Annex A

Formal Syntax

(Normative)

The formal syntax of SystemVerilog is described using Backus-Naur Form (BNF). The conventions used are:

— Keywords and punctuation are in **bold** ~~and enclosed in single quotes~~ text.

— Syntactic categories are named ~~between < >~~ in non-bold text.

— Italics do not form part of the category, so ~~<~~*type*_identifier~~>~~ has the same syntax as ~~<~~identifier~~>~~

— A vertical bar (|) separates alternatives.

— Square brackets ([ ]) enclose optional items.

— Braces ({ }) enclose items which may be repeated zero or more times.

The formal syntax in this Annex includes portions of the IEEE 1364-2001 Verilog standard formal syntax, with the extensions provided by SystemVerilog. This Annex does not include all of the IEEE 1364 standard formal syntax. It only includes the portions required to show the full context of the SystemVerilog extensions.

A.1 Source text

source_text ::= [unit] [precision] {declaration_or_statement}

declaration_or_statement ::=
library_declaration
| include_statement
| config_declaration
| module_declaration
| interface_declaration
| task_declaration
| function_declaration
| udp_declaration
| module_instantiation
| interface_instantiation
| event_declaration
| net_declaration
| data_declaration
| statement

library_declaration ::=
library *library*_identifier file_path_spec **;**

include_statement ::=
include file_path_spec **;**

config_declaration ::=
config *config*_identifier **;** design_statement
{config_rule_statement} **endconfig**

design_statement ::=
design {[*library*_identifier]**.***cell*_identifier}

config_rule_statement ::=
    default liblist_clause
  | cell_clause liblist_clause
  | cell_clause use_clause
  | inst_clause liblist_clause
  | inst_clause use_clause

cell_clause ::= **cell** *library*_identifier **.** *cell*_identifier

inst_clause ::= **instance** name

liblist_clause::= **liblist** {*library*_identifier}

use_clause ::= **use** [*library*_identifier**.**] *cell*_identifier

module_declaration ::=
    module_keyword identifier [parameter_port_list]
      [**(** port_list **)**] **;** [unit] [precision]
      {module_item} **endmodule**
  | module_keyword identifier [parameter_port_list]
      [**(** port_decls **)**] **;** [unit] [precision]
      {non_port_module_item} **endmodule**

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

port_list ::= port { **,** port }

port ::=
    port_expression
  | **.** identifier **(** [port_expression] **)**

port_decls ::= port_declaration { **,** port_declaration }

port_declaration ::=
    attribute_instance port_declaration
  | direction] [port_type] variables
  | **interface** variables
  | **interface .** identifier variables
  | identifier variables
  | identifier **.** identifier variables
  | direction **.** identifier **(** port_expression **)**

direction ::= **input** | **output** | **inout**

port_type ::=
    data_type {const_range}
  | net_type [signing] {~~const_range~~ packed_dimension }
  | **event**

per Peter's e-mail on 10 Nov 2001

port_expression::=
    indexed_identifier
  | **{** indexed_identifier { **,** indexed_identifier } **}**

module_item ::=
    io_declaration
  | non_port_module_item

non_port_module_item ::=
    attribute_instance non_port_module_item
  | module_or_generate_item
  | module_declaration
  | interface_declaration
  | parameter_declaration
  | specparam_declaration
  | specify_block

module_or_generate_item ::=
event_declaration
| module_instantiation
| primitive_instantiation
| interface_or_generate_item
| generated_instantiation

interface_declaration ::=
interface identifier [parameter_port_list] **;**
[**(** port_list **)**] **;** [unit][precision]
{ interface_item }
endinterface [**:** identifier]
| **interface** identifier [parameter_port_list] **;**
[**(** port_decls **)**] **;** [unit][precision]
{ non_port_interface_item }
endinterface [**:** identifier]

extra | ?

interface_item ::=
io_declaration
| non_port_interface_item

non_port_interface_item ::=
attribute_instance non_port_interface_item
| interface_declaration
| parameter_declaration
| specparam_declaration
| interface_or_generate_item
| generated_instantiation

interface_or_generate_item::=
event_declaration
| net_declaration
| interface_instantiation
| data_declaration
| task_declaration
| function_declaration
| modport_declaration
| initial_statement
| always_statement
| continuous_assign

parameter_port_list ::=
**(** parameter_declaration {parameter_declaration} **)**

missing comma?

unit ::= [**timeunit** [time_literal] **;**]

precision ::= [**timeprecision** [time_literal] **;**]

modport_declaration ::= **modport** modport_item{**,** modport_item} **;**

modport_item ::= identifier **(** modport_port{**,** modport_port} **)**

modport_port ::=
[direction] [port_type] identifier
| identifier **.** identifier

A.2 Data, Event and Net declarations

parameter_declaration ::=
parameter [data_type] initial_assignments **;**
| **parameter** [signing] {packed_dimension} initial_assignments **;**
| **parameter type** type_assignments **;**

specparam_declaration ::=
param [data_type] initial_assignments **;**
| param [signing] {packed_dimension} initial_assignments **;**

param ::= **localparam** | **specparam**

net_declaration ::=
net_type [strength] [vector_or_scalar] [signing] {packed_dimension}
[delay_values] vars_or_assigns **;**

event_declaration ::= **event** variables **;**

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

strength ::=
(strength0 **,** strength1 **)**
| **(** strength1**,** strength0 **)**
| **(** charge_strength **)**

strength0 ::= **supply0** | **strong0** | **pull0** | **weak0** | **highz0**

strength1 ::= **supply1** | **strong1** | **pull1** | **weak1** | **highz1**

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

vector_or_scalar ::= **vectored** | **scalared**

delay_values ::=
delay_value
| **# (** delay_value [**,** delay_value [**,** delay_value]] **)**

initial_assignments ::= initial_assignment { **,** initial_assignment }

initial_assignment ::= identifier **=** expression

type_assignments ::= type_assignment { **,** type_assignment }

type_assignment ::= identifier **=** data_type

data_declaration ::=
variable_declaration
| constant_declaration
| type_declaration
| state_declaration

variable_declaration::= [lifetime] data_type vars_or_assigns **;**

lifetime ::= **static** | **automatic**

constant_declaration::= **const** data_type initial_assignments **;**

type_declaration ::=
typedef data_type variable **;**
| **typedef** identifier {**[***constant*_expression**]**} **.** *type*_identifier variable **;**

state_declaration::= **state** { state_list } identifier [delay_or_event] **;**

state_list ::= and_states | or_states

and_states ::= state **and** state { **and** state }

or_states ::= state **,** state { **,** state }

state ::= [**{** state_list **}**] identifier

vars_or_assigns ::= var_or_assign { **,** var_or_assign }

var_or_assign ::= variable [= *constant*_expression]

variables ::= variable { , variable }

variable ::= identifier { unpacked_dimension }

unpacked_dimension ::= packed_dimension

simple_type ::= integer_type | non_integer_type | *type*_identifier

A.3 Tasks and Functions

task_declaration ::=
    task [**automatic**] tf_name **;**
        { task_item_declaration } { statement }
    endtask [**:** identifier]
  | **task** [**automatic**] tf_name **(** task_formals **) ;**
        { data_declaration } { statement }
    endtask [**:** identifier]

task_item_declaration ::=
    direction [data_type] variables **;**
  | data_declaration

task_formals ::= [task_formal {**,** task_formal }]

task_formal ::=
    [**port**] [direction] [data_type] variable
  | **port event** variable

task_prototype ::= **task (** [task_proto_formal {**,** task_proto_formal} **)**

named_task_proto ::= **task** identifier **(** [task_proto_formal {**,** task_proto_formal} **)**

task_proto_formal ::=
    [**port**] direction data_type [variable]
  | **port event** variable

function_declaration ::=
    function [**automatic**] data_type tf_name **;**
        {fn_item_declaration}
        { statement }
    endfunction [**:** identifier **:**]
  | **function** [**automatic**] data_type tf_name (fn_formals) **;**
        {data_declaration}
        { statement }
    endfunction [**:** identifier **:**]

extra : at end?

extra : at end?

fn_item_declaration ::=
    [direction] [data_type] variables **;**
  | data_declaration

tf_name ::= identifier [**.** identifier]

fn_formals ::= [fn_formal { **,** fn_formal }]

fn_formal ::= [direction] [data_type] variable

fn_prototype ::= **function** data_type **(** fn_proto_formals **)**

named_fn_proto::= **function** data_typeidentifier **(** fn_proto_formals **)**

fn_proto_formals ::= [fn_proto_formal { **,** fn_proto_formal }]

fn_proto_formal ::=
    [direction] data_type [variable]
  | variable

A.4 Data Types

data_type ::=
integer_type [signing] {packed_dimension}
| *type*_identifier} {packed_dimension}
| non_integer_type
| **struct** { **{** struct_union_member } **}**
| **union** { **{** struct_union_member } **}**
| **enum {** enum_member **}**
| **void**

integer_type ::= **bit** | **logic** | **reg** | **byte** | **char** | **shortint** | **int** | **longint** | **integer**

non_integer_type ::= **time** | **shortreal** | **real** | $built-in

signing ::= [**signed**] | [**unsigned**]

packed_dimension ::= **[** *constant*_expression : *constant*_expression **]**

struct_union_member ::= data_type variables **;**

enum_member ::=
identifier
| identifier **=** *constant*_expression

A.5 Instantiations

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 /* if in module */
| interface_or_generate_item /* if in interface */

are comments OK in a BNF?

generate_conditional_statement ::=
if' (*constant*_expression **)** generate_item_or_null
[**else** generate_item_or_null]

generate_case_statement ::=
case (*constant*_expression **)**
generate_case_item {generate_case_item}
endcase

generate_case_item ::=
*constant*_expression {**,** *constant*_expression **:** generate_item_or_null

generate_loop_statement ::=
for (genvar_decl_or_assign **;** expression **;** genvar_expr_or_assign **)**
generate_named_block

genvar_decl_or_assign ::= [**genvar**] identifier **=** *constant*_expression

genvar_expr_or_assign ::= unary_expression | operator_assignment

generate_named_block ::=
begin : identifier {generate_item} **end**
| identifier **:** generate_block

generate_block ::= **begin** [**:** identifier] {generate_item} **end**

module_instantiation ::=
*module*_identifier [parameter_values] named_instance {**,** named_instance } **;**

interface_instantiation ::=
*interface*_identifier [parameter_values] named_instance {**,** named_instance } **;**

udp_instantiation ::=
identifier [strength] [delay_values] primitive_instance {**;** primitive_instance} **;**

coma separated?

gate_instantiation ::=
gate [strength] [delay_values] primitive_instance {**;** primitive_instance} **;**

coma separated?

gate ::=
and | **nand** | **or** | **nor** | **xor** | **xnor** | **buf**' | **not** | **bufif0** | **bufif1** | **notif0** | **notif1**
| **cmos** | **rcmos** | **nmos** | **rnmos** | **pmos** | **rpmos**
| **tranif0** | **rtranif0** | **tranif1** | **rtranif1** | **tran** | **rtran**

parameter_values ::=
 # (expression {**,** expression } **)**
| **# (** named_param_val { **,** named_param_val} **)**

named_param_val ::= **.** identifier **(** expression **)**

named_instance ::= identifier {**[** expression **:** expression **]**} [**(** port_connections **)**]

primitive_instance ::= [identifier] {**[** expression **:** expression **]**} [**(** port_connections **)**]

port_connections ::=
[expression] { **,** [expression] }
| named_port_connection { **,** named_port_connection }

named_port_connection ::= **.** identifier **(** expression **)**

A.6 Procedural Statements

initial_statement ::= **initial** statement

always_statement ::= **always** statement

combinational_statement ::= **always_comb** statement

latch_statement ::= **always_latch** statement

ff_statement ::= **always_ff** statement

statement_or_null ::= statement | **;**

"statement" was spelled out in some places, and not in others

statement ::=
blocking_assignment **;**
| non_blocking_assignment **;**
| selection
| loop
| jump
| delay_control statement_or_null
| event_control statement_or_null
| **wait (** expression **)** statement_or_null
| **process** statement
| **disable** name **;**
| sequential_block
| parallel_block
| **–>** *event*_name **;**
| transition_to_state statement_or_null
| expression **;**
| proc_continuous_assign **;**
| identifier **:** statement

selection ::=
[up] **if (** expression **)** statement_or_null
[**else** statement_or_nulll]
| [up] case **(** expression **)** case_item { case_item } **endcase**
| **transition (** name **)** transition { transition } **endtransition**

up ::= **unique** | **priority**

case ::= **case** | **casez** | **casex**

loop ::=
forever statement
| **repeat (** expression **)** statement_or_null
| **while (** expression **)** statement_or_null
| **for (** [declare_or_assign] **;** [expression] **;** [expression_or_assign] **)** statement_or_null
| **do** statement **while (** expression **)**

jump ::=
return [expression] **;**
| **break ;**
| **continue ;**

declare_or_assign ::=
lvalue **=** expression
| data_type identifier **=** expression

declare_or_exp ::=
unary_expression
| data_type identifier

blocking_assignment ::=
operator_assignment
| lvalue **=** delay_or_event expression

operator_assignment ::= lvalue assignment_operator expression

assignment_operator ::= **=** | ***=** | **/=** | **%=** | **+=** | **– =** | **<<=** | **>>=** | **&=** | **^=** | **|=**

non_blocking_assignment ::= lvalue **<=** [delay_or_event] expression

delay_or_event ::=
delay_control
| [**repeat (** expression **)**] event_control

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

transition ::=
state_conditions **:** statement_or_null
| **default** [**:**] statement_or_null

state_conditions ::= state_condition { **,** state_condition }

state_condition ::= *state*_identifier { **and** *state*_identifier }

sequential_block ::= **begin** [**:** identifier] {statement} **end** [**:** identifier]

parallel_block ::= **fork** [**:** identifier] {statement} **join** [**:** identifier]

transition_to_state ::=
–>> *machine*_name **.** *state*_identifier
| **–>>** *machine*_name **. (** state_condition **)**
| [*transition*_identifier] **–>>** state_condition

proc_continuous_assign ::=
assign name_or_names **=** expression
| **deassign** name
| **force** name_or_names **=** expression
| **release** name

A.7 Names

name_or_names ::= name | **{** name { **,** name } **}**

name ::= indexed_identifier { **.** indexed_identifier }

indexed_identifier ::= identifier [**[** *constant*_expression [**:** *constant*_expression] **]**]

A.8 Delay and Event Controls

delay_control ::=
number
| **#** name
| **#** **(** expression **)**

event_control ::=
@ name
| **@** **(** event_expressions **)**
| **@***

also @(*) ?

event_expressions ::=
event_expression { **or** event_expression }
| event_expression { **,** event_expression }

event_expression ::=
[edge] expression [**iff** expression]
| **(** [edge] expression [**iff** expression] **)**

edge ::= **posedge** | **negedge** | **changed**

A.9 Expressions

expression ::=
unary_expression
| expression binary_operator expression
| expression **?** expression **:** expression
| **(** operator_assignment **)**

unary_expression ::= [unary_operator] lvalue

unary_operator ::= **+** | **–** | **!** | **~** | **&** | **~&** | | | **~|** | **^** | **~^** | **^~** | bump_operator

lvalue ::= postfix_expression

postfix_expression ::=
primary [bump_operator]
| postfix_expression **.** identifier
| postfix_expression **->** identifier
| postfix_expression **[** expression **]**
| postfix_expression **[** expression **:** expression **]**
| postfix_expression **(** [expression { **,** expression }] **)**
| postfix_expression bump_operator

bump_operator ::= **+ +** | **– –**

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

primary ::=
identifier
| literal
| data_type
| (expression)
| { expression { , expression } }
| { expression { expression } }
| simple_type '(expression)
| simple_type '{ expression { , expression } }
| simple_type '{ expression { expression } }

A.10 Literals

literal ::=
string_literal
| number
| time_literal
| '0 | '1 | 'z | 'Z | 'x | 'X

string_literal::= " value "

number ::=
integer
| [integer] ' base value
| real

base ::= 'b | 'd | 'h | 'o | 'sb | 'sd | 'sh | 'so

uppercase also?

time_literal ::= integer [. integer] time_unit

time_unit ::= s | ms | us | ns | ps | fs