**Chapter 8: Names**

Chapter 8 focuses on the concept of names in VHDL, which are identifiers used to refer to objects, types, subprograms, and other design elements. Names ensure clarity, organization, and accessibility in a VHDL design.

**8.1 General**

* Purpose: Names uniquely identify design elements and enable referencing them in expressions, statements, and configurations.
* Types of Names: Includes simple names, selected names, indexed names, slice names, attribute names, and external names.

**8.2 Simple Names**

* Refer to declared entities within a specific scope**.**



**8.3 Selected Names**

* Refer to elements within a hierarchical design or package.
* Syntax: <prefix>.<suffix>

A close-up of a standard

Description automatically generated

**8.4 Indexed Names**

* Access individual elements in arrays or vectors using an index.
* Syntax: <array\_name>(<index>)



**8.5 Slice Names**

* Refer to a subset of an array or vector.
* Syntax: <array\_name>(<range>)

A screenshot of a computer error

Description automatically generated

**8.6 Attribute Names**

* Refer to attributes of objects, such as predefined attributes ('event, 'length, etc.) or user-defined attributes.
* Syntax: <object>'<attribute>

A white background with black text

Description automatically generated

**8.7 External Names**

* Refer to objects or entities external to the current design unit, useful for modular and hierarchical designs.



**From :** <https://adaptivesupport.amd.com/s/question/0D54U00007hX4mrSAC/vhdl-external-name-alias-for-internal-array-signal-not-being-resolved?language=en_US>

**Chapter 9: Expressions**

Chapter 9 focuses on expressions in VHDL, which are combinations of operators and operands used to compute values, control behavior, or specify conditions. Expressions are fundamental for describing hardware behavior.

**9.1 General**

Expressions: Formed by combining operands (constants, variables, signals, or literals) with operators.

Usage: Used in assignments, conditions, and computations throughout a design.

**9.2 Operators**

Operators define operations on operands. They are categorized as:

1. **Arithmetic Operators**:
   * Perform mathematical operations.
   * Examples: +, -, \*, /, mod, rem.
2. **Relational Operators**:
   * Compare two operands and return a boolean.
   * Examples: =, /=, <, <=, >, >=.
3. **Logical Operators**:
   * Perform bitwise or boolean operations.
   * Examples: and, or, nand, nor, xor, not.
4. **Shift Operators**:
   * Shift the bits of operands.
   * Examples: sll, srl, sla, sra, rol, ror.
5. **Concatenation Operator**:
   * Combines elements into an array or string.
   * Example: &.
6. **Miscellaneous Operators**:
   * Include unary operators like + and - for sign operations.

**9.3 Operands**

Operands are the entities acted upon by operators:

* **Literals**: Constant values like 10, '1', or "1010".
* **Names**: References to signals, variables, or constants.
* **Function Calls**: Results of subprogram executions.
* **Qualified Expressions**: Specify the type of a literal or expression.

**9.4 Static Expressions**

* Computed entirely at compile-time.
* Used in constant declarations or constraints.

A white rectangular object with black text

Description automatically generated

**9.5 Universal Expressions**

* Expressions involving universal types like integers or real numbers.
* Automatically converted to specific types when necessary.

**Chapter 10: Sequential Statements**

Chapter 10 describes sequential statements in VHDL, which are used to control the flow of execution in processes, functions, and procedures.

**10.1 General**

* Sequential Statements: Specify the behavior of a design in processes or subprograms.
* Execute in a defined sequence, unlike concurrent statements.

**10.2 Wait Statement**

* Suspends execution of a process until a condition is met.

**10.3 Assertion Statement**

* Verifies conditions during simulation and reports errors or warnings if conditions are false.

**10.4 Report Statement**

* Displays messages during simulation for debugging.

A white background with black text

Description automatically generated

**From:** <https://peterfab.com/ref/vhdl/vhdl_renerta/mobile/source/vhd00007.htm>

**10.5 Signal Assignment Statement**

* A white screen with black text

  Description automatically generatedAssigns values to signals

**10.6 Variable Assignment Statement**

* Updates the value of a variable immediately.
* Updates the value of a variable immediately.

**10.7 Procedure Call Statement**

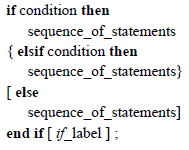
* Invokes a procedure to perform a specific task.

A computer code with black text

Description automatically generated

**From:** <https://peterfab.com/ref/vhdl/vhdl_renerta/mobile/source/vhd00052.htm>

**10.8 If Statement**

* Conditional execution of statements.

**10.9 Case Statement**

* A screenshot of a computer program

  Description automatically generatedExecutes one of several branches based on a value.

**10.10 Loop Statement**

* A close up of a text

  Description automatically generatedRepeats a set of statements

**10.11 Next Statement**

* A white screen with black text

  Description automatically generatedSkips the remaining statements in the current iteration of a loop.

**From:** https://ics.uci.edu/~jmoorkan/vhdlref/nexts.html

**10.12 Exit Statement**

* A white background with black text

  Description automatically generatedTerminates a loop prematurely.

**From:** <https://peterfab.com/ref/vhdl/vdlande/exits.html>

**10.13 Return Statement**

* Exits a subprogram and optionally returns a value.

A screenshot of a computer code

Description automatically generated

**From:** <https://docs.amd.com/r/en-US/ug901-vivado-synthesis/Relaxed-Return-Rules-for-Function-Return-Values>

**10.14 Null Statement**

* A black text with white text

  Description automatically generated with medium confidencePlaceholder for no operation.

From: <https://peterfab.com/ref/vhdl/vhdl_renerta/source/vhd00045.htm>

**Chapter 17: VHDL Procedural Interface (VHPI) Overview**

Chapter 17 introduces the **VHDL Procedural Interface (VHPI)**, a standardized mechanism to integrate external software tools and programs with VHDL simulation environments. VHPI is essential for extending VHDL's functionality, enabling interaction with external systems for simulation, analysis, or tool integration.

**17.1 General**

* VHPI provides a procedural (function-call) interface for accessing and controlling VHDL simulation environments.
* **Purpose**:
  + Enhance interoperability between VHDL simulators and external software tools.
  + Allow runtime manipulation and observation of VHDL models.

**17.2 Organization of the Interface**

* VHPI is organized into a well-defined hierarchy and categorized into capability sets, access functions, and handles.
* Key components:
  1. **Capability Sets**: Define what functionalities a simulator provides.
  2. **Handles**: Represent VHDL objects or simulator elements (e.g., signals, variables, or processes).
  3. **Access Functions**: Facilitate interactions with the simulation environment.

**17.3 Capability Sets**

* Capabilities are sets of features that a simulator supports, allowing users to query its compatibility and functionality.
* Examples of capabilities:
  + Reading and writing values of VHDL objects.
  + Accessing simulation time.
  + Triggering callbacks for specific events.
* Simulators may implement a subset of VHPI features, and capability queries ensure compatibility checks.

**17.4 Handles**

* Handles are the central abstraction in VHPI, representing simulation objects (signals, variables, instances, etc.).
* **Types of Handles**:
  + **Object Handles**: Represent VHDL objects like signals, constants, or variables.
  + **Entity/Instance Handles**: Represent design entities or instantiated components.
  + **Process Handles**: Represent concurrent processes in the simulation.
* **Use of Handles**:
  + Handles are retrieved using specific VHPI functions and are used to perform operations like value reading, writing, or querying properties.

**Key Concepts in VHPI**

1. **Access Functions**:
   * Provide procedural access to simulation elements through handles.
   * Allow querying, reading, or modifying simulation states dynamically.
2. **Callback Mechanism**:
   * Enable event-driven interaction.
   * Users can register functions to be executed upon specific simulation events (e.g., signal changes or process activations).
3. **Simulator Interoperability**:
   * VHPI ensures a standardized way of interacting with different simulators, improving portability of tools and extensions.

**Applications of VHPI**

1. **Debugging and Monitoring**:
   * External tools can observe signals or variables during simulation to aid debugging.
2. **Dynamic Model Interaction**:
   * Allows runtime injection of stimuli or modification of simulation states.
3. **Custom Verification**:
   * Integrate custom verification tools or frameworks with VHDL simulators.
4. **Co-Simulation**:
   * Enable integration of VHDL models with other simulation environments (e.g., system-level or mixed-signal simulation).