Appendix 3
Syntax of AleC++

We systematized the syntax of AleC++ in the following text.

Operator	Association
[] () . -> ::	left
~ ! sizeof lengthof - ++ -- (cast) new delete $ @	right
.* ->*	left
* / %	left
+ -	left
<< >>	left
< <= > >=	left
== !=	left
& ~&	left

^ ~^	left
| ~|	left
?:	right
= <- *op*=	right
,	left

Syntax begins from the symbol **global_data**:

```
/*********************************************************************/
global_data:
        global_item
        global_data global_item

global_item:
        external_definition
        module_definition
        implicit_definition
        model_card
        spice_code
        library_specification

/********************** #1.1 Function definition **********************/
nonempty_fpar_list:
        on_line_declaration

on_line_declaration:
        on_line_decl
        on_line_decl ,  ...
        ...

on_line_decl:
        on_line_item
        on_line_decl , on_line_item

on_line_item:
        auto_decl_specifiers   on_line_declarator

on_line_declarator:
        init_declarator
        abstract_declarator   initializer

function_body:
        function_statement

function_statement:
        compound_statement
```

*/********** #1.2 Data definition **********/*
external_definition:
<decl_specifiers> externals
decl_specifiers

externals:
declarator ctor_initializer function_body
declaration_list

*/********** Base constructor initializer **********/*
ctor_initializer:
: member_initializer_list
: (expression_list)

member_initializer_list:
member_initializer
member_initializer_list , member_initializer

member_initializer:
class_name (expression_list)
identifier (expression_list)

*/********** #1.3 Module definition **********/*
module_definition:
module *module_definition_or_decl*
root_module_definition

root_module_definition:
root <**module**> *module_prototype module_body*

module_definition_or_decl:
module_prototype module_body
module_prototype_list ;

module_prototype_list:
module_prototype_item
module_prototype_list , module_prototype_item

module_prototype:
<signal_sc> full_module_name < (<module_interface>) >

full_module_name:
module_name
class_name :: module_name

module_name:
<any_name.>module_id

module_id:
identifier
class_name

module_prototype_item:
module_prototype <actpars>

actpars:
action *(<fpar_list>)*

module_interface:
partial_interface
full_inter_decl
vararg_interface
full_inter_decl ; vararg_interface

partial_interface:
partial_node
partial_interface , partial_node

partial_node:
free_form_node

full_inter_decl:
full_interface
full_inter_decl ; full_interface

full_interface:
arbit_sc <signal_type_specifiers> <direction> full_inter_list

signal_type_specifiers:
auto_decl_specifier
signal_type_specifiers auto_decl_specifier

vararg_interface:
arbit_sc <signal_type_specifiers> <direction> ...

arbit_sc:
signal_sc
type_specifier

full_inter_list:
signal_init_declarator
full_inter_list , signal_init_declarator

signal_declarator:
free_form_node
signal_declarator [array_size]
signal_declarator [**auto** *]*

signal_init_declarator:
signal_declarator <initializer> converters

signal_sc:
signal
node
charge
current
flow

direction:
in
out
inout

converters:
: (<module_name , module_name)
: module_name

free_form_node:
identifier
integer_constant

*/********** #1.3.1 Module body **********/*
module_body:
{ <structural_decl> <component_map>
<conversion_spec> <simulation_spec> <action_decl> }

*/********** #1.3.1.1 Structural declaration **********/*
structural_decl:
local_declaration
structural_decl local_declaration

local_declaration:
signal_declaration
component_declaration

signal_declaration:
arbit_sc <signal_type_specifiers> signal_list ;

signal_list:
signal_init_declarator
signal_list , signal_init_declarator

component_declaration:
component_sc component_list ;

component_list:
one_component
component_list , one_component

component_declarator:
identifier
component_declarator [array_size]

one_component:
component_declarator

component_sc:
module *module_prototype_item*
module_name
builtin_element_type
switch

full_name:
identifier
library_name . identifier

*/********** #1.3.1.2 Component mapping **********/*
component_map_list:
component_map
component_map_list component_map

component_map:
component_name (<actual_signal_list>) parameters

component_name:
any_name
return *module_name*

parameters:
;
constant_expression ;
par_assignment ;
special_assignment <special_assignment> ;
{ parameter_list } <;>
{ pwl_list <;> } <;>

special_assignment:
model_attach
action_positional

action_positional:
action *(<expression_list >)*

actual_signal_list:
actual_signal
actual_signal_list , actual_signal

actual_signal:
static_signal
void

pwl_list:
pwl_pair
pwl_list ; pwl_pair

pwl_pair:
assignment_expression , assignment_expression

*/********** #1.3.1.3 Conversion declaration **********/*
conversion_spec:
conversion *{ parameter_list } <;>*

*/********** #1.3.1.4 Simulation conditions (root module only) *********/*
simulation_spec:
simulation_item
simulation_spec simulation_item

simulation_item:
options_list
timing_list
output_list

options_list:
options *{ parameter_list } <;>*

timing_list:
timing *{ parameter_list } <;>*

output_list:
out *{ output_groups } <;>*

output_groups:
output_group
output_groups output_group

output_group:
arbit_sc <signal_type_specifiers> <direction> output_item_list ;
caption *string_constant ;*
sweep *arbit_sc <signal_type_specifiers> output_item ;*

output_item_list:
output_item
output_item_list , output_item

output_item:
<path_sc /> static_signal <conv_select>
identifier compound_statement

path_sc:
path_level
path_sc / path_level

path_level:
identifier
string_constant

conv_select:
(identifier)

*/********** #1.3.1.5 Action declaration **********/*
action_decl:
action *<update> generic action_body*

update:
structural
post_structurual
initial
per_moment
post_moment
per_iteration
final

generic :
(<fpar_list>)

action_body:
{ <action_context> }

action_context:
action_statements

action_statements:
action_statement
action_statements action_statement

action_statement:
statement
process_statement

process_statement:
process_header process_update compound_statement

process_header:
<identifier :> **process**

process_update:
<**virtual**> *update*

```
        ( sensitivity_list )

sensitivity_list:  sensitive_signal
        sensitivity_list  ,  sensitive_signal
        sensitivity_list  ...

sensitive_signal:
        static_signal

static_signal:
        free_form_node
        $  constant_expression
        :: identifier
        ( static_signal )
        static_signal  [ constant_expression ]
        static_signal [ constant_expression : <constant_expression> ]
        static_signal  .  identifier

/********************** #1.5   Implicit definition ************************/
implicit_definition:
        implicit { implicit_context } <;>

implicit_context:
        implicit_list
        implicit_context implicit_list

implicit_list:
        component_sc short_list  ;

short_list:
        identifier
        short_list ,  identifier

/********************** #1.6    Library specification **********************/
library_specification:
        library library_list  ;

library_list:
        library_item
        library_list , library_item

library_item:
        any_name

/********************** #2.1    Declarations ***************************/
declaration:
        decl_specifiers  <init_declarator_list>;

decl_specifiers:
        decl_spec_list
```

decl_spec_list:
first_decl_specifier
decl_spec_list decl_specifier

auto_decl_specifiers:
auto_decl_spec_list

auto_decl_spec_list:
type_specifier
auto_decl_spec_list auto_decl_specifier

decl_specifier:
first_decl_specifier
bus_resolution_specifier
attribute_specifier

first_decl_specifier:
type_specifier
sc_specifier
signal_sc
fct_specifier
friend

auto_decl_specifier:
type_specifier
bus_resolution_specifier
attribute_specifier

length_specifier:
long
short
signed
unsigned

cv_qualifier:
const
volatile

sc_specifier:
auto
extern
static
typedef
register

fct_specifier:
inline
virtual

bus_resolution_specifier:
: function_name

attribute_specifier:
@ signal_attribute_sc

signal_attribute_sc:
simple_type_name <(<expression_list >) >

type_specifier:
simple_type_name
struct_union_specifier
enum_specifier
cv_qualifier

basic_types:
void
char
int
float
double

simple_type_name:
basic_types
length_specifier
class_enum_name
typedef_name

*/********** #2.2 Declarators **********/*
*/********** #2.2.1 Names **********/*
declarator_name:
identifier
operator_name
qualified_name

qualified_name:
class_name :: operator_name
class_name :: conversion_fct_name
class_name :: identifier

*/********** #2.2.2 Lists **********/*
init_declarator_list:
init_declarator
init_declarator_list , init_declarator

*/********** #2.2.3 Items **********/*
declarator_item:
declarator

ptr_operator:
** <cv_qualifier>*
& <cv_qualifier>
*class_name :: * <cv_qualifier>*
class_name :: & <cv_qualifier>

declarator:
primary_name
qualified_name
class_name
~ class_name
> class_name
ptr_operator declarator
(declarator)
declarator [array_size]
declarator (<on_line_declaration>) <cv_qualifier>

init_declarator:
declarator <initializer>

operator_name:
operator *opname*

opname:
(svioperatori osim $ $$, . ->* <- ?: @)*

conversion_function_name:
operator *simple_type_name <ptr_operator>*

*/********************** #2.3 Structures/unions *************************/*
struct_union_specifier:
struct_header struct_decl_list }
struct_keyword identifier
struct_keyword tag_name
union_header union_decl_list }
union *tag_name*

struct_keyword:
struct
class

*/********************** #2.3.1 Class/structure header **********************/*
struct_header:
struct_keyword <tag_name> <base_spec> {

base_spec:
: base_list

base_list:
base_specifier

base_list , base_specifier

base_specifier:
*<base_access> class_name <***virtual***>*
*<base_access> <***virtual***> class_name*

base_access:
public
private
protected

*/*********************#2.3.2 Union declaration *********************/*
union_header:
union *<tag_name> {*

union_decl_list:
union_declaration
union_decl_list union_declaration

union_declaration:
auto_decl_specifiers union_declarator_list ;

union_declarator_list:
union_declarator
union_declarator_list , union_declarator

union_declarator:
declarator_item

struct_decl_list:
struct_declaration
struct_decl_list struct_declaration

struct_declaration:
<access_specifier> < decl_specifiers> <struct_list>;
<access_specifier> declarator <ctor_initializer> function_body <;>
<access_specifier> **friend module** *friend_module_list ;*

friend_module_list:
friend_module_name
friend_module_list , friend_module_name

access_specifier:
base_access :

*/********************* #2.4 Initialization ********************/*
initializer:
= init_expr
<- init_expr
(expression_list)

init_expr:
constant_expression
{ }
{ initializer_list }
{ initializer_list , }

initializer_list:
constant_expression
initializer_list , initializer_list
{ initializer_list }

*/********** #2.5 Names for type conversion **********/*
type_specifiers:
type_specifier
type_specifiers type_specifier

type_name:
type_specifiers <abstract_declarator>

restricted_type_name:
type_specifier <restricted_declarator>

abstract_declarator:
ptr_operator abstract_declarator
abstract_declarator (< on_line_declaration >)
abstract_declarator [array_size]
(abstract_declarator)

restricted_declarator:
ptr_operator restricted_declarator
restricted_declarator restricted_array

restricted_array:
[expression]

typedef_name:
new_type (expression_list)

*/********** #2.6 Enumeration type **********/*

enum_specifier:
enum *<tag_name> { enum_list }*
enum *tag_name*

enum_list:
enumerator
enum_list , enumerator

enumerator:

```
        enum_symbol
        enum_symbol = constant_expression
        enum_symbol = void

enum_symbol:
        identifier
        character_constant

/*************************** #2.7   Array size ************************/
array_size:
        <constant_expression>
        constant_expression : constant_expression

/*************************** #3   Expressions*************************/
/*************************** #3.1   Literals ***************************/
constant:
        integer_constant <identifier>
        double_constant  <identifier>
        character_constant

string:
        string_cat

string_cat:
        string_constant
        string_cat string_constant

any_name:
        string_constant
        identifier

asgnop:
        =    +=    -=    *=    /=    %=    &=    |=    ^=    >>=    <<=

primary_name:
        identifier
        operator_name

/************************ 3.2   Primary  expression *********************/
primary_expression:
        basic_name
        constant
        string
        this
        :: primary_name
        ( expression )
        now
        ( signal_sc) free_form_node
        $$
```

```
/************************ 3.3    Postfix expression ************************/
postfix_expression:
        primary_expression
        postfix_expression ( <expression_list> )
        postfix_expression [ expression ]
        simple_type_name ( expression_list )
        postfix_expression ++
        postfix_expression --
        postfix_expression . basic_name
        postfix_expression -> basic_name

/************************3.4    Expression list ****************************/
expression_list:
        assignment_expression
        expression_list , assignment_expression

/************************3.5    Unary expression ************************/
unary_expression:
        postfix_expression
        ++ unary_expression
        -- unary_expression
        * cast_expression
        &  cast_expression
        - cast_expression
        + cast_expression
        ! cast_expression
        ~  cast_expression
        sizeof  ( type_name )
        sizeof  unary_expression
        lengthof  unary_expression
        allocator
        deallocator
        @ cast_expression
        $  cast_expression

allocator:
        <::> new  <(expression_list)> (type_name )
        <::> new  <(expression_list)> restricted_type_name  (expression_list)

deallocator:
        <::> delete <[expression]>cast_expression

/********************** 3.6    Cast expression ****************************/
cast_expression:
        unary_expression
        ( type_name ) cast_expression

/************************ 3.7 PM-expression ****************************/
pm_expression:
        cast_expression
```

```
        pm_expression  .* cast_expression
        pm_expression  ->* cast_expression

/********************* 3.8   Multiplicative expression **********************/
multiplicative_expression:
        pm_expression
        multiplicative_expression *  pm_expression
        multiplicative_expression /  pm_expression
        multiplicative_expression % pm_expression

/*********************       3.9   Additive  expression ***********************/
additive_expression:
        multiplicative_expression
        additive_expression  +  multiplicative_expression
        additive_expression  -  multiplicative_expression

/********************* 3.10   Shift  expression ****************************/
shift_expression:
        additive_expression
        shift_expression  <<  additive_expression
        shift_expression  >> additive_expression

/********************** 3.11   Relational  expression ***********************/
relational_expression:
        shift_expression
        relational_expression  < shift_expression
        relational_expression  <= shift_expression
        relational_expression  > shift_expression
        relational_expression  >= shift_expression

/********************** 3.12   Equality  expression ************************/
equality_expression:
        relational_expression
        equality_expression  ==  relational_expression
        equality_expression  !=  relational_expression

/********************** 3.13 AND  expression ******************************/
AND_expression:
        equality_expression
        AND_expression  &  equality_expression
        AND_expression ~& equality_expression

/********************** 3.14   Exclusive-OR expression **********************/
exclusive_OR_expression:
        AND_expression
        exclusive_OR_expression  ^ AND_expression
        exclusive_OR_expression  ~^ AND_expression

/********************** 3.15   Inclusive-OR  expression *********************/
inclusive_OR_expression:
```

```
        exclusive_OR_expression
        inclusive_OR_expression | exclusive_OR_expression
        inclusive_OR_expression  ~| exclusive_OR_expression

/********************** 3.16  Logical  AND  expression *******************/
logical_AND_expression:
        inclusive_OR_expression
        logical_AND_expression  &&  inclusive_OR_expression

/********************** 3.17  Logical  OR  expression ********************/
logical_OR_expression:
        logical_AND_expression
        logical_OR_expression || logical_AND_expression

/********************** 3.18  Conditional  expression ********************/
conditional_expression:
        logical_OR_expression
        logical_OR_expression  ?  expression : conditional_expression

/**********************    3.19    Assignment  expression *******************/
assignment_expression:
        conditional_expression
        unary_expression  asgnop  assignment_expression

/************************** 3.20  Expression ****************************/
expression:
        assignment_expression
        expression ,  assignment_expression

/********************** 3.22  Constant  expression **********************/
constant_expression:
        conditional_expression

/****************************#4 Statements ******************************/
compound_statement:
        { <statement_list> }

statement_list:
        statement
        statement_list statement

statement:
        compound_statement
        simple_statement
        labeled_statement
        declaration
        asm_statement

simple_statement:
        expression  ;
```

if_statement
while_statement
do_statement
for_statement
switch_statement
break_statement
continue_statement
return_statement
goto_statement
alecsis_statement
;

if_statement:
if *(expression) statement*
if *(expression) statement else statement*

while_statement:
while *(expression) statement*

do_statement:
do *statement* **while** *(expression) ;*

for_statement:
for *(<for_expression> <expression>; <expression>) statement ;*

for_expression:
<expression >;
<declaration>;

switch_statement:
switch *(expression) { <switch_body> }*

switch_body:
switch_groups

switch_groups:
switch_group
switch_groups switch_group

switch_group:
switch_items statement_list
default_item

switch_items:
switch_item
switch_items switch_item

switch_item:
case *constant_expression :*

default_item:
default *: statement_list*

break_statement:
break *;*

continue_statement:
continue *;*

return_statement:
return *;*
return *expression ;*

goto_statement:
goto *Identifier ;*

labeled_statement:
identifier : simple_statement

asm_statement:
asm *asm_code*

asm_code:
asm_line ;
{ asm_lines }

asm_lines:
asm_line
asm_line s asm_sep asm_line

asm_sep :
newline
;

asm_line:
<identifier :> nolab_asm_line

nolab_asm_line:
fixer asm_instr <.character_constant> <asm_operand<,asm_operand>>

fixed_instr:
!
volatile

asm_operand:
logical_OR_expression
% register_name
(% register_name)

*/********** 4.1 Specific Alecsis statements **********/*
alecsis_statement:
matrix_fillin_statement
clone_statement
signal_assign_statement
wait_statement
nlgen_statement
allocate_statement

*/********** 4.1.1 matrix fillin statement **********/*
matrix_fillin_statement:
eqn *any_equation_statement*

any_equation_statement:
simple_equation_statement
through_equation_statement
across_equation_statement

*/********** 4.1.1.1 simple equation statement **********/*
simple_equation_statement:
matrix_column : fillin_list = constant_expression ;

fillin_list:
matrix_entry
fillin_list + matrix_entry
fillin_list - matrix_entry

matrix_entry:
*multiplicative_expression * ddt matrix_column_pair*
ddt matrix_column_pair
- ddt matrix_column_pair
+ ddt matrix_column_pair
multiplicative_expression

ddt:
ddt
idt
dtdt2

matrix_column_pair:
{ static_signal } < . identifier>
{ static_signal , static_signal } < . identifier>

*/********** 4.1.1.2 through equation statement **********/*
through_equation_statement:
matrix_column_pair = fillin_list ;

*/********** 4.1.1.3 across equation statement **********/*
across_equation_statement:

matrix_column , matrix_column_pair = fillin_list ;

*/********** 4.1.2 Signal assignment **********/*
signal_assign_statement:
*postfix_expression <- <***transport***> ass_value ;*

ass_value:
assignment_expression
delay_list

delay_list:
assignment_expression **after** *constant_expression*
delay_list , assignment_expression **after** *constant_expression*

*/********** 4.1.3 Wait statement **********/*
wait_statement:
wait *<sensitivity_list> condition_clause timeout_clause ;*

condition_clause:
*<***while** *expression >*

timeout_clause:
*<***for** *expression>*

*/********** 4.1.4 Clone statement **********/*
clone_statement:
clone *clone_set ;*

clone_set:
one_clone_el
{ clone_list }

clone_list:
one_clone_el
clone_list one_clone_el

one_clone_el:
identifier <[constant_expression]> (<actual_signal_list>) parameters

*/********** 4.1.5 Nlgen statement **********/*
nlgen_statement:
nlgen_key identifier = conditional_expression <{ partial_derivative_list }> ;

nlgen_key:
nlcgen
nlvgen
nlgen

partial_derivative_list:
partial_derivative
partial_derivative_list partial_derivative

partial_derivative:
@ static_signal = conditional_expression ;

*/************************ 4.1.6 allocate_statement ************************/*
allocate_statement:
allocate *allocate_list ;*

allocate_list:
allocate_node
allocate_list , allocate_node

allocate_node:
identifier [constant_expression]

*/************************ 5.0 Model card definition ************************/*
model_card:
model_header { <model_body> } <;>

model_header:
model *<class_name ::> full_name model_class_name <(<expression_list>)>*

model_body:
model_parameter
model_body model_parameter

model_parameter:
model_assignment ;

model_assignment:
model_lhs model_rhs

model_lhs:
<class_name ::> identifier
model_lhs [constant_expression]
model_lhs . identifier

model_rhs:
= initializer
= model_assignment

*/************************ #6.0 General-purpose parameter list ************/*
parameter_list:
general_parameter

parameter_list general_parameter

general_parameter:
par_assignment ;
model_attach ;
template_attach ;

par_assignment:
identifier = par_rvalue

model_attach:
<private> **model** *= full_name*

par_rvalue:
expression
par_assignment

*/********** #8.0 SPICE code **********/*
spice_code:
spice *{ sp_lines }*

sp_lines:
sp_line
sp_lines sp_line

sp_line:
** line_of_any_text*
newline
.**model** *model_name model_class <sp_sets> newline*
+ sp_sets newline

sp_sets:
spice_assign
sp_sets spice_assign

spice_assign:
identifier = spice_constant

spice_constant:
double_constant
- double_constant
+ double_constant