

```

// =====
// SimilarHDL
// =====

SimilarHDL =
    [ library dummy_ident ';' ]
    [ use dummy_ident { '.' dummy_ident } ';' ]
    ( entity_declaration
      | architecture_body )

// =====
// Entity
// =====

entity_declaration =
    entity ident is
        [ port_clause ]
    end [ entity ] [ident] ';'

port_clause =
    port '(' port_interface_element { ';' port_interface_element } ')' ';'

port_interface_element =
    decl_ident_list ':' [mode] similar_port_subtype_indication [ "!=" expression ]

decl_ident_list =
    ident { ',' ident }

mode = in | out

similar_port_subtype_indication =
    "std_ulogic"
    | "std_ulogic_vector"
    | ( "std_ulogic_vector" similar_index_constraint )

similar_index_constraint =
    '(' discrete_range ')'

```

```

// =====
// Architecture
// =====

architecture_body =
    architecture ident of ident is
        { architecture_declarative_item }
    begin
        { [ ident ":" ] process_statement }
    end [ architecture ] [ident] ';'

architecture_declarative_item =
    type_declaration
    | subtype_declaration
    | signal_declaration
    | constant_declaration
    | shared_variable_declaration
    | subprogram_declaration

process_statement =
    process [ '(' sensitivity_list' )' ] [ is ]
        { process_declarative_item }
    begin
        { seq_stat }
    end process [ ident ] ';'

sensitivity_list =
    name { ',' name }

condition =
    boolean_expression

process_declarative_item =
    type_declaration
    | subtype_declaration
    | constant_declaration
    | variable_declaration

```

```

// =====
// Declarations
// =====

// --- Type Declaration -----

type_declaration =
    type ident is type_definition ';'

type_definition =
    // composite_types (array, record) are not yet implemented
    scalar_type_definition ';'

scalar_type_definition =
    enumeration_type_definition | integer_type_definition

enumeration_type_definition =
    '(' ident { ',' ident }

integer_type_definition =
    range_constraint

// --- Subtype Declaration -----

subtype_declaration =
    subtype ident is subtype_indication ';'

subtype_indication =
    ident [ similar_index_constraint | range_constraint ]

// --- Constant, Variable and Signal Declaration -----

variable_declaration =
    variable var_const_sig_declaration

shared_variable_declaration =
    shared variable var_const_sig_declaration

constant_declaration =
    constant var_const_sig_declaration

signal_declaration =
    signal var_const_sig_declaration

var_const_sig_declaration =
    decl_ident_list ':' subtype_indication [ "!=" expression ] ';'

```

```

// --- Subprogram_Declaration -----

subprogram_declaration =
    function_definition

function_definition =
    [ pur | impure ] function ident [ '(' function_parameter_interface_list ')' ]
    return type_mark is
    { subprogram_declarative_item }
    begin
    { seq_stat }
    end [ function ] [ ident ] ';'

function_parameter_interface_list =
    function_parameter_interface_element
    { ';' function_parameter_interface_element }

function_parameter_interface_element =
    [ variable | signal | constant ] function_var_const_interface_declaration

function_var_const_interface_declaration =
    decl_ident_list ':' [ in ] subtype_indication [ "<=" expression ]

```

```

// =====
// Sequential Statements
// =====

seq_stat =
[ ident ':' ]
  assignment_statement
| if_statement
| loop_statement
| case_statement
| return_statement
| wait_statement
| null_statement
| assertion_statement      // Modified in Similar
| report_statement        // Modified in Similar
| print_statement          // Specific in Similar

// --- assign -----

assignment_statement =
  name
  ( "!=" expression )
| ( "<=" [ delay_mechanism ] waveform )
';'

delay_mechanism =
  transport
| ( [ reject time_expression ] inertial )

waveform =
  waveform_element { ',' waveform_element }

waveform_element =
  value_expression [ after time_expression ]

// --- if -----

if_statement =
  if condition then
    seq_stat { seq_stat }
  { elsif condition then
    seq_stat { seq_stat } }
  [ else
    seq_stat { seq_stat } ]
  end if [ ident ]';'

condition =
  boolean_expression

```

```

// --- loop -----
loop_statement =
    for_statement | while_statement

for_statement =
    for ident in discrete_range loop
        seq_stat { seq_stat }
    end loop [ ident ] ';'

while_statement =
    while condition loop
        seq_stat{ seq_stat }
    end loop [ ident ] ';'

// --- case -----
case_statement =
    case expression is
        { when case_choices "=>" { seq_stat } }
    end case [ ident ] ';'

case_choices =
    case_choice { '|' case_choice }

case_choice =
    discrete_range | simple_expression | "others"

// --- wait -----
return_statement =
    return expression ';'

// --- wait -----
wait_statement =
    wait [ sensitivity_clause ] [ condition_clause ] [ timeout_clause ] ';'

sensitivity_clause =
    on sensitivity_list

condition_clause =
    until condition

timeout_clause =
    for time_expression

// --- null -----
null_statement =
    null ';'

```

```

// =====
// Expression
// =====

expression = logical_expression

time_expression = simple_expression

value_expression = expression

boolean_expression = expression

logical_expression =
    (relation
        [ ( and relation { and relation } )
          | ( or relation { or relation } )
          | ( xor relation { xor relation } )
          | ( xnor relation { xnor relation } )
          | ( nand relation )
          | ( nor relation )
        ]
    )

relation =
    (shift_expression [ relational_op shift_expression ]

shift_expression =
    simple_expression [ shift_op simple_expression ]

simple_expression =
    [ '+' | '-' ] term { adding_op term }

term =
    factor { multiplying_op factor }

factor =
    ( primary [ "*" primary ] )
    | abs primary
    | not primary
    | ( logic_red primary )

```

```

// =====
// Primaries
// =====

primary =
    name
    | function_call
    | aggregate
    | '(' expression ')'
    | literal
    | similar_builtin

// --- Names -----

name =
    ident
    [ object_range<sDesc> { object_range } ]
    [ attribute_designator ]

// --- Function Call -----

function_call =
    ident [ '(' function_parameter_association_list ')' ]

function_parameter_association_list =
    function_parameter_association_element
    { ',' function_parameter_association_element }

function_parameter_association_element =
    [ ident ">" selector ] expression

// --- Literals -----

literal =
    numeric_literal | string_literal | enumeration_literal | null

string_literal =
    bin_string_literal
    | oct_string_literal
    | hex_string_literal
    | char_string_literal

enumeration_literal =
    'U' | 'X' | '0' | '1' | 'Z' | 'W' | 'L' | 'H' | '-' | true | false

// --- Similar Builtins -----

similar_builtin =
    (to_bit_vector | to_std_ulogic_vector)
    '(' simple_expression ',' simple_expression ',' expression ')'
    | to_integer '(' expression ',' expression ')'
    | now [ '(' ')' ]

```



```

// =====
// Ranges
// =====

discrete_range =
    range_attribute
    | ident [ range_constraint ]
    | ( simple_expression range_direction simple_expression )

range_constraint =
    range
    range_attribute | ( simple_expression range_direction simple_expression )

range_attribute =
    ident range_attribute_designator

range_attribute_designator =
    'range' | 'reverse_range'

range_direction =
    to | downto

// =====
// Attributes
// =====

attribute_designator =
    'event'
    | 'left'
    | 'right'
    | 'low'
    | 'high'
    | 'length'
    | 'ascending'

// =====
// Operators
// =====

multiplying_op =
    '*' | '/' | mod | rem

adding_op =
    '+' | '-' | '&'

relational_op =
    '=' | '/=' | '<' | "<=" | '>' | ">="

shift_op<int &op> =
    sll | srl | sla | sra | rol | ror

logic_red =
    and | or | xor | xnor | nand | nor
// =====

```

```

// Identifier
// =====

letter      = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
digit       = "0123456789"
letter_or_digit = letter + digit

ident = letter { ['_'] letter_or_digit }

// =====
// Similar Specifics
// =====

assertion_statement =
    assert condition report print_expressions ';'

report_statement =
    report print_expressions ';'

print_statement =
    ( "println" | "print" ) '(' print_expressions ')' ';'

print_expressions =
    printExpr { ',', printExpr }

printExpr =
    scalar_value_name | char_string_literal

// =====

```