# False Paths Sprint-2 IEEE 1076-2019 VHDL Manual Review

Mert Ecevit

Edited at 05.12.2024

**Review and Taken Notes of the Document**

*Project Leader: Koray Karakurt*

# Chapter 3 Review and Notes

## 3.2 Entity Declarations

- The entity declaration provides an external view of a component but does not provide information about how a component is implemented.
- An entity declaration defines the interface between a given design entity and the environment in which it is used.
- A given entity declaration may be shared by many design entities, each of which has a different architecture.
- The entity declarative part of a given entity declaration declares items that are common to all design entities whose interfaces are defined by the given entity declaration.
- The entity statement part contains concurrent statements that are common to each design entity with this interface.
- All entity statements shall be passive. Such statements may be used to monitor the operating conditions or characteristics of a design entity.

## 3.3 Architecture Declarations

- An architecture provides an “internal” view of an entity. An entity may have more than one architecture.
- It defines the relationships between the inputs and the outputs of a design entity which may be expressed in terms of:
  - Behavioural style
  - Data flow style
  - Structural style
- An architecture determines the function of an entity.
- It consists of a declaration section where signals, types, constants, components, and subprograms are declared, followed by a collection of concurrent statements.
- More than one architecture body may exist corresponding to a given entity declaration.
- Each declares a different body with the same interface; thus, each together with the entity declaration represents a different design entity with the same interface.
- The architecture statement part contains statements that describe the internal organization and/or operation of the block defined by the design entity.
- All of the statements in the architecture statement part are concurrent statements, which execute asynchronously with respect to one another.

## 3.4 Configuration Declarations

- During the design process, a designer may want to experiment with different variations of a design by selecting different architectures.
- Configurations can be used to provide fast substitutions of component instances of a structural design.

### 3.4.1 Block Configurations

- Blocks are used to organise a set of concurrent statements hierarchically.

### 3.4.2 Component Configurations

- A component declaration is required to make a design entity useable within the current design.

# Chapter 4 Review and Notes

## 4.2 Subprogram Declarations

- Subprograms consist of procedures and functions that can be invoked repeatedly from different locations in a VHDL description
- VHDL provides two kinds of subprograms : Procedures and Functions
- Note: Different from 1076-2002 Document The properties of implicitly declared subtypes denoted by return identifier added in this section.

## 4.3 Subprogram Bodies

- A subprogram body specifies the execution of a subprogram
- The declaration of subprogram is optional, in absence of that subprogram specification acts as declaration
- It is an error if subprogram declarative part declares a shared variable.

## 4.4 Subprogram Instantiation Declarations

- **Note :** This section is is absent in 1076-2002 Document
- The uninstantiated subprogram name shall denote an uninstantiated subprogram declared in a subprogram declaration
- Subprogram instantiations is creating anonymous subprograms which allows you to declare and use subprograms directly within the body of another construct, such as an architecture, without needing a separate named declaration.

## 4.5 Subprogram Overloading

- If the two subprogram names match, the parameter set/return values have to differ. This is called overloading and is allowed for all subprograms. It is especially useful when applied to operators, which can be seen as functions with a special name
- This allows, for example, to use the conventional '+' symbol for the addition of integer values and, likewise, with bit vectors that should be interpreted as numbers

### 4.5.3 Signatures

- A signature distinguishes between overloaded subprograms and enumerationliterals based on their parameter and result type profiles. A signature may be used in an attribute name, entity designator or alias declaration.

## 4.6 Resolution Functions

- A resolution function defines how values from multiple sources, multiple drivers, are resolved into a single value.
- A type or a signal may be defined to have a resolution. This type or signal uses the resolution functions when there are multiple drivers.

## 4.7 Package Declarations

- The primary purpose of a package is to collect elements that can be shared among two or more design units.
- It contains some common data types, constants, and subprogram specifications.
- A package declaration declares all the names of items that will be seen by the design units that use the package

## 4.8 Package Bodies

- A package body contains the implementation details of the subprograms declared in the package declaration.
- A package body is not required if no subprograms are declared in a package declaration.
- The separation between package declaration and package body serves thesame purpose as the separation between the entity declaration and architecture body.

## 4.9 Package Instantiation Declarations

- **Note :** This section is added in VHDL 1076-2008 document
- A package with a generic list is called an uninstantiated package
- The uninstantiated package serves as a form of template that we must instantiate separately.
- Uninstantiated packages and package instances that are declared as design units and stored in a design library, they can be written as a further form of design unit

## 4.10 Conformance Rules

- Two specifications for one subprogram are conform if
  - A numeric literal can be replaced by a different numeric literal if and only if both have the same value.
  - A simple name can be replaced by an expanded name in which this simple name is the suffix if, and only if, at both places the meaning of the simple name is given by the same declaration.

  both specifications are made up of the same sequence of lexical elements and corresponding lexical elements have the same meaning

# Chapter 5 Review and Notes

Data types are crucial to the modeling and manipulation of data in the hardware design in VHDL

## 5.2 Scalar Types

- 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.

### 5.2.2 Enumeration types

- In VHDL, enumeration data types let you provide a collection of named values.

### 5.2.3 Integer Types

- The signed integers that fall inside a given range are represented by the integer data type

### 5.2.4 Physical Types

- Physical type is a numeric scalar that represents some quantity

### 5.2.5 Floating-Point Types

- Floating point type provides an approximation of the real number value.

## 5.3 Composite Types

- A composite type object is one having multiple elements

### 5.3.2 Array Type

- Collections of the same data type elements are referred to as arrays in VHDL

### 5.3.3 Record Type

- Records define custom data types that contain multiple fields of different data types

## 5.4 Access Types

- Access type allows to manipulate data, which are created dynamically during simulation and which exact size is not known in advance

## 5.5 File Types

- A type that provides access to objects containing a sequence of values of a given type

## 5.6 Protection Types

- Types which implements instantiatiable regions of sequential statements and have exclusive access to shared data

## 5.7 String Representation

- Character arrays are called strings. Used frequently to manipulate files or textual data.

## 5.8 Unspecified Types

- An unspecified type abstractly defines the type class for generics, ports, parameters, and external names

# Chapter 6 Review and Notes

- Various declarations may be used in various design units.

## 6.2 Type Declarations

- Custom structures and types for more accurate data representation are made possible by type declarations, which establish new data types in VHDL.

## 6.3 Subtype Declarations

- A type together with a constraint.

## 6.4 Objects

- Object are constants, signals and variables.

### 6.5 Interface Declarations

- Interface declarations define interface objects of a precisely defined type.
- Interface objects are interface constants, interface signals, interface variables and interface files.

### 6.6 Alias Declarations

- An alias is an alternative name for an existing object (signal, variable or constant). They don't define a new object.

### 6.7 Attribute Declarations

- Attributes are user-defined properties attached to types, signals, entities, etc. They can store metadata or control simulation aspects.

### 6.8 Component Declarations

- Components act as templates for modules or logic blocks that can be instantiated multiple times in a design. This allows reusability and modular design.

# 1 Chapter 7 Review and Notes

- In VHDL, Specifications are used to define additional information associated with entity in a design.

### 7.2 Attribute Specifications

- Attributes are a feature of VHDL that allow you to extract additional information about an object.

### 7.3 Configuration Specifications

- A VHDL configuration specification defines how instances of a specific component are connected and used in a design.
- It associates component instances with binding information, determining the role of the component within the design structure.

### 7.4 Disconnection Specification

- A disconnection specification defines the time delay after which the driver of a guarded signal will automatically disconnect. It controls how long a signal remains active before it is disconnected.

# Chapter 8 Review and Notes

- In code, names refer to various entities like objects, values, methods, attributes, or slices.

## 8.2 Simple Names

- While selected names denote a sub-element, simple names explicitly identify an item.

## 8.3 Selected Names

- When referring to an entity that is declared within another entity or a design library, selected names are utilized.

## 8.4 Indexed Names

- The indexed name indicates an element of an array, which is indicated by an expression list forming the array index.

## 8.5 Slice Names

- The slice name allows indicating a part of a one-dimensional array. The slice name is called a null slice if its discrete range is a null range

## 8.6 Attribute Names

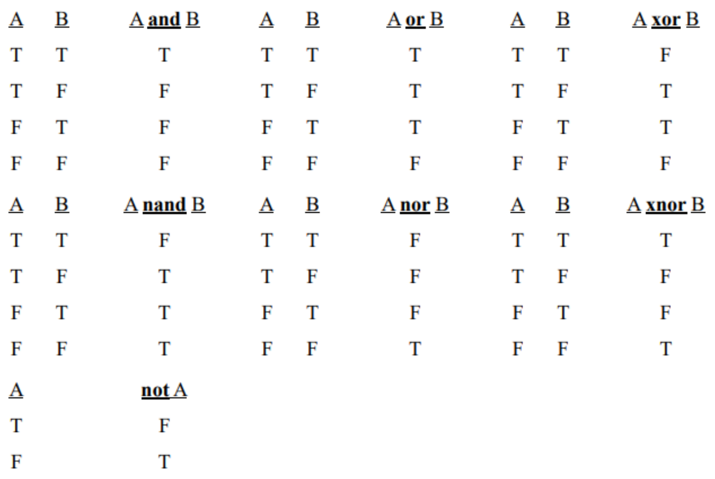
- The attribute name consists of two elements: a prefix and an attribute designator

## 8.7 External Names

- An object designated in the design hierarchy that contains the external name is indicated by its external name.

# Chapter 9 Review and Notes

- A formula that defines the computation of a value.
- The type of an expression depends only upon
  - The types of its operands
  - On the operators applied
- The operators that can be used in expressions are defined below:

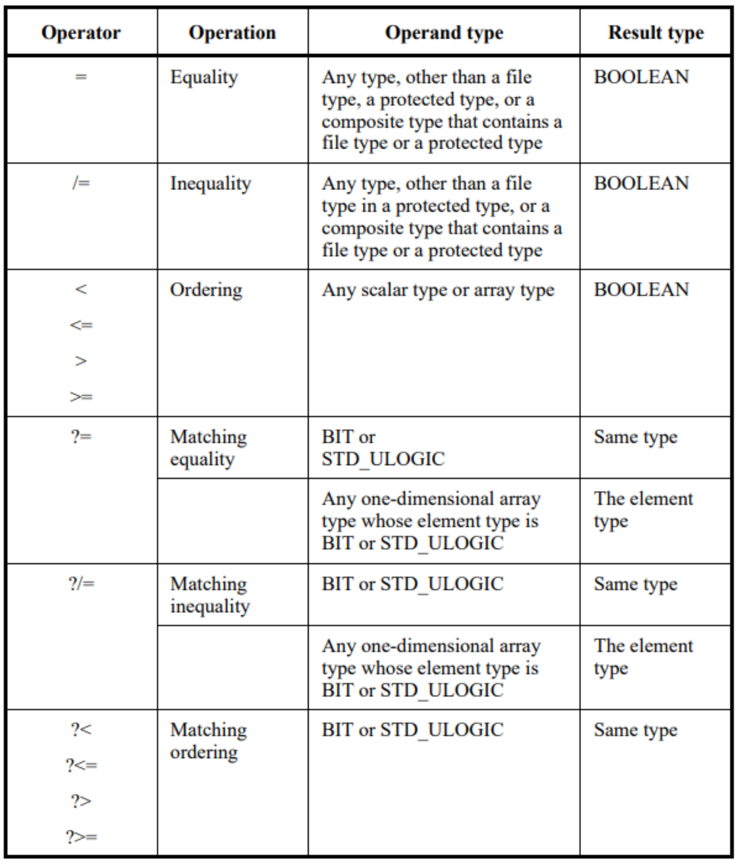


| A | B | A **and** B | A | B | A **or** B | A | B | A **xor** B |
|---|---|---|---|---|---|---|---|---|
| T | T | T | T | T | T | T | T | F |
| T | F | F | T | F | T | T | F | T |
| F | T | F | F | T | T | F | T | T |
| F | F | F | F | F | F | F | F | F |

| A | B | A **nand** B | A | B | A **nor** B | A | B | A **xnor** B |
|---|---|---|---|---|---|---|---|---|
| T | T | F | T | T | F | T | T | T |
| T | F | T | T | F | F | T | F | F |
| F | T | T | F | T | F | F | T | F |
| F | F | T | F | F | T | F | F | T |

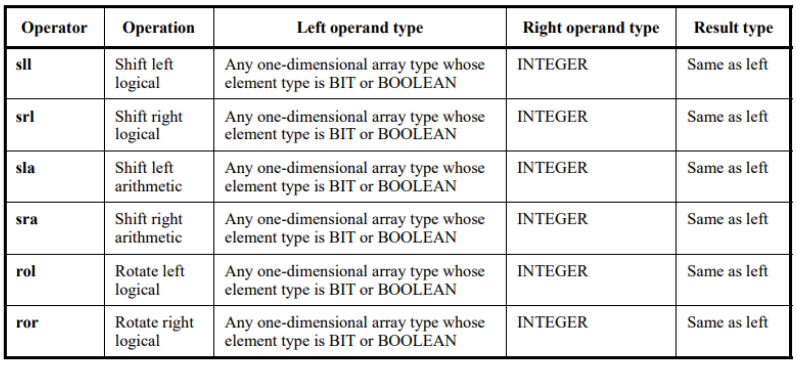
| A | **not** A |
|---|---|
| T | F |
| F | T |

Figure 1:



| Operator | Operation | Operand type | Result type |
|---|---|---|---|
| = | Equality | Any type, other than a file type, a protected type, or a composite type that contains a file type or a protected type | BOOLEAN |
| /= | Inequality | Any type, other than a file type in a protected type, or a composite type that contains a file type or a protected type | BOOLEAN |
| < <= > >= | Ordering | Any scalar type or array type | BOOLEAN |
| ?= | Matching equality | BIT or STD_ULOGIC | Same type |
| | | Any one-dimensional array type whose element type is BIT or STD_ULOGIC | The element type |
| ?/= | Matching inequality | BIT or STD_ULOGIC | Same type |
| | | Any one-dimensional array type whose element type is BIT or STD_ULOGIC | The element type |
| ?< ?<= ?> ?>= | Matching ordering | BIT or STD_ULOGIC | Same type |

Figure 2:



| Operator | Operation | Left operand type | Right operand type | Result type |
|---|---|---|---|---|
| **sll** | Shift left logical | Any one-dimensional array type whose element type is BIT or BOOLEAN | INTEGER | Same as left |
| **srl** | Shift right logical | Any one-dimensional array type whose element type is BIT or BOOLEAN | INTEGER | Same as left |
| **sla** | Shift left arithmetic | Any one-dimensional array type whose element type is BIT or BOOLEAN | INTEGER | Same as left |
| **sra** | Shift right arithmetic | Any one-dimensional array type whose element type is BIT or BOOLEAN | INTEGER | Same as left |
| **rol** | Rotate left logical | Any one-dimensional array type whose element type is BIT or BOOLEAN | INTEGER | Same as left |
| **ror** | Rotate right logical | Any one-dimensional array type whose element type is BIT or BOOLEAN | INTEGER | Same as left |

Figure 3:

## 9.3 Operands

- The operands in an expression include names (that denote objects, values, or attributes that result in a value), literals, aggregates, function calls, qualified expressions, type conversions, and allocators. In addition, an expression enclosed in parentheses may be an operand in an expression.

## 9.4 Static Expressions

- Utilized in constant declarations or restrictions; fully computed at compile time.
- There are two categories of static expression:
  - If every operator in expression is an implicit defined operator, these are called **locally static** expressions
  - If every in the expression is a pure function and every primary in the expression is a global static primary, they are called **globally static** expressions

## 9.5 Universal Expressions

- A universal expression is either an expression that delivers a result of type universal integer or one that delivers a result of type **universal real**
- The same operations are predefined for the type **universal integer** as for any integer type
- The same operations are predefined for the type **universal real** as for any floating point type
- In addition, these operations include the multiplication and division operators

# Chapter 10 Notes and Review

## 10.2 Wait Statement

- Causes execution of sequential statements to wait.

- Includes optional sensitivity clause, condition clause, or timeout clause.

### 10.3 Assertion Statement

- Used for internal consistency checks or error message generation.
- Can include an optional report string and severity name.
- Predefined severity names: NOTE, WARNING, ERROR, FAILURE.
- Default severity for assert is ERROR.
- If the condition is not met, we say that an assertion violation has occurred.

### 10.4 Report Statement

- Used to output messages.
- Default severity name is NOTE.

### 10.5 Signal Assignment Statement

- Typically considered a concurrent statement but can be used sequentially.
- Target is updated following specific delay mechanisms and waveforms.
- Delay mechanisms include transport, reject, and inertial.
- Waveforms can consist of values or nulls with optional delays.

### 10.6 Variable Assignment Statement

- Assigns the value of an expression to a target variable.

### 10.7 Procedure Call Statement

- Used to call a procedure.
- Can include actual parameters with positional or named associations.

### 10.8 If Statement

- Implements conditional structures.
- Can include optional `elsif` and `else` clauses.

### 10.9 Case Statement

- Executes one specific case of an expression equal to a choice.
- Choices must be constants of the same discrete type as the expression.
- Can include an optional `when others` clause if all choices are not covered.

## 10.10 Loop Statement

- Supports three kinds of iteration statements: simple, `for`, and `while`.
- All loop types may include `next` and `exit` statements.

## 10.11 Next Statement

- Used in a loop to cause the next iteration.
- May include an optional condition.

## 10.12 Exit Statement

- Used in a loop to immediately exit the loop.
- May include an optional condition.

## 10.14 Return Statement

- Required in a function and optional in a procedure.
- May include an optional expression to return a value.

## 10.15 Null Statement

- Used when a statement is needed but there is nothing to do.

## 10.16 Sequential Block Statement

- A sequential block statement encloses a sequence of sequential statements. Sequential block statements may be nested.

# Chapter 11 Review and Notes

- The VHDL language models real systems as a set of subsystems that operate concurrently.
- Concurrent statements are used to define interconnected blocks and processes that jointly describe the overall behavior or structure of a design.

## 11.2 Block Statement

- A block statement defines an internal block representing a portion of a design
- Using of the block statement is supplying the limit the scope of the variables used within a portion of the code.
- This can significantly improve code readability in a large design where signals can be declared and utilized in the same region

## 11.3 Process Statement

- Processes are composed of sequential statements, but process declarations are concurrent statements.
- It is executed in parallel with other processes.
- It cannot contain concurrent statements.
- It allows access to signals defined in the architecture in which the process appears and to those defined in the entity to which the architecture is associated.

## 11.4 Concurrent Procedure Call Statements

- It is a shorthand notation commonly used for processes.
- The body contains a sequential procedure but it is different from it, it is a concurrent statement.

## 11.5 Concurrent Assertion Statements

- A concurrent assertion statement represents a passive process statement containing the specified assertion statement.
- The syntax appears to be sequential assertion statement but it is a concurrent statements like concurrent procedure call statements

## 11.6 Concurrent Signal Assignment Statements

- Concurrent signal assignment statements are equivalent to sequential signal assignments contained in process statements
- Unlike ordinary signal assignments, concurrent signal assignment statements can be included in the statement part of an architecture body.

## 11.7 Component Instantiation Statements

- Component instantiation allows you to instantiate one design unit (component) inside another design unit to create a hierarchically structured design description.
- To perform component instantiation:
  - Create the design unit (entity and architecture) modeling the functionality to be instantiated.
  - Declare the component to be instantiated in the declarative region of the parent design unit architecture.
  - Instantiate and connect this component in the architecture body of the parent design unit.
  - Map (connect) formal ports of the component to actual signals and ports of the parent design unit.

## 11.8 Generate Statements

- Generate statements are used to accomplish one of two goals:
  - Replicating Logic in VHDL
  - Turning on/off blocks of logic in VHDL
- The generate keyword is always used in a combinational process or logic block.
- It should not be driven with a clock.
- If the digital designer wants to create replicated or expanded logic in VHDL, the generate statement with a for loop is the way to accomplish this task.
- The second use case is very handy for debugging purposes, or for switching out different components without having to edit lots of code.
- The designer needs to ensure that these generate blocks are mutually exclusive, such that no two can be active at the same time

# Chapter 12 Review and Notes

## 12.1 Declerative Region

- A declarative region in VHDL is a defined area where specific declarations are valid
- Declarative regions are established by constructs like entity, architecture, process, block, package, component, and configuration, each forming its own scope

## 12.2 Scope of Declarations

- A declaration's scope begins where it is declared and extends to the end of its nearest declarative region, defining where it can be used.
- The immediate scope covers the area from the start of the declaration to the end of its nearest declarative region
- Certain declarations, like those in package, record, subprogram parameters, or components (generic and port), can extend beyond their immediate scope, applying throughout their entire enclosing region or library

## 12.3 Visibility

- Visibility rules define which identifier is accessible within a certain area, and overloaded names are clarified by additional rules
- Visibility can be achieved by selection (using use clauses) or directly, where identifiers are accessible in their immediate area unless hidden by another declaration with the same name

## 12.4 Use Clause

- The use clause makes names defined in a library directly visible within another region of the VHDL code
- The visibility of a use clause starts right after it is written and continues to the end of its surrounding region, such as a design unit or configuration

## 12.5 The context of overload resolution

- Overload resolution defines the exact meaning of an identifier or operator when multiple meanings are possible, ensuring a single valid interpretation within a specific context.

# Chapter 13 Review and Notes

## 13.1 Design Units

- Certain constructs are independently analyzed and inserted into a design library; these constructs are called design units

## 13.2 Design Libraries

- The results of a VHDL compilation (analyze) are kept inside of a library for subsequent simulation, for use as a component in other designs

## 13.3 Context Declarations

- A context declaration defines context items that may be referenced by design units.

## 13.4 Context Clauses

- A context clause defines the initial name environment in which a design unit is analyzed.

## 13.5 Order of Analysis

- The rules defining the order in which design units can be analyzed are direct consequences of the visibility rules.
- Once you have analyzed all of the design units you need for a simulation run into a VHDL library, you can run the simulation

# Chapter 14 Review and Notes

- Definition of Elaboration of a block header:
  - Purpose: Sets up the interface for each design block, including generics and ports

  - Generics: Assigns values to parameters that control block behavior, like bit widths or timing
  - Ports: Connects inputs and outputs to external signals or expressions

- Definition of Elaboration of a declarative part:
  - Purpose: Prepares internal elements like signals, constants, and variables in each block
  - Declarations: Initializes constants, allocates memory for variables, and sets up signals

- Definition of Elaboration of a statement part:
  - Purpose: Prepares executable statements for simulation, including both concurrent and sequential statements
  - Concurrent Statements: Includes signal assignments and processes that run simultaneously
  - Sequential Statements: Ordered operations within processes or subprograms

- Definition of Dynamic elaboration:
  - Purpose: Manages runtime scenarios like deferred binding or conditional elaboration
  - Deferred Binding: Allows components to be bound conditionally, adapting to runtime states

- Definition of Execution of a model:
  - Purpose: Simulates the design by executing concurrent and sequential statements over time

## Chapter 15 Review and Notes

- VHDL divides two main groups: graphic characters and format effectors
  - Graphic Characters: Readability for users and compilers
  - Format Effectors: For visual layout (line breaks, tabs, and new lines

## 15.3 Lexical Elements

Seperators:

- Separates lexical elements
- A separator is either a space character(SPACE or NBSP), a format effector, or the end of a line

Delimiters:

- Delimiter characters separate different elements in the syntax of the language

- The delimiters in VHDL are generally specific characters, and each one performs a specific function

Identifiers:

- Basic identifiers ;
  - Define signals, ports, and variables
  - Can Contain Letters, Digits, Underscores
- Extended Identifiers
  - Define signals, ports, and variables
  - Case Sensitive
  - Enclosed in Backslashes
  - Enable the Inclusion of Special Characters

Literals:

- Abstract Literals: Represent values in basic numerical types.
- Decimal Literals: Express numerical values (integer or fractional)
- Based Literals: Specify numerical values in a specific base (binary, octal, hexadecimal).
- Bit String Literals: Define numerical systems with specific bit sequences (B, O, X)
- Character Literals: Represent a single character
- String Literals: Represent a sequence of characters

# Chapter 16 Review and Notes

- This section describes predefined attributes of VHDL, and the packages that all VHDL implementations shall provide.

## Predefined attributes

- Predefined attributes denote values, functions, types, subtypes, mode views, signals and ranges associated with various kinds of named entities.

## Active

- When applied to a signal, this attribute works like a function call, returning true if s is active during the current simulation cycle and false if not

## Ascending

- This attributes can be applied to scalar types or objects of them, including subtypes and aliases. It returns a boolean value that will be true if p has ascending range or false if it’s descending.

## Base

- When applied to an object, type, or subtype p, the base attribute returns the underlying type from which it originates. If there are multiple layers ıf subtypes, you get the root type that's not a subtype.
- This attribute cannot used standalone. It must appear in conjuction with a second attribute.

## Converse

- In VHDL2019 adds mode views to interfaces and this attribute along with them.

## Delayed

- When applied to a signal, this attribute produced a signal that's a copy of s, but with the transitions delayed by time units. The derived signal will have a delay of one delta cycle if the optional argument is omitted.

## Designed-subtype

- When called on access type or file type, the designated-subtype attribute returns the subtype that the object references.

## Driving

- This attribute acts as function returning a boolean value when applied to a signal. You can only use this attribute from within a process or equivalent concurrent statement/subprogram. It will return true if the process is driving the signal and false otherwise.
- When used in a regular signal, this attribute always returns true. That's because a process controlling a signal will always be driving it. However, that's not always the case when it comes guarded signals.

## Driving-value

- When applied to a signal, this attribute works like a function call, returning the value that the enclosing process is driving onto the signal.
- Only used with a process or subprogram.

## Element

- This attributes can only be applied to an array type or a signal or variable of an array type. When applied to such an object, it returns the subtype of the array elements.
- Can be used for declare new objects.

## Event

- When applied to a signal, this attribute works like a function call, returning true if an event occured on during the current simulation cycle and false if not.

## High

- This attribute works like a function returning the upper bound of the object it's attached to. It may be used on any object whose range is contrained.

## Left

- This attribute works like a function returning the left bound of the object it's attached to. It may be used on any object whose is constrained

## Low

- This attribute works like a function returning the left bound of the object it's attached to. It may be used on any object whose range is constrained.

## Quiet

- When applied to a signal this attribute produces a drived signal of boolean type. The resulting signal's value is true if signal was quiet for time before the current simulation time. Otherwise it has the value false.

## Right

- This attribute works like a function returning the right bound of the object it's attached to. It may be used on any object whose range is constrained.

## Stable

- When applied to a signal, this attribute produces a derived signal of boolean type, The resulting signal's value is falsse if there were events on signal for time before the current simulation time. If there were no events, it is true