [ ( expression { , expression } ) ];
system_task_name ::=
    $identifier
Note: The $ may not be followed by a space.
seq_block ::=
    begin [ : block_identifier { block_item_declaration } ]
    { statement }
    end
procedural_assignment ::=
    lexpr = expression ;
conditional_statement ::=
    if ( expression ) statement_or_null
    [ else statement_or_null ]
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
loop_statement ::=
    repeat ( expression ) statement
    | while ( expression ) statement
    | for ( procedural_assignment ; expression ;
    procedural_assignment ) statement

```

A.8 Analog expressions

```

analog_expression ::=
    expression
    | analog_operator ( analog_operator_arg_list )
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 | idtm | 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
genvar_expression ::=
    genvar_primary
    | unary_operator genvar_primary
    | genvar_expression binary_operator genvar_primary
    | genvar_expression ? genvar_expression : genvar_expression
    | string
genvar_primary ::=
    constant_primary
    | genvar_identifier
    | genvar_identifier [ genvar_expression ]
    | analysis ( arg_list )
genvar_assignment ::=
    genvar_identifier = genvar_expression

```

A.9 Expressions

```

range ::=
    [ constant_expression : constant_expression ]
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 }
attribute_reference ::=
    net_identifier . pot_or_flow . attribute_identifier
    | net_identifier.attribute_identifier
constant_primary ::=
    number
    | parameter_identifier
    | constant_concatenation
constant_array_expression ::=
    { constant_arrayinit_element { , constant_arrayinit_element } }
constant_arrayinit_element ::=
    constant_expression
    | integer_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 )
function_call ::=
    function_identifier ( expression { , expression } )
arg_list ::=
    argument { , argument }
argument ::=
    expression
    | constant_array_expression
constant_array_expression ::=
    { constant_array_init_element { , constant_array_init_element } }
constant_array_init_element ::=
    constant_expression
    | integer_constant_expression { , constant_expression }
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 ]
unary_operator ::=
    + | - | ! | ~
binary_operator ::=
    + | - | * | / | % | == | === | != | !== | && | ||
    | < | <= | > | >= | & | | | ^ | ^~ | ~^ | >> | <<
digital_primary ::=
    primary
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 )
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
concatenation ::=
    { expression { , expression } }
sign ::=
    +
    | -
unsigned_num ::=
    decimal_digit { _ | decimal_digit }
digital_number ::=
    number
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
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
driver_access_function ::=
    driver_update | net_resolution
system_function ::=
    $abstime | $realttime | $temperature | $vt | $bound_step | $discontinuity
    | $driver_count | $driver_state | $driver_strength

```

A.10 General

```

comment ::=
    short_comment
    | long_comment
short_comment ::=
    // comment_text \n
long_comment ::=
    /* comment_text */
comment_text ::=
    { Any_ASCII_character }
string ::=
    " { Any_ASCII_character_except_newline } "
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
white_space ::=
    space
    | tab
    | newline
    | formfeeds

```