Annex A

Syntax

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
    | *digital*_udp_declaration
    | nature_definition
    | discipline_definition
    | connect_specification

A.1.1 Module Declaration

module_declaration ::=
    module_keyword *module*_identifier [*digital*_list_of_ports] **;**
     [module_items]
    endmodule
module_keyword ::=
     module
    | **macromodule**
connectmodule_declaration ::=
    connectmodule module_identifier **(** connectmod_port **,** connectmod_port **) ;**
     [module_items]
    endmodule
connectmod_port ::=
     *connectmod_port*_identifier

A.1.2 UDP Declaration

A.1.3 Nature Definition

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.1.4 Discipline Definition

discipline_declaration ::=
    discipline *discipline*_identifier
     [discipline_descriptions]
    enddiscipline
discipline_descriptions ::=
    discipline_description { discipline_description }
discipline_description ::=
      nature_binding
    | attr_override
    | domain_binding
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.1.5 Connectrule Specification

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.2 Module Declarations

module_item_declaration ::=
parameter_declaration
| *digital*_input_declaration
| d*igital*_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
| d*igital*_reg_declaration
| *digital*_time_declaration
| d*igital*_realtime_declaration
| *digital*_event_declaration
| *digital*_task_declaration

A.2.1 Parameters

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**} }**

A.2.2 Variables

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 **]**

A.2.3 Nets

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 **]**
*digital*_net_declaration ::=
     *digital*_net_declaration
    | **wreal** [list_of_identifiers] **;**

A.2.4 Branches

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

A.2.5 Genvars

genvar_declaration ::=
    genvar list_of_genvar_identifiers **;**
list_of_genvar_identifiers ::=
    *genvar*_identifier { **,** *genvar*_identifier }

A.2.6 Analog Functions

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.2.7 Digital Functions

A.2.8 Digital Tasks

A.3 Module Constructs

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

A.3.1 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.3.2 UDP Instantiations

A.3.3 Gate and Switch Instantiations

A.3.4 Specify Block

A.3.5 initial and Always Blocks

A.3.6 Continuous Assignments

A.3.7 Analog Block

analog_block ::=
analog analog_statement

A.3.8 Default Parameter Overrides

parameter_override ::=
defparam list_of_param_assignments **;**

A.4 Analog Behavioral 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

A.4.1 Null Statements

analog_statement_or_null ::=
analog_statement | **;**
statement_or_null ::=
statement | **;**

A.4.2 Sequential Block

analog_seq_block ::=
begin [**:** *block*_identifier { block_item_declaration }]
{ analog_statement }
end
seq_block ::=
begin [**:** *block*_identifier { block_item_declaration }]
{ statement }
end

A.4.3 Contribution Statement

analog_branch_contribution ::=
bvalue <+ analog_expression **;**

analog_indirect_branch_assignment ::=
bvalue **:** nexpr **==** analog_expression **;**
nexpr ::=
bvalue
| pvalue
| **ddt (** bvalue | pvalue **)**
| **idt (** bvalue | pvalue **)**

A.4.4 Procedural Assignment 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.4.5 Conditional Statement

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

A.4.6 Case statement

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
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

A.4.7 Genvar For Statement

analog_for_statement ::=
for (genvar_assignment **;** genvar_expression **;**
genvar_assignment **)** analog_statement

A.4.8 Event Control Statement

event_control_statement ::=
event_control statement_or_null
event_control ::=
@ *event*_identifier
| **@ (** *event*_expression **)**

A.4.9 System Task Calls

system_task_enable ::=
system_task_name [**(** expression { **,** expression } **)**] **;**
system_task_name ::=
$identifier
Note: The **$** may not be followed by a space.

A.4.10 Loop Statements

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

A.5 Module Expressions

A.5.1 Analog Expressions

analog_expression ::=
expression
| analog_operator **(** analog_operator_arg_list **)**

A.5.2 Genvar Expressions

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.5.3 Constant Expressions

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 }

constant_primary ::=
  number
| *parameter*_identifier
| *constant*_concatenation

A.5.4 Expressions

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 **)**

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.5.5 Digital Expressions

*digital*_primary ::=
    primary
*digital*_number ::=
    number

A.5.6 Nature Attribute Accessing

attribute_reference ::=
    *net*_identifier **.** pot_or_flow **.** attribute_identifier

A.5.7 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 }

A.5.8 Analog Function Calls

function_call ::=
    *function*_identifier **(** expression { **,** expression } **)**
arg_list ::=
    argument { **,** argument}
argument ::=
      expression
    | constant_array_expression

A.5.9 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.5.10 Unary and Binary Operators

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

A.5.11 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.5.12 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

concatenation ::=
{ expression { **,** expression } **}**
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.5.13 Built-in 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**
driver_access_function ::=
driver_update | **net_resolution**

A.5.14 System Function Calls

system_function ::=
$abstime | **$realtime** | **$temperature** | **$vt** | **$bound_step** | **$discontinuity**
| **$driver_count** | **$driver_state** | **$driver_strength**

A.5.15 Range

range ::=
[constant_expression **:** constant_expression **]**

A.5.16 String

string ::=
" { Any_ASCII_character_except_newline } "

A.6 Module Event Expressions

A.6.1 Analog

*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 **)**

A.6.2 Digital

*digital_event*_expression ::=
*digital*_expression
| *event*_identifier
| **posedge** *digital*_expression
| **negedge** *digital*_expression
| event_function
| d*igital*_event_expression **or** *digital*_event_expression

A.7 General

A.7.1 Comments

comment ::=
short_comment
| long_comment

short_comment ::=
// comment_text **n**

long_comment ::=
/* comment_text ***/**

comment_text ::=
{ Any_ASCII_character }

A.7.2 Identifiers

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.7.3 White Space

white_space ::=
  space
| tab
| newline
| formfeeds