Clause 6

Names

The rules applicable to the various forms of name are described in this ~~section~~ clause[1].

6.1 Names

Names can denote declared entities, whether declared explicitly or implicitly. Names can also denote

— Objects denoted by access values,

— Methods (see 3.5.1) of protected types,

— Subelements of composite objects,

— Subelements of composite values,

— Slices of composite objects,

— Slices of composite values, and

— Attributes of any named entity.

```
name ::=
        simple_name
      | operator_symbol
      | selected_name
      | indexed_name
      | slice_name
      | attribute_name

prefix ::=
        name
      | function_call
```

Certain forms of names[2] (indexed and selected names, ~~slices~~ slice names[3], and attribute names) include a *prefix* that is a name or a function call. If the prefix of a name is a function call, then the name denotes an element, a slice, or an attribute, either of the result of the function call, or (if the result is an access value) of the object designated by the result. Function calls are defined in 7.3.3.

If the type of a prefix is an access type, then the prefix must not be a name that denotes a formal parameter of mode **out** or a subelement thereof.

1. To conform to IEEE rules.
2. Ernst Christen's review of D1.
3. IR1000.2.5.

A prefix is said to be *appropriate* for a type in either of the following cases:

— The type of the prefix is the type considered.

— The type of the prefix is an access type whose designated type is the type considered.

The evaluation of a name determines the named entity denoted by the name. The evaluation of a name that has a prefix includes the evaluation of the prefix, that is, of the corresponding name or function call. If the type of the prefix is an access type, the evaluation of the prefix includes the determination of the object designated by the corresponding access value. In such a case, it is an error if the value of the prefix is a null access value. It is an error if, after all type analysis (including overload resolution) the name is ambiguous.

A name is said to be a *static name* if and only if one of the following conditions holds:

— The name is a simple name or selected name (including those that are expanded names) that does not denote a function call, an object or value of an access type, or an object of a protected type and (in the case of a selected name) whose prefix is a static name.

— The name is an indexed name whose prefix is a static name, and every expression that appears as part of the name is a static expression.

— The name is a slice name whose prefix is a static name and whose discrete range is a static discrete range.

— The name is an attribute name whose prefix is a static signal name and whose suffix is one of the predefined attributes 'DELAYED, 'STABLE, 'QUIET, or 'TRANSACTION.[4]

Furthermore, a name is said to be a *locally static name* if and only if one of the following conditions hold:

— The name is a simple name or selected name (including those that are expanded names) that is not an alias and that does not denote a function call, an object or value of an access type, or an object of a protected type and (in the case of a selected name) whose prefix is a locally static name.

— The name is a simple name or selected name (including those that are expanded names) that is an alias, and that the aliased name given in the corresponding alias declaration (see 4.3.3) is a locally static name, and (in the case of a selected name) whose prefix is a locally static name.

— The name is an indexed name whose prefix is a locally static name, and every expression that appears as part of the name is a locally static expression.

— The name is a slice name whose prefix is a locally static name and whose discrete range is a locally static discrete range.

A *static signal name* is a static name that denotes a signal. The *longest static prefix* of a signal name is the name itself, if the name is a static signal name; otherwise, it is the longest prefix of the name that is a static signal name. Similarly, a *static variable name* is a static name that denotes a variable, and the longest static prefix of a variable name is the name itself, if the name is a static variable name; otherwise, it is the longest prefix of the name that is a static variable name.

Examples:

```
S(C,2)          -- A static name: C is a static constant.
R(J to 16)      -- A nonstatic name: J is a signal.
                -- R is the longest static prefix of R(J to 16).
```

4. LCS 20.

```
T(n)        -- A static name; n is a generic constant.
T(2)        -- A locally static name.
```

6.2 Simple names

A simple name for a named entity is either the identifier associated with the entity by its declaration, or another identifier associated with the entity by an alias declaration. In particular, the simple name for an entity ~~interface~~ declaration[5], a configuration, a package, a procedure, or a function is the identifier that appears in the corresponding entity declaration, configuration declaration, package declaration, procedure declaration, or function declaration, respectively. The simple name of an architecture is that defined by the identifier of the architecture body.

simple_name ::= identifier

The evaluation of a simple name has no other effect than to determine the named entity denoted by the name.

6.3 Selected names

A selected name is used to denote a named entity whose declaration appears either within the declaration of another named entity or within a design library.

selected_name ::= prefix . suffix

suffix ::=
    simple_name
  | character_literal
  | operator_symbol
  | **all**

A selected name ~~may be used to~~ can[6] denote an element of a record, an object designated by an access value, or a named entity whose declaration is contained within another named entity, particularly within a library, a package, or a protected type. Furthermore, a selected name ~~may be used to~~ can[7] denote all named entities whose declarations are contained within a library or a package.

For a selected name that is used to denote a record element, the suffix must be a simple name denoting an element of a record object or value. The prefix must be appropriate for the type of this object or value.

For a selected name that is used to denote the object designated by an access value, the suffix must be the reserved word **all**. The prefix must belong to an access type.

The remaining forms of selected names are called *expanded names*. The prefix of an expanded name ~~may~~ must[8] not be a function call.

An expanded name denotes a primary unit contained in a design library if the prefix denotes the library and the suffix is the simple name of a primary unit whose declaration is contained in that library. An expanded name denotes all primary units contained in a library if the prefix denotes the library and the suffix is the reserved word **all**. ~~An expanded name is not allowed for a secondary unit, particularly for an architecture body.~~[9]

An expanded name denotes a named entity declared in a package if the prefix denotes the package and the suffix is the simple name, character literal, or operator symbol of a named entity whose declaration occurs immediately within that package. An expanded name denotes all named entities declared in a package if the prefix denotes the package and the suffix is the reserved word **all**.

---

5. Terminological correction; IR1000.2.6.
6. IR1000.4.7.
7. IR1000.4.7.
8. IR1000.4.7.
9. LCS 3.

An expanded name denotes a named entity declared immediately within a named construct if the prefix denotes a construct that is an entity ~~interface~~ declaration[10], an architecture body[11], a subprogram declaration, a subprogram body[12], a block statement, a process statement, a generate statement, or a loop statement, and the suffix is the simple name, character literal, or operator symbol of a named entity whose declaration occurs immediately within that construct. This form of expanded name is only allowed within the construct itself.

An expanded name denotes a named entity declared immediately within a protected type if the prefix denotes an object of a protected type and the suffix is a simple name of a method whose declaration appears immediately within the protected type declaration.

If, according to the visibility rules, there is at least one possible interpretation of the prefix of a selected name as the name of an enclosing entity ~~interface~~ declaration[13], architecture, subprogram, block statement, process statement, generate statement, or loop statement, or if there is at least one possible interpretation of the prefix of a selected name as the name of an object of a protected type, then the only interpretations considered are those of the immediately preceding two paragraphs. In this case, the selected name is always interpreted as an expanded name. In particular, no interpretations of the prefix as a function call are considered.

Examples:

```
-- Given the following declarations:

type INSTR_TYPE is
    record
        OPCODE:OPCODE_TYPE;
    end record;
signal INSTRUCTION: INSTR_TYPE;

-- The name "INSTRUCTION.OPCODE" is the name of a record element.

-- Given the following declarations:

type INSTR_PTR is access INSTR_TYPE;
variable PTR: INSTR_PTR;

-- The name "PTR.all" is the name of the object designated by PTR.

-- Given the following library clause:

library TTL, CMOS;

-- The name "TTL.SN74LS221" is the name of a design unit contained in a library
-- and the name "CMOS.all" denotes all design units contained in a library.

-- Given the following declaration and use clause:

library MKS;
use MKS.MEASUREMENTS, STD.STANDARD;

-- The name "MEASUREMENTS.VOLTAGE" denotes a named entity declared in a
-- package and the name "STANDARD.all" denotes all named entities declared in a
-- package.
```

10. Terminological correction.
11. IR1000.2.6.
12. IR1000.2.6.
13. Terminological correction.

```
-- Given the following process label and declarative part:

P: process
     variable DATA: INTEGER;
begin

-- Within process P, the name "P.DATA" denotes a named entity declared in process P.

end process;

counter.increment(5);                    -- See 4.3.1.3 for the definition of "counter"
counter.decrement(i);
if counter.value = 0 then … end if;

result.add(sv1, sv2);                    -- See 4.3.1.3 for the definition of "result"

bit_stack.add_bit(1, '1');               -- See 4.3.1.3 for the definition of "bit_stack"
bit_stack.add_bit(2, '1');
bit_stack.add_bit(3, '0');
```

NOTES

1—The object denoted by an access value is accessed differently depending on whether the entire object or a subelement of the object is desired. If the entire object is desired, a selected name whose prefix denotes the access value and whose suffix is the reserved word **all** is used. In this case, the access value is not automatically dereferenced, since it is necessary to distinguish an access value from the object denoted by an access value.

If a subelement of the object is desired, a selected name whose prefix denotes the access value is again used; however, the suffix in this case denotes the subelement. In this case, the access value is automatically dereferenced.

These two cases are shown in the following example:

```
type rec;

type recptr is access rec;

type rec is
     record
          value: INTEGER;
          \next\: recptr;
     end record;

variable list1, list2: recptr;
variable recobj: rec;

list2 := list1;                  -- Access values are copied;
                                 -- list1 and list2 now denote the same object.
list2 := list1.\next\;           -- list2 denotes the same object as list1.\next\.
                                 -- list1.\next\ is the same as list1.all.\next\.
                                 -- An implicit dereference of the access value occurs before the
                                 -- "\next\" field is selected.
recobj := list2.all;             -- An explicit dereference is needed here.
```

2—Overload resolution ~~may be~~ is[14] used to disambiguate selected names. See rules 1 and 3 of 10.5.

3—If, according to the rules of this clause and of 10.5, there is not exactly one interpretation of a selected name that satisfies these rules, then the selected name is ambiguous.

14. IR1000.4.7.

6.4 Indexed names

An indexed name denotes an element of an array.

indexed_name ::= prefix (expression { , expression })

The prefix of an indexed name must be appropriate for an array type. The expressions specify the index values for the element; there must be one such expression for each index position of the array, and each expression must be of the type of the corresponding index. For the evaluation of an indexed name, the prefix and the expressions are evaluated. It is an error if an index value does not belong to the range of the corresponding index range of the array.

Examples:

```
REGISTER_ARRAY(5)          -- An element of a one-dimensional array.
MEMORY_CELL(1024,7)        -- An element of a two-dimensional array.
```

NOTE

—If a name (including one used as a prefix) has an interpretation both as an indexed name and as a function call, then the innermost complete context is used to disambiguate the name. If, after applying this rule, there is not exactly one interpretation of the name, then the name is ambiguous. See 10.5.

6.5 Slice names

A slice name denotes a one-dimensional array composed of a sequence of consecutive elements of another one-dimensional array. A slice of a signal is a signal; a slice of a variable is a variable; a slice of a constant is a constant; a slice of a value is a value.

slice_name ::= prefix (discrete_range)

The prefix of a slice must be appropriate for a one-dimensional array object. The base type of this array type is the type of the slice.

The bounds of the discrete range define those of the slice and must be of the type of the index of the array. The slice is a *null slice* if the discrete range is a null range. It is an error if the direction of the discrete range is not the same as that of the index range of the array denoted by the prefix of the slice name.

For the evaluation of a name that is a slice, the prefix and the discrete range are evaluated. It is an error if either of the bounds of the discrete range does not belong to the index range of the prefixing array, unless the slice is a null slice. (The bounds of a null slice need not belong to the subtype of the index.)

Examples:

```
signal      R15:    BIT_VECTOR (0 to 31) ;
constant    DATA:   BIT_VECTOR (31 downto 0) ;

R15(0 to 7)             -- A slice with an ascending range.
DATA(24 downto 1)       -- A slice with a descending range.
DATA(1 downto 24)       -- A null slice.
DATA(24 to 25)          -- An error.
```

NOTE

—If A is a one-dimensional array of objects, the name A(N **to** N) or A(N **downto** N) is a slice that contains one element; its type is the base type of A. On the other hand, A(N) is an element of the array A and has the corresponding element type.

6.6 Attribute names

An attribute name denotes a value, function, type, range, signal, or constant associated with a named entity.

attribute_name ::=
prefix [signature] ' attribute_designator [(expression)]

attribute_designator ::= *attribute*_simple_name

The applicable attribute designators depend on the prefix plus the signature, if any. The meaning of the prefix of an attribute must be determinable independently of the attribute designator and independently of the fact that it is the prefix of an attribute.

~~A~~ It is an error if a[15] signature ~~may follow~~ follows[16] the prefix ~~if and only if~~ and[17] the prefix ~~denotes~~ does not denote[18] a subprogram or enumeration literal, or an alias thereof. In this case, the signature is required to match (see 2.3.2) the parameter and result type profile of exactly one visible subprogram or enumeration literal, as is appropriate to the prefix.

If the attribute designator denotes a predefined attribute, the expression either must or may appear, depending upon the definition of that attribute (see ~~Section~~ Clause[19] 14); otherwise, it must not be present.

If the prefix of an attribute name denotes an alias, then the attribute name denotes an attribute of the aliased name and not the alias itself, except when the attribute designator denotes any of the predefined attributes 'SIMPLE_NAME, 'PATH_NAME, or 'INSTANCE_NAME. If the prefix of an attribute name denotes an alias and the attribute designator denotes any of the predefined attributes SIMPLE_NAME, 'PATH_NAME, or 'INSTANCE_NAME, then the attribute name denotes the attribute of the alias and not of the aliased name.

If the attribute designator denotes a user-defined attribute, the prefix cannot denote a subelement or a slice of an object.

Examples:

```
REG'LEFT(1)              -- The leftmost index bound of array REG.

INPUT_PIN'PATH_NAME      -- The hierarchical path name of the port INPUT_PIN.

CLK'DELAYED(5 ns)        -- The signal CLK delayed by 5 ns.
```

15. IR1000.4.7.
16. IR1000.4.7.
17. IR1000.4.7.
18. IR1000.4.7.
19. To conform to IEEE rules.