comp.lang.vhdl

Frequently Asked Questions And Answers: Part 4

Preliminary Remarks

This is a monthly posting to comp.lang.vhdl containing a VHDL glossary. Please send additional information directly to the editor:

edwin@ds.e-technik.uni-dortmund.de (Edwin Naroska)

Corrections and suggestions are appreciated. Thanks for all corrections.

There are three other regular postings: part 1 lists general information on VHDL, part 2 lists books on VHDL, part 3 lists products and services (PD+commercial).

The following text is reprinted from the Annex B of the IEEE Std 1076-1993 IEEE Standard VHDL Language Reference Manual. Text added by the editor of the FAQ is enclosed in square brackets . Note, the html links and the examples are not part of the original IEEE document.

VHDL Glossary

This glossary contains brief, informal descriptions for a number of terms and phrases used to define this language. The complete, formal definition of each term or phrase is provided in the main body of the standard.

For each entry, the relevant clause numbers [(of the VHDL Language Reference Manual)] in the text are given. Some descriptions refer to multiple clauses in which the single concept is discussed; for these, the clause number containing the definition of the concept is given in italics. Other descriptions contain multiple clause numbers when they refer to multiple concepts; for these, none of the clause numbers are italicized.

B.1 abstract literal: A literal of the universal_real abstract type or the universal_integer abstract type. (§13.2, §13.4)

B.2 access type: A type that provides access to an object of a given type. Access to such an object is achieved by an access value returned by an allocator; the access value is said to designate the object. (§3, §13.3)

[Example E.1]

B.3 access mode: The mode in which a file object is opened, which can be either read-only or write-only. The access mode depends on the value supplied to the Open_Kind parameter. (§3.4.1, §14.3).

B.4 access value: A value of an access type. This value is returned by an allocator and designates an object (which must be a variable) of a given type. A null access value designates no object. An access value can only designate an object created by an allocator; it cannot designate an object declared by an object declaration. (§3, 3.3)

[Example E.1]

B.5 active driver: A driver that acquires a new value during a simulation cycle regardless of whether the new value is different from the previous value. (§12.6.2, §12.6.4)

B.6 actual: An expression, a port, a signal, or a variable associated with a formal port, formal parameter, or formal generic. (§1.1.1.1, §1.1.1.2, §3.2.1.1, §4.3.1.2, §4.3.2.2, §5.2.1, §5.2.1.2)

B.7 aggregate:

1. The kind of expression, denoting a value of a composite type. The value is specified by giving the value of each of the elements of the composite type. Either a positional association or a named association may be used to indicate which value is associated with which element.
2. A kind of target of a variable assignment statement or signal assignment statement assigning a composite value. The target is then said to *be in the form of an aggregate*. (§7.3.1, §7.3.2, §7.3.4, 7.3.5, §7.5.2)

[Example E.2]

B.8 alias: An alternate name for a named entity. (§4.3.3)

B.9 allocator: An operation used to create anonymous, variable objects accessible by means of access values. (§3.3, §7.3.6)

B.10 analysis: The syntactic and semantic analysis of source code in a VHDL design file and the insertion of intermediate form representations of design units into a design library. (§1 1.1, §11.2, §11.4)

B.11 anonymous: The undefined simple name of an item, which is created implicitly. The base type of a numeric type or an array type is anonymous; similarly, the object denoted by an access value is anonymous. (§4.1)

B.12 appropriate: A prefix is said to be appropriate for a type if the type of the prefix is the type considered, or if the type of the prefix is an access type whose designated type is the type considered. (§6.1)

B.13 architecture body: A body associated with an entity declaration to describe the internal organization or operation of a design entity. An architecture body is used to describe the behavior, data flow, or structure of a design entity. (§1, §1.2)

B.14 array object: An object of an array type. (§3)

[Example E.3]

B.15 array type: A type, the value of which consists of elements that are all of the same subtype (and hence, of the same type). Each element is uniquely distinguished by an index (for a one-dimensional array) or by a sequence of indexes (for a multidimensional array). Each index must be a value of a discrete type and must lie in the correct index range. (§3.2.1)

[Example E.3]

B.16 ascending range: A range L **to** R. (§3.1)

B.17 ASCII: The American Standard Code for Information Interchange. The package Standard contains the definition of the type Character, the first 128 values of which represent the ASCII character set. (§3.1.1, §14.2)

B.18 assertion violation: A violation that occurs when the condition of an assertion statement evaluates to false. (§8.2)

[Example E.4]

B.19 associated driver: The single driver for a signal in the (explicit or equivalent) process statement containing the signal assignment statement. (§12.6.1)

B.20 associated in whole: When a single association element of a composite formal supplies the association for the entire formal. (§4.3.2.2)

B.21 associated individually: A property of a formal port, generic, or parameter of a composite type with respect to some association list. A composite formal whose association is defined by multiple association elements in a single association list is said to be *associated individually* in that list. The formats of such association elements must denote non-overlapping subelements or slices of the formal. (§4.3.2.2)

B.22 association element: An element that associates an actual or local with a local or formal. (§4.3.2.2)

B.23 association list: A list that establishes correspondences between formal or local port or parameter names and local or actual names or expressions. (§4.3.2.2)

B.24 attribute: A definition of some characteristic of a named entity. Some attributes are predefined for types, ranges, values, signals, and functions. The remaining attributes are user defined and are always constants. (§4.4)

[Example E.5]

B.25 base specifier: A lexical element that indicates whether a bit string literal is to be interpreted as a binary, octal, or hexadecimal value. (§13.7)

B.26 base type: The type from which a subtype defines a subset of possible values, otherwise known as a constraint. This subset is not required to be proper. The base type of a type is the type itself. The base type of a subtype is found by recursively examining the type mark in the subtype indication defining the subtype. If the type mark denotes a type, that type is the base type of the subtype; otherwise, the type mark is a subtype, and this procedure is repeated on that subtype. (§3) *See also* subtype.

[Example E.6]

B.27 based literal: An abstract literal expressed in a form that specifies the base explicitly. The base is restricted to the range 2 to 16. (§13.4.2)

B.28 basic operation: An operation that is inherent in one of the following:

1. An assignment (in an assignment statement or initialization);
2. An allocator;
3. A selected name, an indexed name. or a slice name;
4. A qualification (in a qualified expression), an explicit type conversion, a formal or actual designator in the form of a type conversion, or an implicit type conversion of a value of type universal_integer or universal_real to the corresponding value of another numeric type; or
5. A numeric literal (for a universal type), the literal null (for an access type), a string literal, a bit string literal, an aggregate, or a predefined attribute. (§3)

B.29 basic signal: A signal that determines the driving values for all other signals. A basic signal is

- Either a scalar signal or a resolved signal;
- Not a subelement of a resolved signal;
- Not an implicit signal of the form S'Stable(T), S'Quiet(T), or S'Transaction; and
- Not an implicit signal GUARD. (§12.6.2)

B.30 belong (to a range): A property of a value with respect to some range. The value V is said to *belong to a range* if the relations (lower bound <= V) and (V <= upper bound) are both true, where lower bound and upper bound are the lower and upper bounds, respectively, of the range. (§3.1, §3.2.1)

B.31 belong (to a subtype): A property of a value with respect to some subtype. A value is said to *belong to a subtype* of a given type if it belongs to the type and satisfies the applicable constraint. (§3, §3.2.1)

B.32 binding: The process of associating a design entity and, optionally, an architecture with an instance of a component. A binding can be specified in an explicit or a default binding indication. (§1.3, §5.2.1, §5.2.2, §12.3.2.2, §12.4.3)

B.33 bit string literal: A literal formed by a sequence of extended digits enclosed between two quotation (") characters and preceded by a base specifier. The type of a bit string literal is determined from the context. (§7.3.1, §13.7)

[Example E.7]

B.34 block: The representation of a portion of the hierarchy of a design. A block is either an external block or an internal block. (§1, §1.1.1.1, §1.1.1.2, §1.2.1, §1.3, §1.3.1, §1.3.2)

[Example E.8]

B.35 bound: A label that is identified in the instantiation list of a configuration specification. (§5.2)

B.36 box: The symbol <> in an index subtype definition, which stands for an undefined range. Different objects of the type need not have the same bounds and direction. (§3.2.1)

B.37 bus: One kind of guarded signal. A bus floats to a user-specified value when all of its drivers are turned off. (§4.3.1.2, §4.3.2)

B.38 character literal: A literal of the character type. Character literals are formed by enclosing one of the graphic characters (including the space and nonbreaking space characters) between two apostrophe (') characters. (§13.2, §13.5)

B.39 character type: An enumeration type with at least one character literal among its enumeration literals. (§3.1.1, §3.1.1.1)

B.40 closely related types: Two type marks that denote the same type or two numeric types. Two array types may also be closely related if they have the same dimensionality, if their index types at each position are closely related, and if the array types have the same element types. Explicit type conversion is only allowed between closely related types. (§7.3.5)

B.41 complete: A loop that has finished executing. Similarly, an iteration scheme of a loop is complete when the condition of a while iteration scheme is FALSE or all of the values of the discrete range of a for iteration scheme have been assigned to the iteration parameter. (§8.9)

B.42 complete context: A declaration, a specification, or a statement; complete contexts are used in overload resolution. (§10.5)

B.43 composite type: A type whose values have elements. There are two classes of composite types: array types and record types. (§3, §3.2)

[Example E.3]

B.44 concurrent statement: A statement that executes asynchronously, with no defined relative order. Concurrent statements are used for dataflow and structural descriptions. (§9)

[Example E.9]

B.45 configuration: A construct that defines how component instances in a given block are bound to design entities in order to describe how design entities are put together to form a complete design. (§1, §1.3. 5.2)

B.46 conform: Two subprogram specifications, are said to conform if, apart from certain allowed minor variations, both specifications are formed by the same sequence of lexical elements, and corresponding lexical elements are given the same meaning by the visibility rules. Conformance is defined similarly for deferred constant declarations. (§2.7)

B.47 connected: A formal port associated with an actual port or signal. A formal port associated with the reserved word **open** is said to be *unconnected*. (§1.1.1.2)

B.48 constant: An object whose value may not be changed. Constants may be *explicitly* declared, subelements of explicitly declared constants, or interface constants. Constants declared in packages may also be deferred constants. (§4.3.1.1)

B.49 constraint: A subset of the values of a type. The set of possible values for an object of a given type that can be subjected to a condition called a constraint. A value is said to satisfy the constraint if it satisfies the corresponding condition. There are index constraints, range constraints, and size constraints. (§3)

B.50 conversion function: A function used to convert values flowing through associations. For interface objects of mode **in**, conversion functions are allowed only on actuals. For interface objects of mode **out** or **buffer**, conversion functions are allowed only on formals. For interface objects of mode **inout** or **linkage**, conversion functions are allowed on both formals and actuals. Conversion functions have a single parameter. A conversion function

associated with an actual accepts the type of the actual and returns the type of the formal. A conversion function associated with a formal accepts the type of the formal and returns the type of the actual. (§4.3.2.2)

B.51 convertible: A property of an operand with respect to some type. An operand is convertible to some type if there exists an implicit conversion to that type. (§7.3.5)

B.52 current value: The value component of the single transaction of a driver whose time component is not greater than the current simulation time. (§12.6. 12.6.1, §12.6.2, §12.6.3)

B.53 decimal literal: An abstract literal that is expressed in decimal notation. The base of the literal is implicitly 10. The literal may optionally contain an exponent or a decimal point and fractional part. (§13.4.1)

B.54 declaration: A construct that defines a declared entity and associates an identifier (or some other notation) with it. This association is in effect within a region of text that is called the scope of the declaration. Within the scope of a declaration, there are places where it is possible to use the identifier to refer to the associated declared entity; at such places, the identifier is said to be the simple name of the named entity. The simple name is said to denote the associated named entity. (§4)

B.55 declarative part: A syntactic component of certain declarations or statements (such as entity declarations, architecture bodies, and block statements). The declarative part defines the lexical area (usually introduced by a keyword such as **is** and terminated with another keyword such as **begin**) within which declarations may occur. (§1.1.2, §1.2.1, §1.3, §2.6, §9.1, §9.2, §9.6.1, §9.6.2)

B.56 declarative region: A semantic component of certain declarations or statements. A declarative region may include disjoint parts, such as the declarative region of an entity declaration, which extends to the end of any architecture body for that entity. (§10.1)

B.57 decorate: To associate a user-defined attribute with a named entity and to {define} the value of that attribute. (§5.1)

B.58 default expression: A default value that is used for a formal generic, port, or parameter if the interface object is unassociated. A default expression is also used to provide an initial value for signals and their drivers. (§4.3.1.2, §4.3.2.2)

[Example E.10]

B.59 deferred constant: A constant that is declared without an assignment symbol (:=) and expression in a package declaration. A corresponding full declaration of the constant must exist in the package body to define the value of the constant. (§4.3.1.1)

[Example E.11]

B.60 delta cycle: A simulation cycle in which the simulation time at the beginning of the cycle is the same as at the end of the cycle. That is, simulation time is not advanced in a delta cycle. Only nonpostponed processes can be executed during a delta cycle. (§12.6.4)

B.61 denote: A property of the identifier given in a declaration. Where the declaration is visible, the identifier given in the declaration is said to *denote* the named entity declared in the declaration. (§4)

B.62 depend (on a library unit): A design unit that explicitly or implicitly mentions other library units in a use clause. These dependencies affect the allowed order of analysis of design units. (§11.4)

B.63 depend (on a signal value): A property of an implicit signal with respect to some other signal. The current value of an implicit signal R is said to *depend* on the current value of another signal S if R denotes an implicit signal S'Stable(T), S'Quiet(T), or S'Transaction, or if R denotes an implicit GUARD signal and S is any other implicit signal named within the guard expression that defines the current value of R. (§12.6.3)

B.64 descending range: A range L **downto** R. (§3.1)

B.65 design entity: An entity declaration together with an associated architecture body. Different design entities may share the same entity declaration, thus describing different components with the same interface or different views of the same component. (§1)

B.66 design file: One or more design units in sequence. (§1.1)

B.67 design hierarchy: The complete representation of a design that results from the successive decomposition of a design entity into subcomponents and binding of those components to other design entities that may be decomposed in a similar manner. (§1)

B.68 design library: A host-dependent storage facility for intermediate-form representations of analyzed design units. (§11.2)

B.69 design unit: A construct that can be independently analyzed and stored in a design library. A design unit may be an entity declaration, an architecture body, a configuration declaration, a package declaration, or a package body declaration. (§11.1)

B.70 designate: A property of access values that relates the value to some object when the access value is nonnull. A nonnull access value is said to *designate* an object. (§3.3)

B.71 designated subtype: For an access type, the subtype defined by the subtype indication of the access type definition. (§3.3)

B.72 designated type: For an access type, the base type of the subtype defined by the subtype indication of the access type definition. (§3.3)

B.73 designator:

1. Syntax that forms part of an association element. A formal designator specifies which formal parameter, port, or generic (or which subelement or slice of a parameter, port, or generic) is to be associated with an actual by the given association element. An actual designator specifies which actual expression, signal, or variable is to be associated with a formal (or subelement or subelements of a formal). An actual designator may also specify that the formal in the given association element is to be left unassociated (with an actual designator of **open**). (§4.3.2.2)
2. An identifier, character literal, or operator symbol that defines an alias for some other name. (§4.3.3)
3. A simple name that denotes a predefined or user-defined attribute in an attribute name, or a user-defined attribute in an attribute specification. (§5.1, §6.6)
4. An simple name, character literal, or operator symbol, and possibly a signature, that denotes a named entity in the entity name list of an attribute specification. (§5.1)
5. An identifier or operator symbol that defines the name of a subprogram. (§2.1)

B.74 directly visible: A visible declaration that is not visible by selection. A declaration is directly visible within its immediate scope, excluding any places where the declaration is hidden. A declaration occurring immediately within the visible part of a package can be made directly visible by means of a use clause. (§10.3, 10.4). *See also* visible.

B.75 discrete array: A one-dimensional array whose elements are of a discrete type. (§7.2.3)

B.76 discrete range: A range whose bounds are of a discrete type. (§3.2.1, §3.2.1.1)

B.77 discrete type: An enumeration type or an integer type. Each value of a discrete type has a position number that is an integer value. Indexing and iteration rules use values of discrete types. (§3.1)

B.78 driver: A container for a projected output waveform of a signal. The value of the signal is a function of the current values of its drivers. Each process that assigns to a given signal implicitly contains a driver for that signal. A signal assignment statement affects only the associated driver(s). (§12.4.4, §12.6.1, §12.6.2, §12.6.3)

B.79 driving value: The value a signal provides as a source of other signals. (§12.6.2)

B.80 effective value: The value obtained by evaluating a reference to the signal within an expression. (§12.6.2)

B.81 elaboration: The process by which a declaration achieves its effect. Prior to the completion of its elaboration (including before the elaboration), a declaration is not yet elaborated. (§12)

B.82 element: A constituent of a composite type. (§3) *See also* subelement.

B.83 entity declaration: A definition of the interface between a given design entity and the environment in which it is used. It may also specify declarations and statements that are part of the design entity. A given entity declaration may be shared by many design entities, each of which has a different architecture. Thus, an entity declaration can potentially represent a class of design entities, each with the same interface. (§1, §1.1)

B.84 enumeration literal: A literal of an enumeration type. An enumeration literal may be either an identifier or a character literal. (§5.1.1, §7.3.1)

B.85 enumeration type: A type whose values are defined by listing (enumerating) them. The values of the type are represented by enumeration literals. (§3.1, §3.1.1)

[Example E.13]

B.86 error: A condition that makes the source description illegal. If an error is detected at the time of analysis of a design unit, it prevents the creation of a library unit for the given design unit. A run-time error causes simulation to terminate. (§11.4)

B.87 erroneous: An error condition that cannot always be detected. (§2.1.1.1, §2.2)

B.88 event: A change in the current value of a signal, which occurs when the signal is updated with its effective value. (§12.6.2)

B.89 execute:

1. When first the design hierarchy of a model is elaborated, then its nets are initialized, and finally simulation proceeds with repetitive execution of the simulation cycle, during which processes are executed and nets are updated.
2. When a process performs the actions specified by the algorithm described in its statement part. (§12, 12.6)

B.90 expanded name: A selected name (in the syntactic sense) that denotes one or all of the primary units in a library or any named entity within a primary unit. (§6.3, §8.1) *See also* selected name.

B.91 explicit ancestor: The parent of the implicit signal that is defined by the predefined attributes ’DELAYED, ’QUIET, ’STABLE, or ’TRANSACTION. It is determined using the prefix of the attribute. If the prefix denotes an explicit signal or (or member thereof), then that is the explicit ancestor of the implicit signal. If the prefix is one of the implicit signals defined by the predefined attributes ’DELAYED, ’QUIET, ’STABLE, or ’TRANSACTION, this rule is applied recursively. If the prefix is an implicit signal GUARD, the signal has no explicit ancestor. (§2.2)

B.92 explicit signal: A signal defined by the predefined attributes ’DELAYED, ’QUIET, ’STABLE, or ’TRANSACTION. (§2.2)

B.93 explicitly declared constant: A constant of a specified type that is declared by a constant declaration. (§4.3.1.1)

B.94 explicitly declared object: An object of a specified type that is declared by an object declaration. An object declaration is called a single-object declaration if its identifier list has a single identifier; it is called a *multiple-object declaration* if the identifier list has two or more identifiers. (§4.3, §4.3.1) *See also* implicitly declared object.

B.95 expression: A formula that defines the computation of a value. (§7.1)

B.96 extend: A property of source text forming a declarative region with disjoint parts. In a declarative region with disjoint parts, if a portion of text is said to *extend* from some specific point of a declarative region to the end of the region, then this portion is the corresponding subset of the declarative region (and does not include intermediate declarative items between an interface declaration and a corresponding body declaration). (§10.1)

B.97 extended digit: A lexical element that is either a digit or a letter. (§13.4.2)

B.98 external block: A top-level design entity that resides in a library and may be used as a component in other designs. (§1)

B.99 file type: A type that provides access to objects containing a sequence of values of a given type. File types are typically used to access files in the host system environment. The value of a file object is the sequence of values contained in the host system file. (§3, §3.4)

B.100 floating point types: A discrete scalar type whose values approximate real numbers. The representation of a floating point type includes a minimum of six decimal digits of precision. (§3.1, §3.1.4)

B.101 foreign subprogram: A subprogram that is decorated with the attribute ’FOREIGN, defined in package STANDARD. The STRING value of the attribute may specify implementation-dependent information about the foreign subprogram. Foreign subprograms may have non-VHDL implementations. An implementation may place restrictions on the allowable modes, classes, and types of the formal parameters to a foreign subprogram, such as constraints on the number and allowable order of the parameters. (§2.2)

B.102 formal: A formal port or formal generic of a design entity, a block statement, or a formal parameter of a subprogram. (§2.1.1, §4.3.2.2, §5.2.1.2, §9.1)

B.103 full declaration: A constant declaration occurring in a package body with the same identifier as that of a deferred constant declaration in the corresponding package declaration. A full type declaration is a type declaration corresponding to an incomplete type declaration. (§2.6)

[Example E.11]

B.104 fully bound: A binding indication for the component instance implies an entity interface and an architecture. (§5.2.1.1)

B.105 generate parameter: A constant object whose type is the base type of the discrete range of a generate parameter specification. A generate parameter is declared by a generate statement. (§9.7)

B.106 generic: An interface constant declared in the block header of a block statement, a component declaration, or an entity declaration. Generics provide a channel for static information to be communicated to a block from its environment. Unlike constants, however, the value of a generic can be supplied externally, either in a component instantiation statement or in a configuration specification. (§1.1.1.1)

B.107 generic interface list: A list that defines local or formal generic constants. (§1.1.1.1, §4.3.2.1)

B.108 globally static expression: An expression that can be evaluated as soon as the design hierarchy in which it appears is elaborated. A locally static expression is also globally static unless the expression appears in a dynamically elaborated context. (§7.4)

B.109 globally static primary: A primary whose value can be determined during the elaboration of its complete context and that does not thereafter change. Globally static primaries can only appear within statically elaborated contexts. (§7.4.2)

B.110 group: A named collection of name entities. Groups relate different name entities for the purposes not specified by the language. In particular, groups may be decorated with attributes. (§4.6, §4.7)

B.111 guard: *See* guard expression.

B.112 guard expression: A Boolean-valued expression associated with a block statement that controls assignments to guarded signals within the block. A guard expression defines an implicit signal GUARD that may be used to control the operation of certain statements within the block. (§4.3.1.2, §9.1, §9.5)

B.113 guarded assignment: A concurrent signal assignment statement that includes the option **guarded**, which specifies that the signal assignment statement is executed when a signal GUARD changes from FALSE to TRUE, or when that signal has been TRUE and an event occurs on one of the signals referenced in the corresponding GUARD expression. The signal GUARD may be one of the implicitly declared GUARD signals associated with block statements that have guard expressions, or it may be an explicitly declared signal of type Boolean that is visible at the point of the concurrent signal assignment statement. (§9.5)

B.114 guarded signal: A signal declared as a register or a bus. Such signals have special semantics when their drivers are updated from within guarded signal assignment statements. (§4.3.1.2)

B.115 guarded target: A signal assignment target consisting only of guarded signals. An unguarded target is a target consisting only of unguarded signals. (§9.5)

B.116 hidden: A declaration that is not directly visible. A declaration may be *hidden* in its scope by a homograph of the declaration. (§10.3)

B.117 homograph: A reflexive property of two declarations. Each of two declarations is said to be a *homograph* of the other if both declarations have the same identifier and overloading is allowed for at most one of the two. If overloading is allowed for both declarations, then each of the two is a homograph of the other if they have the same identifier, operator symbol, or character literal, as well as the same parameter and result type profile. (§1.3.1, §10.3)

B.118 identify: A property of a name appearing in an element association of an assignment target in the form of an aggregate. The name is said to *identify* a signal or variable and any subelements of that signal or variable. (§8.4, §8.5)

B.119 immediate scope: A property of a declaration with respect to the declarative region within which the declaration immediately occurs. The immediate scope of the declaration extends from the beginning of the declaration to the end of the declarative region. (§10.2)

B.120 immediately within: A property of a declaration with respect to some declarative region. A declaration is said to occur *immediately within* a declarative region if this region is the innermost region that encloses the declaration, not counting the declarative region (if any) associated with the declaration itself. (§10.1)

B.121 implicit signal: Any signal S'Stable(T), S'Quiet(T), S'Delayed, or S'Transaction, or any implicit GUARD signal. A member of an implicit signal is also an implicit signal. (§12.6.2,§12.6.3, §12.6.4)

B.122 implicitly declared object: An object whose declaration is not explicit in the source description, but is a consequence of other constructs; for example, signal GUARD. (§4.3, §9.1, §14.1) *See also* explicitly declared object.

B.123 imply: A property of a binding indication in a configuration specification with respect to the design entity indicated by the binding specification. The binding indication is said to *imply* the design entity; the design entity may be indicated directly, indirectly, or by default. (§5.2.1.1)

B.124 impure function: A function that may return a different value each time it is called, even when different calls have the same actual parameter values. A pure function returns the same value each time it is called using the same values as actual parameters. A impure function can update objects outside of its scope and can access a broader class of values than a pure function. (§2)

B.125 incomplete type declaration: A type declaration that is used to {define} mutually dependent and recursive access types. (§3.3.1)

[Example E.12]

B.126 index constraint: A constraint that determines the index range for every index of an array type, and thereby the bounds of the array. An index constraint is *compatible* with an array type if and only if the constraint defined by each discrete range in the index constraint is compatible with the corresponding index subtype in the array type. An array value *satisfies* an index constraint if the array value and the index constraint have the same index range at each index position . (§3.1, §3.2.1.1)

B.127 index range: A multidimensional array has a distinct element for each possible sequence of index values that can be formed by selecting one value for each index (in the given order). The possible values for a given index are all the values that belong to the corresponding range. This range of values is called the *index range*. (§3.2.1)

B.128 index subtype: For a given index position of an array, the *index subtype* is denoted by the type mark of the corresponding index subtype definition. (§3.2.1)

B.129 inertial delay: A delay model used for switching circuits; a pulse whose duration is shorter than the switching time of the circuit will not be transmitted. Inertial delay is the default delay mode for signal assignment statements. (§8.4) *See also* transport delay.

[Example E.15]

B.130 initial value expression: An expression that specifies the initial value to be assigned to a variable. (§4.3.1.3)

B.131 inputs: The signals identified by the longest static prefix of each signal name appearing as a primary in each expression (other than time expressions) within a concurrent signal assignment statement. (§9.5)

B.132 instance: A subcomponent of a design entity whose prototype is a component declaration, design entity, or configuration declaration. Each instance of a component may have different actuals associated with its local ports and generics. A component instantiation statement whose instantiated unit denotes a component creates an instance of the corresponding component. A component instantiation statement whose instantiated unit denotes either a design entity or a configuration declaration creates an instance of the denoted design entity. (§9.6, §9.6.1, §9.6.2)

B.133 integer literal: An abstract literal of the type universal_integer that does not contain a base point. (§13.4)

B.134 integer type: A discrete scalar type whose values represent integer numbers within a specified range. (§3.1, §3.1.2)

[Example E.14]

B.135 interface list: A list that declares the interface objects required by a subprogram, component, design entity, or block statement. (§4.3.2.1)

B.136 internal block: A nested block in a design unit, as defined by a block statement. (§1)

[Example E.8]

B.137 ISO: The International Organization for Standardization.

B.138 ISO 8859-1: The ISO Latin-l character set. Package Standard contains the definition of type Character, which represents the ISO Latin-l character set. (§3.1.1, §14.2)

B.139 kernel process: A conceptual representation of the agent that coordinates the activity of user-defined processes during a simulation. The kernel process causes the execution of I/O operations, the propagation of signal values, and the updating of values of implicit signals [such as S'Stable(T)]; in addition, it detects events that occur and causes the appropriate processes to execute in response to those events. (§12.6)

B.140 left of: When both a value V1 and a value V2 belong to a range and either the range is an ascending range and V2 is the successor of V1, or the range is a descending range and V2 is the predecessor of V1. (§3.1)

B.141 left-to-right order: When each value in a list of values is to the left of the next value in the list within that range, except for the last value in the list. (§3.1)

B.142 library: *See* design library.

B.143 library unit: The representation in a design library of an analyzed design unit. (§11.1)

B.144 literal: A value that is directly specified in the description of a design. A literal can be a bit string literal, enumeration literal, numeric literal, string literal, or the literal **null**. (§7.3.1)

B.145 local generic: An interface object declared in a component declaration that serves to connect a formal generic in the interface list of an entity and an actual generic or value in the design unit instantiating that entity. (§4.3, §4.3.2.2, §4.5)

B.146 local port: A signal declared in the interface list of a component declaration that serves to connect a formal port in the interface list of an entity and an actual port or signal in the design unit instantiating that entity. (§4.3, §4.3.2.2, §4.5)

B.147 locally static expression: An expression that can be evaluated during the analysis of the design unit in which it appears. (§7.4, §7.4.1)

B.148 locally static name: A name in which every expression is locally static (if every discrete range that appears as part of the name denotes a locally static range or subtype and if no prefix within the name is either an object or value of an access type or a function call). (§6.1)

B.149 locally static primary: One of a certain group of primaries that includes literals, certain constants, and certain attributes. (§7.4)

B.150 locally static subtype: A subtype whose bounds and direction can be determined during the analysis of the design unit in which it appears. (§7.4.1)

B.151 longest static prefix: The name of a signal or a variable name, if the name is a static signal or variable name. Otherwise, the longest static prefix is the longest prefix of the name that is a static signal or variable name. (§6.1) *See also* static signal name.

B.152 loop parameter: A constant, implicitly declared by the for clause of a loop statement, used to count the number of iterations of a loop. (§8.9)

B.153 lower bound: For a nonnull range L **to** R or L **downto** R, the smaller of L and R. (§3.1)

B.154 match: A property of a signature with respect to the parameter and subtype profile of a subprogram or enumeration literal. The signature is said to *match* the parameter and result type profile if certain conditions are true. (§2.3.2)

B.155 matching elements: Corresponding elements of two composite type values that are used for certain logical and relational operations. (§7.2.3)

B.156 member: A slice of an object, a subelement, or an object; or a slice of a subelement of an object. (§3)

B.157 mode: The direction of information flow through the port or parameter. Modes are **in**, **out**, **inout**, **buffer**, or **linkage**. (§4.3.2)

B.158 model: The result of the elaboration of a design hierarchy. The *model* can be executed in order to simulate the design it represents. (§12, §12.6)

B.159 name: A property of an identifier with respect to some named entity. Each form of declaration associates an identifier with a named entity. In certain places within the scope of a declaration, it is valid to use the identifier to refer to the associated named entity; these places are defined by the visibility rules. At such places, the identifier is said to be the *name* of the named entity. (§4, §6.1)

B.160 named association: An association element in which the formal designator appears explicitly. (§4.3.2.2, §7.3.2)

B.161 named entity: An item associated with an identifier, character literal, or operator symbol as the result of an explicit or implicit declaration. (§4) *See also* name.

B.162 net: A collection of drivers, signals (including ports and implicit signals), conversion functions, and resolution functions that connect different processes. Initialization of a net occurs after elaboration, and a net is updated during each simulation cycle. (§12, §12.1, §12.6.2)

B.163 nonobject alias: An alias whose designator denotes some named entity other than an object. (§4.3.3, §4.3.3.2) *See also* object alias.

B.164 nonpostponed process: An explicit or implicit process whose source statement does not contain the reserved word **postponed**. When a nonpostponed process is resumed, it executes in the current simulation cycle. Thus, nonpostponed processes have access to the current values of signals, whether or not those values are stable at the current model time. (§9.2)

B.165 null array: Any of the discrete ranges in the index constraint of an array that define a null range. (§3.2.1.1)

B.166 null range: A range that specifies an empty subset of values. A range L **to** R is a null range if L > R, and range L **downto** R is a null range if L < R. (§3.1)

B.167 null slice: A slice whose discrete range is a null range. (§6.5)

B.168 null waveform element: A waveform element that is used to turn off a driver of a guarded signal. (§8.4.1)

B.169 null transaction: A transaction produced by evaluating a null waveform element. (§8.4.1)

B.170 numeric literal: An abstract literal, or a literal of a physical type. (§7.3.1)

B.171 numeric type: An integer type, a floating point type, or a physical type. (§3.1)

B.172 object: A named entity that has a value of a given type. An object can be a constant, signal, variable, or file. (§4.3.3)

B.173 object alias: An alias whose alias designator denotes an object (that is, a constant, signal, variable, or file). (§4.3.3, §4.3.3.1) *See also* nonobject alias.

B.174 overloaded: Identifiers or enumeration literals that denote two different name entities. Enumeration literals, subprograms, and predefined operators may be overloaded. At any place where an overloaded enumeration literal occurs in the text of a program, the type of the enumeration literal must be determinable from the context. (§2.1, §2.3, §2.3.1, §2.3.2, §3.1.1)

B.175 parameter: A constant, signal, variable, or file declared in the interface list of a subprogram specification. The characteristics of the class of objects to which a given parameter belongs are also characteristics of the parameter. In addition, a parameter has an associated mode that specifies the direction of data flow allowed through the parameter. (§2.1.1, §2.1.1.1, §2.1.1.2, §2.1.1.3, §2.3, §2.6)

B.176 parameter interface list: An interface list that declares the parameters for a subprogram. It may contain interface constant declarations, interface signal declarations, interface variable declarations, interface file declarations, or any combination thereof. (§4.3.2.1)

B.177 parameter type profile: Two formal parameter lists that have the same number of parameters, and at each parameter position the corresponding parameters have the same base type. (§2.3)

B.178 parameter and result type profile: Two subprograms that have the same parameter type profile, and either both are functions with the same result base type, or neither of the two is a function. (§2.3)

B.179 parent: A process or a subprogram that contains a procedure call statement for a given procedure or for a parent of the given procedure. (§2.2)

B.180 passive process: A process statement where neither the process itself, nor any procedure of which the process is a parent, contains a signal assignment statement. (§9.2)

B.181 physical literal: A numeric literal of a physical type. (§3.1.3)

B.182 physical type: A numeric scalar type that is used to represent measurements of some quantity. Each value of a physical type has a position number that is an integer value. Any value of a physical type is an integral multiple of the primary unit of measurement for that type. (§3.1, §3.1.3)

B.183 port: A channel for dynamic communication between a block and its environment. A signal declared in the interface list of an entity declaration, in the header of a block statement, or in the interface list of a component declaration. In addition to the characteristics of signals, ports also have an associated mode; the mode constrains the directions of data flow allowed through the port. (§1.1.1.2, §4.3.1.2)

B.184 port interface list: An interface list that declares the inputs and outputs of a block, component, or design entity. It consists entirely of interface signal declarations. (§1.1.1, §1.1.1.2, §4.3.2.1, §4.3.2.2, §9.1)

B.185 positional association: An association element that does not contain an explicit appearance of the formal designator. An actual designator at a given position in an association list corresponds to the interface element at the same position in the interface list. (§4.3.2.2, §7.3.2)

B.186 postponed process: An explicit or implicit process whose source statement contains the reserved word **postponed**. When a postponed process is resumed, it does not execute until the final simulation cycle at the current modeled time. Thus, a postponed process accesses the values of signals that are the "stable" values at the current simulated time. (§9.2)

B.187 predefined operators: Implicitly defined operators that operate on the predefined types. Every predefined operator is a pure function. No predefined operators have named formal parameters; therefore, named association may not be used when invoking a predefined operation. (§7.2, §14.2)

B.188 primary: One of the elements making up an expression. Each primary has a value and a type. (§7.1)

B.189 projected output waveform: A sequence of one or more transactions representing the current and projected future values of the driver. (§12.6.1)

B.190 pulse rejection limit: The threshold time limit for which a signal value whose duration is greater than the limit will be propagated. A pulse rejection limit is specified by the reserved word **reject** in an inertially delayed signal assignment statement. (§8.4)

B.191 pure function: A function that returns the same value each time it is called with the same values as actual parameters. An *impure* function may return a different value each time it is called, even when different calls have the same actual parameter values. (§2.1)

B.192 quiet: In a given simulation cycle, a signal that is not active. (§12.6.2)

B.193 range: A specified subset of values of a scalar type. (§3.1) *See also* ascending range, belong (to a range), descending range, lower bound, and upper bound.

B.194 range constraint: A construct that specifies the range of values in a type. A range constraint is *compatible* with a subtype if each bound of the range belong to a subtype or if the range constraint defines a null range. The direction of a range constraint is the same as the direction of its range. (§3.1, §3.1.2, §3.1.3, §3.1.4)

B.195 read: The value of an object is said to be *read* when its value is referenced or when certain of its attributes are referenced. (§4.3.2)

B.196 real literal: An abstract literal of the type universal_real that contains a base point. (§13.4)

B.197 record type: A composite type whose values consist of named elements. (§3.2.2, §7.3.2.1)

B.198 reference: Access to a named entity. Every appearance of a designator (a name, character literal, or operator symbol) is a reference to the named entity denoted by the designator, unless the designator appears in a library clause or use clause. (§10.4, §11.2)

B.199 register: A kind of guarded signal that retains its last driven value when all of its drivers are turned off. (§4.3.1.2)

B.200 regular structure: Instances of one or more components arranged and interconnected (via signals) in a repetitive way. Each instance may have characteristics that depend upon its position within the group of instances. Regular structures may be represented through the use of the generate statement. (§9.7)

B.201 resolution: The process of determining the resolved value of a resolved signal based on the values of multiple sources for that signal. (§2.4, §4.3.1.2)

B.202 resolution function: A user-defined function that computes the resolved value of a resolved signal. (§2.4, §4.3.1.2)

B.203 resolution limit: The primary unit of type TIME (by default, 1 femtosecond). Any TIME value whose absolute value is smaller than this limit is truncated to zero (0) time units. (§3.1.3.1)

B.204 resolved signal: A signal that has an associated resolution function. (§4.3.1.2)

B.205 resolved value: The output of the resolution function associated with the resolved signal, which is determined as a function of the collection of inputs from the multiple sources of the signal. (§2.4, §4.3.1.2)

B.206 resource library: A library containing library units that are referenced within the design unit being analyzed. (§11.2)

B.207 result subtype: The subtype of the returned value of a function. (§2.1)

B.208 resume: The action of a wait statement upon an enclosing process when the conditions on which the wait statement is waiting are satisfied. If the enclosing process is a nonpostponed process, the process will subsequently execute during the current simulation cycle. Otherwise, the process is a postponed process, which will execute during the final simulation cycle at the current simulated time. (§12.6.3)

B.209 right of: When a value V1 and a value V2 belong to a range and either the range is an ascending range and V2 is the predecessor of V1, or the range is a descending range and V2 is the successor of V1. (§14.1)

B.210 satisfy: A property of a value with respect to some constraint. The value is said to *satisfy* a constraint if the value is in the subset of values determined by the constraint. (§3, §3.2.1.1)

B.211 scalar type: A type whose values have no elements. Scalar types consist of enumeration types, integer types, physical types, and floating point types. Enumeration types and integer types are called discrete types. Integer types, floating point types, and physical types are called numeric types. All scalar types are ordered; that is, all relational operators are predefined for their values. (§3, §3.1)

B.212 scope: A portion of the text in which a declaration may be visible. This portion is defined by visibility and overloading rules. (§10.2)

B.213 selected name: Syntactically, a name having a prefix and suffix separated by a dot. Certain selected names are used to denote record elements or objects denoted by an access value. The remaining selected names are referred to as expanded names. (§6.3, §8.1) *Also see* expanded name.

B.214 sensitivity set: The set of signals to which a wait statement is sensitive. The sensitivity set is given explicitly in an on clause, or is implied by an **until** clause. (§8.1)

B.215 sequential statements: Statements that execute in sequence in the order in which they appear. Sequential statements are used for algorithmic descriptions. (§8)

B.216 short-circuit operation: An operation for which the right operand is evaluated only if the left operand has a certain value. The short-circuit operations are the predefined logical operations **and**, **or**, **nand**, and **nor** for operands of types BIT and BOOLEAN. (§7.2)

B.217 signal: An object with a past history of values. A signal may have multiple drivers, each with a current value and projected future values. The term *signal* refers to objects declared by signal declarations or port declarations. (§4.3.1.2)

B.218 signal transform: A sequential statement within a statement transform that determines which one of the alternative waveforms, if any, is to be assigned to an output signal. A signal transform can be a sequential signal assignment statement, an if statement, a case statement, or a null statement. (§9.5)

B.219 simple name: The identifier associated with a named entity, either in its own declaration or in an alias declaration. (§6.2)

B.220 simulation cycle: One iteration in the repetitive execution of the processes defined by process statements in a model. The first simulation cycle occurs after initialization. A simulation cycle can be a delta cycle or a time-advance cycle. (§12.6.4)

B.221 single-object declaration: An object declaration whose identifier list contains a single identifier; it is called a multiple-object declaration if the identifier list contains two or more identifiers. (§4.3.1)

B.222 slice: A one-dimensional array of a sequence of consecutive elements of another one-dimensional array. (§6.5)

B.223 source: A contributor to the value of a signal. A source can be a driver or port of a block with which a signal is associated or a composite collection of sources. (§4.3.1.2)

B.224 specification: A class of construct that associates additional information with a named entity. There are three kinds of attribute specifications, configuration specifications, and disconnection specifications. (§5)

B.225 statement transform: The first sequential statement in the process equivalent to the concurrent signal assignment statement. The statement transform defines the actions of the concurrent signal assignment statement when it executes. The statement transform is followed by a wait statement, which is the final statement in the equivalent process. (§9.5)

B.226 static: *See* locally static and globally static.

B.227 static name: A name in which every expression that appears as part of the name (for example, as an index expression) is a static expression (if every discrete range that appears as part of the name denotes a static range or subtype and if no prefix within the name is either an object or value of an access type or a function call). (§6.1)

B.228 static range: A range whose bounds are static expressions. (§7.4)

B.229 static signal name: A static name that denotes a signal. (§6.1)

B.230 static variable name: A static name that denotes a variable. (§6.1)

B.231 string literal: A sequence of graphic characters, or possibly none, enclosed between two quotation marks ("). The type of a string literal is determined from the context. (§7.3.1, §13.6)

B.232 subaggregate: An aggregate appearing as the expression in an element association within another, multidimensional array aggregate. The subaggregate is an (*n*-1)-dimensional array aggregate, where *n* is the dimensionality of the outer aggregate. Aggregates of multidimensional arrays are expressed in row-major (rightmost index varies fastest) order. (§7.3.2.2)

B.233 subelement: An element of another element. Where other subelements are excluded, the term *element* is used.(§3)

B.234 subprogram specification: Specifies the designator of the subprogram, any formal parameters of the subprogram, and the result type for a function subprogram. (§2.1)

B.235 subtype: A type together with a constraint. A value belongs to a subtype of a given type if it belongs to the type and satisfies the constraint; the given type is called the base type of the subtype. A type is a subtype of itself. Such a subtype is said to be *unconstrained* because it corresponds to a condition that imposes no restriction. (§3)

B.236 suspend: A process that stops executing and waits for an event or for a time period to elapse. (§12.6.4)

B.237 timeout interval: The maximum time a process will be suspended, as specified by the timeout period in the **until** clause of a wait statement. (§8.1)

B.238 to the left of: *See* left of.

B.239 to the right of: *See* right of.

B.240 transaction: A pair consisting of a value and a time. The value represents a (current or) future value of the driver; the time represents the relative delay before the value becomes the current value. (§12.6.1)

B.241 transport delay: An optional delay model for signal assignment. Transport delay is characteristic of hardware devices (such as transmission lines) that exhibit nearly infinite frequency response: any pulse is transmitted, no matter how short its duration. (§8.4) *See also* inertial delay.

[Example E.16]

B.242 type: A set of values and a set of operations. (§3)

B.243 type conversion: An expression that converts the value of a subexpression from one type to the designated type of the type conversion. Associations in the form of a type conversion are also allowed. These associations have functions and restrictions similar to conversion functions but can be used in places where conversion functions cannot. In both cases (expressions and associations), the converted type must be closely related to the designated type. (§4.3.2.2, §7.3.5) *See also* conversion function and closely related types.

B.244 unaffected: A waveform in a concurrent signal assignment statement that does not affect the driver of the target. (§8.4, §9.5.1)

B.245 unassociated formal: A formal that is not associated with an actual. (§5.2.1.2)

B.246 unconstrained subtype: A subtype that corresponds to a condition that imposes no restriction. (§3, §4.2)

B.247 unit name: A name defined by a unit declaration (either the primary unit declaration or a secondary unit declaration) in a physical type declaration. (§3.1.3)

B.248 universal_integer: An anonymous predefined integer type that is used for all integer literals. The position number of an integer value is the corresponding value of the type universal_integer. (§3.1.2, §7.3.1, §7.3.5)

B.249 universal_real: An anonymous predefined type that is used for literals of floating point types. Other floating point types have no literals. However, for each floating point type there exists an implicit conversion that converts a value of type universal_real into the corresponding value (if any) of the floating point type. (§3.1.4, §7.3.1, §7.3.5)

B.250 update: An action on the value of a signal, variable, or file. The value of a signal is said to be *updated* when the signal appears as the target (or a component of the target) of a signal assignment statement, (indirectly) when it is associated with an interface object of mode **out**, **buffer**, **inout**, or **linkage**, or when one of its subelements (individually or as part of a slice) is updated. The value of a signal is also said to be *updated* when it is subelement or slice of a resolved signal, and the resolved signal is updated. The value of a variable is said to be *updated* when the variable appears as the target (or a component of the target) of a variable assignment statement, (indirectly) when it is associated with an interface object of mode **out** or **linkage**, or when one of its subelements (individually or part of a slice) is updated. The value of a file is said to be *updated* when a WRITE operation is performed on the file object. (§4.3.2)

B.251 upper bound: For a nonnull range L **to** R or L **downto** R, the larger of L and R. (§3.1)

B.252 variable: An object with a single current value. (§4.3.1.3)

B.253 visible: When the declaration of an identifier defines a possible meaning of an occurrence of the identifier used in the declaration. A visible declaration is visible by selection (for example, by using an expanded name) or directly visible (for example, by using a simple name). (§10.3)

B.254 waveform: A series of transactions, each of which represents a future value of the driver of a signal. The transactions in a waveform are ordered with respect to time, so that one transaction appears before another if the first represents a value that will occur sooner than the value represented by the other. (§8.4)

B.255 whitespace character: A space, a nonbreaking space, or a horizontal tabulation character (SP, NBSP, or HT). (§14.3)

B.256 working library: A design library into which the library unit resulting from the analysis of a design unit is placed. (§11.2)

End of document.

Examples

E.1 access types:

```
PROCESS ...
   TYPE twobits IS ARRAY (0 TO 1) OF BIT;
   TYPE twobits_pointer_type IS ACCESS twobits;   -- declare an ACCESS type.
                -- The subprograms NEW and DEALLOCATE are declared implicitly
   VARIABLE p1, p2 : twobits_pointer_type;  -- declare two pointer variables
   ...
BEGIN
   ...
   p1 := NEW twobits;   -- allocate memory for an object of type "twobits"
   p1.ALL := (’0’,’1’);   -- store a value to the memory location
   p1 := (’0’,’1’);  -- equivalent to "p1.ALL := (’0’,’1’)"
   p1.ALL(0) := p1.ALL(1);  -- referencing subelements of an array pointed by
                -- an access value
   p2 := p1;   -- "p2" and "p1" are now pointing to the same memory location
   DEALLOCATE(ip2);   -- free memory
   ip1 := NULL;  -- "ip1" now points to nil
END PROCESS;
```

E.2 array and record aggregates:

```
SIGNAL bvec : BIT_VECTOR(0 TO 3);
SIGNAL one_bit : BIT_VECTOR(0 TO 0);

SIGNAL b1, b2, b3, b4 : BIT;

TYPE rec IS RECORD
   a : BIT;
   b : INTEGER;
END RECORD;
SIGNAL rvec : rec;
...
-- examples for array aggregates
bvec <= (1=>’1’, OTHERS=>’0’);  -- assigns (’0’,’1’,’0’,’0’) to "bvec" (named
                -- association)
bvec <= (’0’,’1’,’0’,’0’);  -- positional association
(b1,b2,b3,b4) <= bvec AFTER 20 ns; -- "b1" will be assigned "bvec(0)",
                -- "b2" "bvec(1)", ...
one_bit <= (0=>’1’);  -- named association has to be used to create an
                     -- aggregat containing only a single element

-- examples for record aggregates
rec <= (’0’, -9);  -- "rec.a" = ’0’, "rec.b" = -9 (positional association)
rec <= (b=>123, a=>’1’);  -- "rec.a" = ’1’, "rec.b" = 123 (named association)
```

E.3 array and record types:

```
-- constrained arrays
TYPE word IS ARRAY (31 DOWNTO 0) OF BIT;
TYPE memory IS ARRAY (0 TO 100, 0 TO 7) OF BIT;  -- two dimensional array

-- unconstrained arrays

TYPE r_vector IS ARRAY (POSITIVE RANGE <>) OF REAL;
TYPE i_vector IS ARRAY (NATURAL RANGE <>) OF INTEGER;  -- POSITIVE and NATURAL
                -- are predefined integer subtypes ranging 1 to INTEGER'HIGH,
                -- respectively 0 to INTEGER'HIGH
...

-- declaring array objects
VARIABLE bus : word := (OTHERS => '0');  -- default value of bus
                                         -- is ('0','0',...'0')
VARIABLE mem : memory := (OTHERS => (OTHERS => '1')); -- default value of mem
                                         -- is (('1',...'1'),...,('1',...'1'))
VARIABLE a : r_vector(1 TO 3) := (1.0, 2.4, 3.4);  -- "a" has 3 elements
VARIABLE b : i_vector(0 TO 1);  -- "b" has two elements
...

-- accessing array elements
bus(3) := mem(2,3);


-- record types
TYPE rec IS RECORD  -- record type "rec" contains 4 elements "a" to "d"
   a : BIT;
   b : INTEGER;
   c : REAL;
   d : bit_vector(0 TO 2);
END RECORD;

-- declaring record objects
VARIABLE rec_var : rec := ('0', 34, -123.4, (OTHERS => '0'));  -- "a" = '0',
                -- "b" = 34, "c" = -123.4 and "d" = ('0','0','0')

-- accessing record objects
rec_var := (a=>'1', d=>('0','1','0'), b=>-1, c=>12.45);
rec_var.a := '0';  -- accessing element "a" of record "rec_var"
rec_var.b := 111;  -- accessing element "b" of record "rec_var"
```

E.4 assertion statement:

```
VARIABLE a,b : INTEGER;
...
a:= 10;
b:= 11;

-- the assertion statement will report a message only if the condition
-- expression evaluates to FALSE

ASSERT a /= b              -- assert statement will report nothing because
   REPORT "a not equal b"  -- a /= b evaluates to TRUE
   SEVERITY WARNING;
```

```
ASSERT a = b               -- assert statement will report the WARNING
   REPORT "a not equal b"  -- message "a not equal b"
   SEVERITY WARNING;

ASSERT a = b
   REPORT "a not equal b"; -- will report the ERROR message "a not equal b"

ASSERT a = b;              -- will report the ERROR message
                           -- "Assertion violation"
```

E.5 attributes:

```
TYPE int_up IS INTEGER 0 TO 100;
TYPE int_down IS INTEGER 99 TO -1;

TYPE vec IS ARRAY (3 DOWNTO -1) OF INTEGER;
TYPE vec2d IS ARRAY (0 TO 3, 5 DOWNTO 1) OF BIT;
VARIABLE bus : vec;

-- examples for some predefined attributes

a := int_up'LEFT;  -- a = 0
a := int_up'LOW;  -- a = 0
a := int_up'HIGH;  -- a = 100
a := int_down'LEFT;  -- a = 99
a := int_down'HIGH;  -- a = 99

a := vec'LEFT;  -- a = 3; note: "vec" is an array type
a := bus'RIGHT;  -- a = -1; note: "bus" is an array object
a := bus'LOW;  -- a = -1
a := vec2d'LENGTH(1);  -- a = 4, size of the first dimension of "vec2d"
a := vec2d'HIGH(2);  -- a = 5
```

E.6 base type:

```
TYPE vec IS ARRAY (3 DOWNTO -1) OF INTEGER;  -- the base type of type
                                             -- "vec" is "vec"
SUBTYPE memory IS bit_vector(1 TO 100);  -- the base type of "memory" is
                                         -- "bit_vector"
SUBTYPE pin_count IS INTEGER 1 TO 20;  -- the base type of "pin_count"
                                       -- is "INTEGER"
```

E.7 bit strings:

```
B"0110_1001";  -- (binary), length is 8, equivalent to
               -- ('0','1','1','0','1','0','0','1')
B"0110100110";  -- (binary), length is 9, equivalent to
                -- B"01_1010_0110"
X"65";  -- (hexadecimal), length is 8, equivalent to
        -- B"0110_0101"
O"126";  -- (octal), length is 9, equivalent to
         -- B"001_010_110"
```

E.8 internal block:

```
ARCHITECTURE block_struct OF test IS
   SIGNAL clock : BIT := '0';
   SIGNAL count : INTEGER := -1;
   ...
BEGIN
   alu: BLOCK
      PORT (clk IN BIT; counter : INOUT INTEGER);  -- interface of block alu
      PORT MAP(clk => clock, counter => count);  -- port map association list
      ...   -- declarations for "alu"
   BEGIN
      counter <= counter + 1;
      ...
   END BLOCK alu;
   ...
END block_struct;
```

E.9 concurrent statements:

```
ARCHITECTURE concurrent OF test IS
   SIGNAL sig, data, dum : BIT;
   PROCEDURE test_proc (val1 : IN BIT; pdata : IN BIT) IS
   BEGIN
      ...
   END test_proc;
BEGIN
   -- concurrent signal assignment
   alab1:  -- label
      sig <= '1' AFTER 10 ns, '0' AFTER 15 ns;  -- INERTIAL delay
   data <= TRANSPORT '1' AFTER 20 ns;  -- TRANSPORT delay

   data <= REJECT 5 ns INERTIAL '1' AFTER 10 ns, '0' AFTER 15 ns;  -- same as
               -- INERTIAL delay, but only spikes with a pulse width less
               -- than 5 ns are deleted. NOTE: you must have a VHDL'93
               -- compliant compiler/simulator to use "REJECT"

   -- conditional signal assignment
   dum <= '1' AFTER 10 ns, '0' AFTER 15 ns WHEN data_i = 100 ELSE
          '1' AFTER 20 ns WHEN data_i = 100 AND data_i = 99 AND sig = '0' ELSE
          '0' AFTER 100 ns;

   -- selected signal assignment
   WITH data_i SELECT
```

```
      '1' AFTER 10 ns, '0' AFTER 15 ns WHEN 1 | 10 , -- assign waveform if
                -- "data_i" = 1 or "data_i" = 10
      '1' AFTER 20 ns WHEN 100,
      '0' AFTER 2 ns WHEN OTHERS;  -- default assignment

   -- process statement
   pname1: PROCESS
      VARIABLE count : INTEGER := 0;
   BEGIN
      COUNT := COUNT + 1;
      ...
      WAIT ON data;
   END PROCESS;

   -- concurrent assertion statement
   ASSERT dum = '1'
      REPORT "signal dum is '0'"
      SEVERITY WARNING;

   -- concurrent procedure call
   test_proc (val1=>sig, pdata=>data);

END concurrent;
```

E.10 default expression:

```
ENTITY test IS
   GENERIC (def_val : BIT := '1');  -- default value of "def_val" is '1'
   PORT (clk : IN BIT := '0'; data : INOUT BIT := def_val);  -- default value
                -- of "clk" is '0', the default value of "data" is "def_val".
                -- Note, "def_val" is declared in the generic clause above!
END test;

ARCHITECTURE block_struct OF test IS
   SIGNAL clock, reset_pin : BIT := '0';
   SIGNAL count : INTEGER := -1;
   SIGNAL bus : bit_vector(0 TO 7) := (4=>'1', OTHERS=>'0');  -- default value
                -- of "bus" is B"0000_1000"
   ...
BEGIN
   ...
END block_struct;
```

E.11 deferred constant:

```
PACKAGE pack IS
   CONSTANT c_normal : INTEGER := 100;  -- normal constant
   CONSTANT c_deffered : INTEGER;  -- deferred constant
END pack;

PACKAGE BODY pack IS
   CONSTANT c_deffered : INTEGER := -99;  -- full declaration of constant
                                          -- "c_deffered"
END pack;
```

E.12 incomplete type declaration:

```
-- example of a recursive type
TYPE mytype;  -- incomplete type declaration
TYPE link_mytpe IS ACCESS mytype;  -- define an access type for "mytype"

TYPE mytpye IS RECORD
   next : link_mytype;
   data : INTEGER;
END RECORD;
```

E.13 enumeration type:

```
TYPE colour IS (red, yellow, green);
TYPE four_state IS (’0’, ’L’, ’Z’ ’X’);
```

E.14 integer type:

```
TYPE state IS RANGE 0 TO 32;
TYPE bit_index IS RANGE 31 DOWNTO 0;
```

E.15 inertial delay:

```
SIGNAL data : INTEGER;

-- assume the actual projected output waveform of signal "data" is
--   (4, 8 ns) (12, 10 ns) (-1, 15 ns) (12, 18 ns) (100, 25 ns),
--    |   |
--    |  time
--   value
-- then the driver list after executing

   data <= 12 AFTER 11 ns, 100 AFTER 15 ns;

-- at simulation time 3 ns evaluates to
--   (12, 10 ns) (12, 14 ns = 3 ns + 11 ns) (100, 18 ns)
```

E.16 transport delay:

```
SIGNAL data : INTEGER;

-- assume the actual projected output waveform of signal "data" is
--   (4, 8 ns) (12, 10 ns) (-1, 15 ns) (12, 18 ns) (100, 25 ns),
--    |   |
--    |  time
--   value
-- then the driver list after executing
```

```
   data <= TRANSPORT 12 AFTER 11 ns, 100 AFTER 15 ns;

-- at simulation time 3 ns evaluates to
--   (4, 8 ns) (12, 10 ns) (12, 14 ns = 3 ns + 11 ns) (100, 18 ns)
```
