**LECTURE NOTE ON VHDL (Very High Speed Integrated Circuit Hardware Description Language)**

**4.1 Hardware Description Languages**

A hardware description language (HDL) is a programming language used to describe the behaviour or structure of digital circuits (ICs). HDL are also used to simulate the circuit and check its response. VHDL stands for “very high-speed integrated-circuit hardware description language”. Both VHDL and Verilog are officially endorsed IEEE (institute of Electrical and Electronics Engineers) standards. Other HDLs include JHDL (Java HDL), and proprietary HDLs such as Cypress Semiconductor Corporation’s Active-HDL. VHDL was developed under the VHSIC program of the U.S Department of Defense. It was originally intended to serve as a language to document descriptions of complex digital circuits. It was also used to describe the behaviour of digital circuits and could be fed to software tools that were used to simulate a circuit’s operation. Hardware description languages, such as VHDL are used to PLD and FPGA-based systems.

**4.2 VHDL Language**

VHDL which means [“VHSIC hardware description language” (VHSIC is “very high-speed integrated circuit”) is one of the two most popular HDLs, the other being Verilog HDL. The VHDL language is very popular for the design entry of digital circuits into CAD systems, simulation and documentation. VHDL is an extremely complex and sophisticated language, so learning it completely can be a daunting task. Note that most design can be accomplished by learning only a subset of the language.

**4.3 VHDL Programming Structure**

A VHDL program is written in a text file and has the extension “.vhd” (sometimes, “.vhdl”). VHDL is not case sensitive. Every VHDL program has associated with it an **entity.** The interface to the outside world (through pins) is described in this section. Every entity has associated with it an architecture. The architecture describes the behaviour or structure of the design coded in the VHDL program. The design units of VHDL, apart from entity and architecture, are package, package body and configuration. These are not required to be present in every VHDL design, but designers use them for a better coding style and for convenience. Each design unit of a VHDL design can be in a separate file. It is not required that an entity and the corresponding architecture be described in the same file. It is important to note that a single IC can be built from many VHDL files. A design may be built hierarchically, and also a multiplier may be built from full-adders, which in turn may be built from half-adders. One could write VHDL programs for the half-adder, put instances (copies) of the half-adder in another VHDL program to build a full-adder, and write a program to put full-adders together to make a multiplier.

Note that VHDL programming is not case sensitive; however, to identify the keywords of a VHDL program, they will be written in lowercase boldface letters. A typical VHDL program consists of a library declaration, an entity declaration, and an architecture declaration as shown in Figure 4.1.



Figure 4.1: VHDL Program Structure

Library declarations are generally included in the first lines of code, which locate the system library and the user library sources to resolve and translate the language statements within the body of the program. The IEEE standard library is often included in the VHDL program and used by the VHDL program.

**4.3.1 Entity**

The entity of a VHDL program defines the external interface to the design. The term **entity** is a keyword and cannot be used as a variable in the VHDL program. The entity defines the input and output ports of a digital system. Every VHDL program must have at least one **entity**, with a designated name. The entity declaration associates the entity with a particular design, and specifies the names of the ports, their associated data type, and the direction of the ports. The entity declaration does not describe how the design functions. Consider example 4.1, which is a logic circuit of a NAND function, which is given in Figure 4.2.

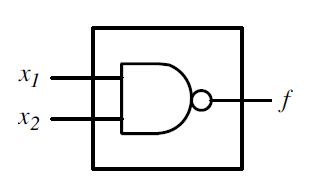


Figure 4.2: NAND Logic Circuit Input and Output Ports

The entity declaration of the NAND function is illustrated in figure 4.3. Note that only the input and the output ports of the logic circuit is identified, there is no reference to the function of the NAND circuit.

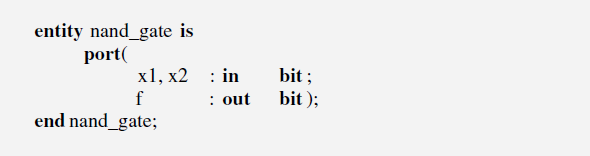


Figure 4.3: Entity Declaration of the NAND Logic Circuit

**4.3.2 Architecture**

The architecture describes the internal structure or behaviour of the corresponding entity. Note that more than one architecture may be associated with an entity, and also, in VHDL (unlike software programming languages such as C and C++), statements inside the architecture model events that happens concurrently. Statements inside a process block (which is defined inside an architecture body) model events that occur simultaneously. It should also be noted that comments in a VHDL program begins with “- -” and every line requires a separate “- -” if the comment spans more than one line.

Every architecture has a name and must include the name of the entity with which it is associated. The architecture realization of the NAND circuit is shown in Figure 4.4.

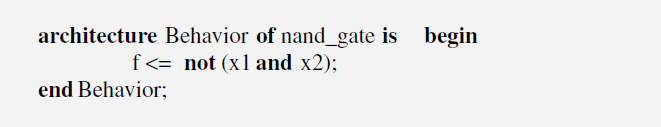


Figure 4.4: Architecture Declaration of the NAND Logic Circuit

Consider a logic circuit, which is given in Figure 4.5, the VHDL implementation of the circuit requires an entity design, which identifies the input and the output ports of the circuit, and the architecture design which identifies the logic function of the circuit.

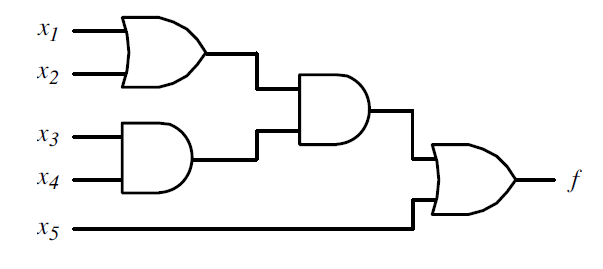


Figure 4.5: Logic Circuit Sample

The VHDL program is given in Figure 4.6, which implements the logic circuit in Figure 4.5 note that the function of the circuit is described in the architecture of the code.

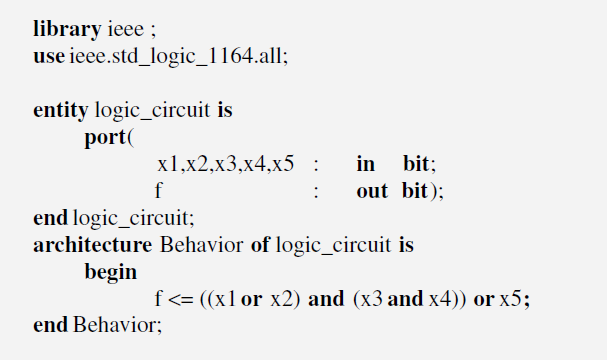


Figure 4.6: VHDL Code for the Logic Circuit

In general, computer aided tools have synthesis features that can be used to optimize the function without the user carrying out any prior simplification.

**4.4 Assignment Statements**

VHDL provides selected signal assignment statements, which will assign a signal from several values using a selection condition. Consider the logic circuit for a 2:1 multiplexer shown in Figure 4.7.

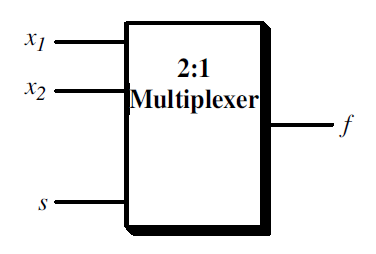


Figure 4.7: Block Diagram of a 2:1 Multiplexer

While the VHDL program is shown in Figure 4.8, which implements the 2:1 multiplexer in Figure 4.7.

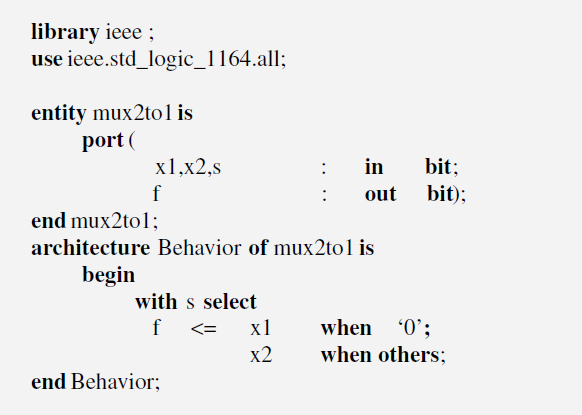


Figure 4.8: VHDL Code for the 2:1 Multiplexer

The function of the multiplexer is described in the architecture of the code using selected signal assignment statements. The signal assignment statements selected begins with the reserved keyword with followed by the selection condition, which is the input signal,. The reserved keyword when selects a possible value for the select signal . The when others (reserved keyword) is included to select the last possible value of .

**4.5 VHDL Data Types**

VHDL is a strongly typed language. This means that every object assumes the value of its nominated type. In a simple language, the data type of the left-hand side (LHS) and right-hand side (RHS) of a VHDL statement must be the same. The standard VHDL 1076 specifications describes four classes of data types.

1. **Scalar types:** this represent a single numeric value or, in the case of enumerated types, and enumeration value. The standard types that fall into this class are integer, real (floating point), physical, and enumerated. All of these basic types can be thought of as numeric values.
2. **Composite types:** which represents a collection of values. There are two classes of composite types: arrays containing elements of the same type, and records containing elements of different types.
3. **Access type:** this provides references to objects in much the same way that the pointer types are used to reference data in software programming languages.
4. **File types:** which reference objects (typically disk files) that contain a sequence of values.

The most common data types are described as follows. Data types define the set of values that an object may take. The name of the type of object must be declared before the object is used in any VHDL statement.

1. **Integer type:** integer numbers range from -2147483647 to 2147483647, as defined by the standard using a 32-bit representation. Notice that there are equal number of positive and negative integer’s number in VHDL. Example include.

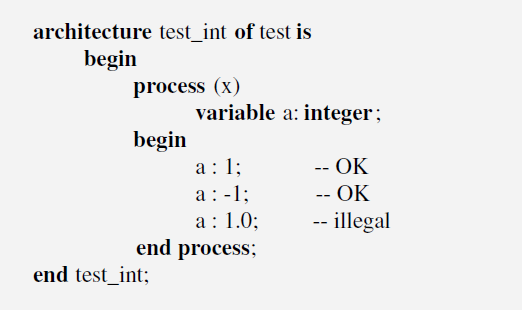


Figure 4.9: Integer Type

1. **Real type:** the real numbers range between -1.0E38 and 1.0E38.
2. **Bit and bit\_vector types:** these types are predefined in VHDL standards IEEE 1076 and IEEE 1164. Hence, no library is needed to use these data types. A bit object can take one of the values 0 and 1. An object of bit\_vector type is an array of Bit objects. Example is given in Figure 4.10 which shows how to declare and use objects of these types.

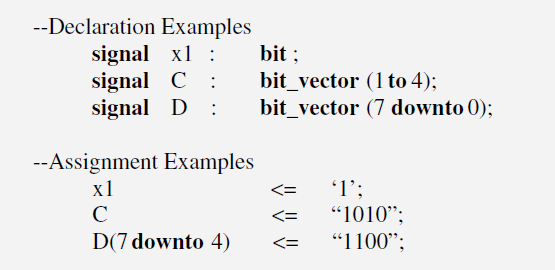


Figure 4.10: Bit and Bit\_Vector Types

1. **Boolean type:** A Boolean object can take a value that is either true or false. “True” is equal to logic 1 and “false” is equal to logic 0.
2. **Enumeration type:** the user specifies a list of possible values that an object could take. An illustration of the enumeration data type is given in Figure 4.11

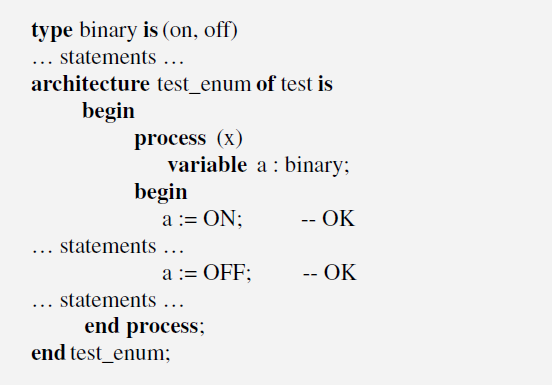


Figure 4.11: Enumeration Type

1. **Physical type:** objects with physical type require associated units. The range of units must be specified. Notice that the “time” is the only physical type object predefined in the VHDL standards. Example of physical-type definition for resistance units is given in Figure 4.12.

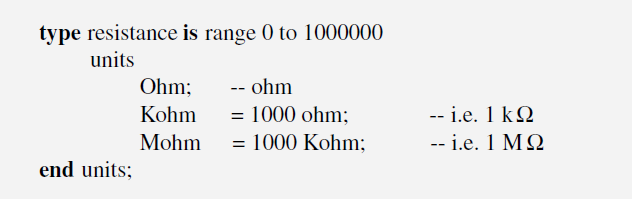


Figure 4.12: Physical Type

1. **Std\_logic and std\_logic\_vector types:** the std\_logic data is a part of IEEE standard 1164. It provides more flexibility than the Bit type. To use this type, two statements must be included in the binary declaration as shown in Figure 4.13.

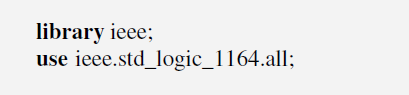


Figure 4.13: Std\_Logic and Std\_Logic\_Vector Type

By including these statements, the user will have access to the std\_logic\_vector\_1164 package, which defines the std\_logic type. The std\_logic vector is a linear array of objects of std\_logic type. The IEEE 1164 standard defines the standard type that would allow multiple values to be represented for a wire. These examples are given in Figure 4.15 which illustrate the std\_logic variable declarations and assignments using IEEE standard 1164.

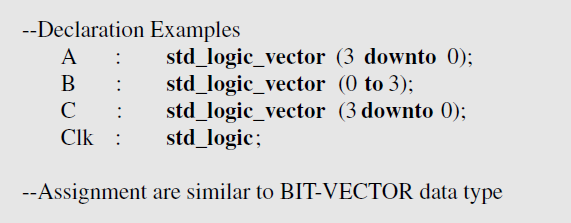


Figure 4.15: Std\_Logic Variable Declaration

1. **Array type:** the array type is used to group elements of the same data type into a single VHDL object. The array may be constrained or unconstrained. Arrays may be one or multidimensional. Figure 4.16 shows an example of array data type.

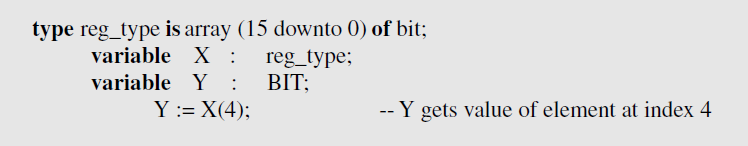


Figure 4.16: Array Type

**4.6 VHDL Operators**

VHDL provides predefined operators which are used as hardware modelling units. These include logical (or Boolean), arithmetic, and relational operators. The logical operators are given in Figure 4.17

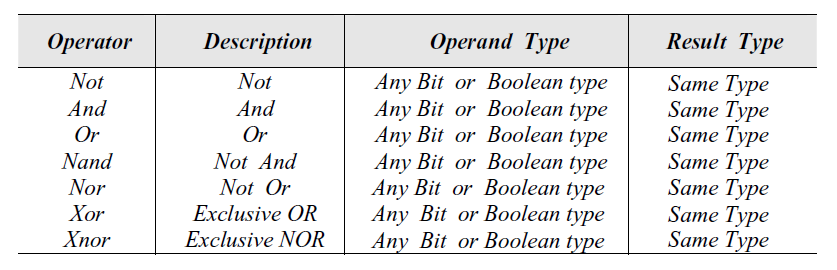


Figure 4.17: VHDL Logical (Boolean) Operators

The NOT operator has one input and one output, whereas the remaining operators are binary operators, which have two input ports and one output port. Figure 4.18 gives the VHDL relational operators.

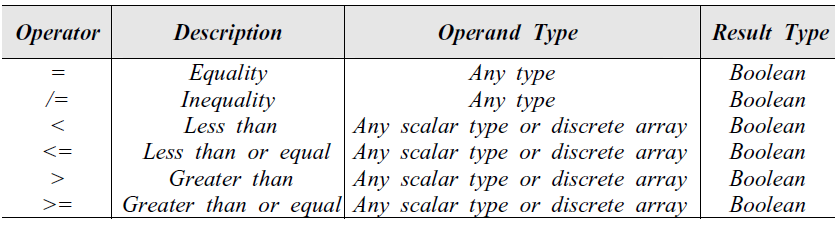


Figure 4.18: Relational Operators

These relational operators are a set of binary logical operators that generate a result of Boolean type: that is either “true” or “false”. Note that relational operands are either bit type or Boolean type, but not of mixed type.

The VHDL arithmetic operators are given in Figure 4.19. They accept operands of integer or floating-point type (real type). Note that VHDL does not accept implicit type conversion between integer and floating-point numbers.

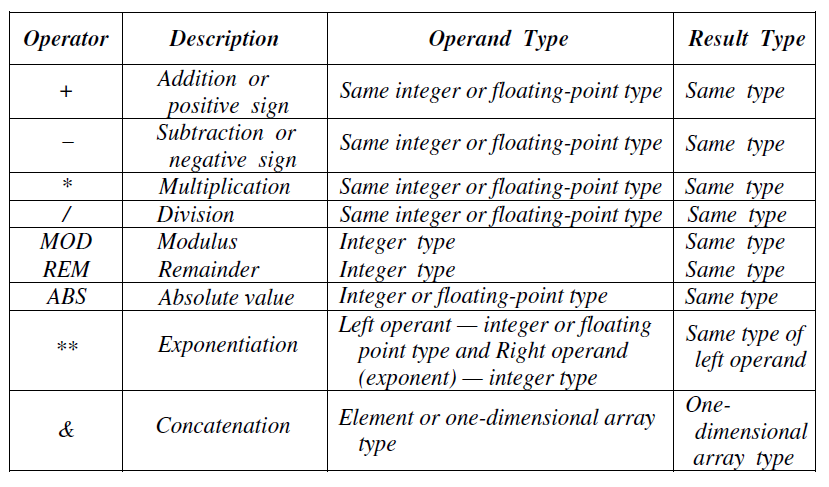


Figure 4.19: VHDL Arithmetic Operators

The exponential operator accepts both integer and floating-point numbers with exceptions. The data type of the left operand of the exponential operator defines the data type of the result, and can be integer or floating point. The right operand (exponent) must be of integer type.

The operator that has the highest precedence includes the exponential, the absolute value (ABS) and the NOT operators. The operator class that has the lowest precedence includes the logical operators like AND, OR, NAND, NOR, XOR and XNOR.

**4.7 VHDL Signal and Generate Statements**

It includes signal statements and generate statements.

**4.7.1 Signal Statement**

A signal is a VHDL keyword. It declares a signal of specified data type. A signal declaration is used to represent internal signals within the architecture declaration. Unlike entity ports, internal signals do not have a direction. Signal assignment statements execute only the associated signals (appearing on the right-hand side of the assignment statement) changes values. In VHDL, code does not affect the order in which the statements are executed. Signal assignment are concurrent and could be executed in parallel fashion. Considering Figure 4.20,

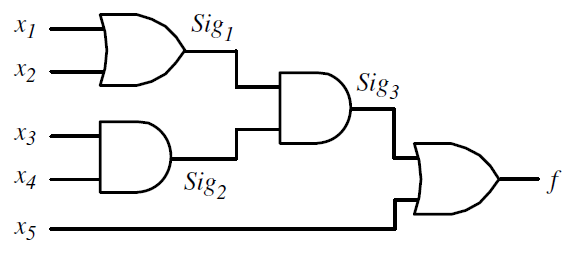


Figure 4.20: Logic Circuit with Internal Signals

Where the internal signals are identified, note that from the point of view of an entity declaration, the signals , and  are internal signals. They are neither input ports nor output ports, and therefore do not have a direction.

Also, the VHDL code is given in Figure 4.21. The architecture declaration has been modified to include the internal signals. The logic function of the circuit is described in an indirect way using the internal signals.

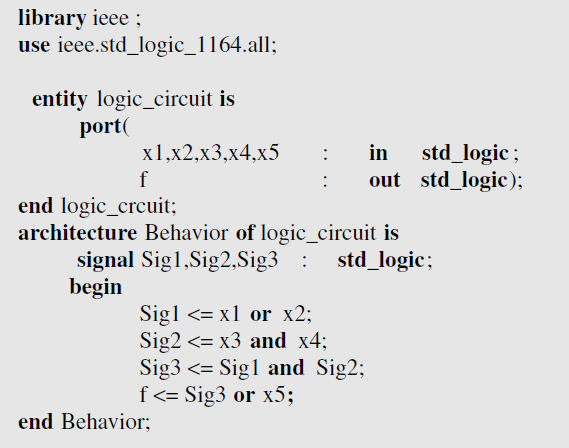


Figure 4.21: VHDL Code for the Logic Circuit in Figure 4.20.

**4.7.2 Generate Statement**

This is a VHDL keyword that is used to replicate a set of concurrent statements or selectively execute a set of concurrent statements if a specified condition is met. The generate statement provides a method of repeating a logic function or a component instantiation without normally writing the logic function or the component instantiation. There are two types of generate statements: **for generate**, which is an iterative generate statement, and the **if generate,** which is a conditional generate statement. Examples of these two is given in Figure 4.22.

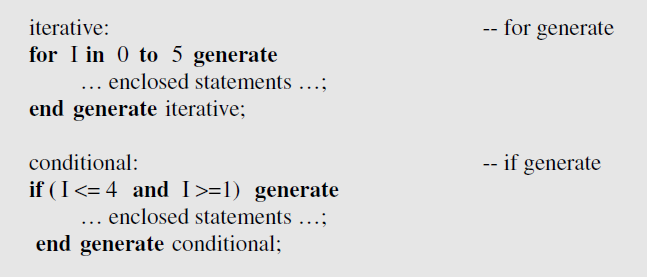


Figure 4.22: Generate Statement

Note that a generate statement begins with a unique label that identifies it. The label also appears at the close of the statement. The generate statement repeats the enclosed statements at each iteration or conditional pass. The index I is declared implicitly inside the generate statement and cannot be changed by the program.

**4.8 Sequential Statements**

Sequential statements allow the designer to describe the operation, or behaviour, of a circuit as a sequence of related events. Sequential statements are found within process, functions and procedures. They differ from concurrent statements, in that they have order dependency, which may or may not imply a sequential circuit (one involving memory elements). VHDL’s process statement is the primary way to enter a sequential statement. A process statement, including all declarations and sequential statements within it, is actually a single concurrent statement within a VHDL architecture. This means that the designer can write as many processes and other concurrent statements as are necessary to describe a design, without worrying about the order in which the simulator will process each statement. Thus, the process statement constitutes the behavioural statement in VHDL. Figure 4.23 shows a pictorial representation of the general structure of the process statement.

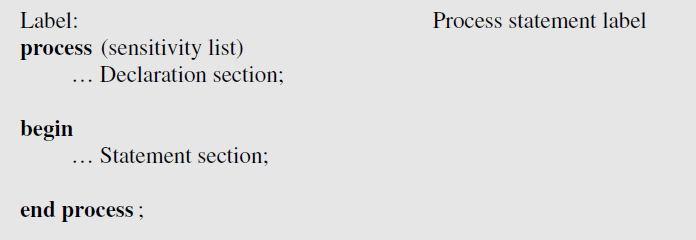


Figure 4.23: Sequential Declaration

A process statement is listed in the architecture declaration. The process statement begins with the reserved keyword process, following the begin statement of the architecture. The process statement includes a begin statement inside it as well. The process statement concludes with an end process statement. The process may have a label, which also appears at the close of the process statement. The process statement structure is composed of a declaration section and a statement section. The declaration section declares the objects that will be used in the statement section of the process statement. The statement section includes sequential statements, which describe the sequential behaviour of the logic circuit. The process statement may include a sensitivity list, which defines the activation and suspension of the process statement based on the changes in signals included in the sensitivity list. Because the process statement represents sequential behaviour, it must include an explicit wait statement to control the activation and suspension of the process statement. When a sensitivity list is included in the process statement, the wait statement is not necessary. The sensitivity list provides an implicit wait, which describes the events of the signals listed.

**4.9 Loops and Decision-Making Statements**

There are three primary types of loops in VHDL, these include; for loops, while loops and infinite loops. VHDL also provides if-then-else and case statements to implement control structures.

**4.9.1 For Loop**

A **for loop** is a sequential statement that allows a designer to specify a fixed number of iterations in a behavioural design description. It is important to note that in VHDL, unlike other software programs, each iteration occurs concurrently, which means that the loop is “unrolled.” A for loop can be used only inside a sequential statement, such as a process statement, a function, or a procedure. Figure 4.24 shows a **for statement.**

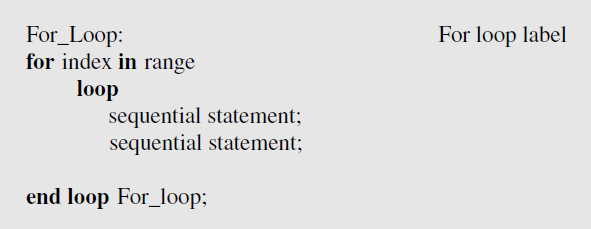


Figure 4.24: **For Loop**

A for loop executes the sequential statement within its body each time the index of the loop changes its value within the range of theloop. The label of the **for loop** is optional; the user may choose not to include it.

**4.9.2 While Loop**

A while loop is another form of sequential loop statement that specifies the conditions under which the loop should continue rather than specifying a discrete number of iterations. The condition must be of type **Boolean**. A while loop executes the sequential statements in its body each time the condition is checked and evaluated to be true. Otherwise, the while loop terminates. Similar to a **for loop**, the label of a **while loop** is optional and can be omitted. Figure 4.25 shows the general form of the **while loop** in the code sample format.

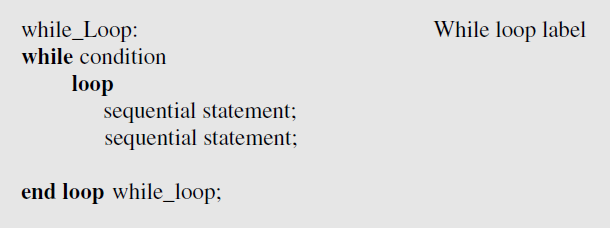


Figure 4.25: While Loop

**4.9.3 If-Then-Else Statement**

The if–then–else statement is the most commonly used form of control statement in VHDL. Figure 4.26 shows the general form of the code sample format.

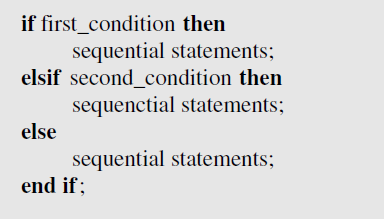


Figure 4.26: If-Then-Else Statement

Note that the condition statements must be expressions of type **Boolean.**

**4.9.4 Case Statement**

A case statement can be used as an alternative to an **if–then–else** control structure. However, it is important to understand the differences in the circuits generated by CAD tools for the **if** and **case** statements, and to use the appropriate control structure for a design. A case statement is generally used when making decisions among options that have equal priority. The making-decision control expression must have a finite set of values. The when statement switches between the possible values. The when others statement is the final statement with a case statement. A when others statement is used to represent the remaining possible values of the control expression, not listed in the previous when statements. Figure 4.27 shows the general form of a **case** statement in the code sample format.

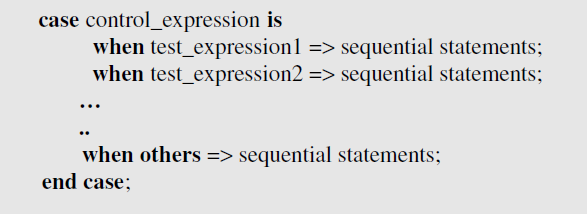


Figure 4.27: Case Statement

Note that the **when others** condition makes sure that all cased (that are not listed within the case body) are covered.