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
T	T	T
T	F	F
F	T	F
F	F	F

A	B	A **or** B
T	T	T
T	F	T
F	T	T
F	F	F

A	B	A **xor** B
T	T	F
T	F	T
F	T	T
F	F	F

A	B	A **nand** B
T	T	F
T	F	T
F	T	T
F	F	T

A	B	A **nor** B
T	T	F
T	F	F
F	T	F
F	F	T

A	B	A **xnor** B
T	T	T
T	F	F
F	T	F
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