IEEE Standard VHDL Packages

Sponsor

Design Automation Standards Committee

of the
IEEE Computer Society

Approved 20 March 1997

IEEE Standards Board

IEEE Std 1076.3-1997

Synthesis

Abstract: The current interpretation of common logic values and the association of numeric values to specific VHDL array types is described. This standard provides semantic for the VHDL synthesis domain, and enables formal verification and simulation acceleration in the VHDL based design. The standard interpretations are provided for values of standard logic types defined by IEEE Std 1164-1993, and of the BIT and BOOLEAN types defined in IEEE Std 1076-1993. The numeric types SIGNED and UNSIGNED and their associated operators define integer and natural number arithmetic for arrays of common logic values. Two's complement and binary encoding techniques are used. The numeric semantic is conveyed by two VHDL packages. This standard also contains any allowable modifications.

Keywords: interpretations, metalogical values, numeric VHDL vector types, signed, synthesis, unsigned

The Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street, New York, NY 10017-2394, USA

ISBN 1-55937-923-5

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE that have expressed an interest in participating in the development of the standard.

Use of an IEEE Standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more

than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA

Note: Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Introduction

(This introduction is not a part of IEEE Std 1076.3-1997, IEEE Standard VHDL Synthesis Packages.)

This standard, IEEE Std 1076.3-1997, supports the synthesis and verification of hardware designs, by defining vector types for representing signed or unsigned integer values and providing standard interpretations of widely used scalar VHDL values.

The standardization activity started during the development of IEEE Std 1076-1993, IEEE Standard VHDL Language Reference Manual, to address a number of issues in the synthesis area that could not be adequately addressed within the scope of the main 1076 project.

The initial Synthesis Special Interest Group (SSIG) analyzed a wide range of requirements and grouped them in four categories:

a) Standard Interpretations of IEEE Std 1164-1993 values for synthesis
b) Numeric types for synthesis
c) Special attribute semantics
d) Constraint specification

Consensus was reached only on solutions presented for categories a) and b). The large working group then commissioned a Pilot Team to drive the standardization effort. The three standardization chapters (North America, Europe, and Asia-Pacific) were all represented in the Pilot Team. The active members of the Pilot Team were the following:

Alex N. Zamfirescu, *Chair*

Wolfgang Ecker	European Chapter Representative
Kazuhiro Yoshinaga	Asia-Pacific Chapter Chair
Rob Anderson	Library Design Chair
J. Bhasker	Ballot Comment Resolution Chair
David Bishop	Repository and WWW Account Administrator
Dominique Borrione	Leader, Formal Verification Effort
James H. Vellenga	Documentation and Pilot Team Co-Chair
Rob Dekker	
Bob Flatt	
Chris Kingsley	

The hard work and professionalism of the Pilot Team members contributed significantly to the final result. Although the Working Group met regularly and voted on all major issues, the Pilot Team also extensively used electronic mail, a common repository, and several World Wide Web pages to accelerate the completely voluntary standardization process of the IEEE. A simple VHDL test suite (not part of the standard) that exercises and verifies the packages was also produced during standardization. A set of axioms and formal properties involving standard operators has been formally proven.

Individuals from many organizations participated in the development of IEEE Std 1076.3-1997. In addition to members of the Pilot Team, the following individuals attended meetings of the Synthesis Working Group:

Dave Ackley
Mart Altmäe
Jean-Michel Bergé
Glenn Boysko
Joanne DeGroat
Allen Dewey
Iain Finlay
Björn Fjellborg
Chris Flynn
Brian Griffin
Bradley Grove
James P. Hanna
Masamichi Kawarabayashi
Chi Lai Huang
Naotaka Maeda
Sabine Maertz
Kiyoshi Makino
Yasunori Mako
Erich Marschner
Victor M. Martin
Francoise Martinolle
Michael McKinney
Adam Morawiec
Yutaka Murase

John Hillawi
Robert Hillman
Masaharu Imai
Zainalabedin Navabi
Kevin O'Brien

The following persons were on the balloting committee:

Mostapha Aboulhamid
John Ainscough
Robert E. Anderson
LaNae Avra
Pete Bakowski
Daniel S. Barclay
David L. Barton
Mike Beaver
Jean-Michel Bergé
Victor Berman
J. Bhasker
William D. Billowitch
Dominique Borrione
Dennis B. Brophy
Yu-I Hsieh
Christophe Hui Bon Hoa
Sylvie Hurat
Masaharu Imai
Mitsuaki Ishikawa
Stephen Ives
David Jakopac
Takashi Kambe
Masamichi Kawarabayashi
Choon B. Kim
Chris Kingsley
Stanley J. Krolikoski
Charles R. Lang
Marc Laurent

Walter H. Burkhardt
Raul Camposano
Todd P. Carpenter
Moon Jung Chung
David Coelho
Edmond S. Cooley
Alan Coppola
Robert A. Cottrell
Timothy R. Davis
Allen Dewey
Michael A. Dukes
Douglas D. Dunlop
William Fazakerly
Robert A. Flatt
Walter Geisselhardt
Brian Griffin
Richard Grisel
Steve Grout
Andrew Guyler
Jean Lebrun
Steven Levitan
Bob Lisanke
Alfred Lowenstein
Rajeev Madhavan
Naotaka Maeda
Serge Maginot
Maqsoodul Mannan
F. Erich Marschner
Victor M. Martin
Peter Marwedel
Paul J. Menchini
Jean Mermet
Gerald T. Michael
Israel Michel
Toshio Misawa
John T. Montague
Larry Moore
Gabe Moretti

James P. Hanna
William A. Hanna
Randolph E. Harr
Frederick Hill
Robert G. Hillman
Kazuyuki Hirakawa
Paul W. Horstmann
Yee-Wing Hsieh
Vijay Nagasamy
Zainalabedin Navabi
Wolfgang W. Nebel
Kevin O'Brien
Eamonn O'Brien-Strain
Yoichi Onishi
Mauro Pipponzi

Venu Pemmaraju

C. R. Ramesh

Ray Ryan

Larry F. Saunders

Jay Schleicher

Quentin Schmierer

Kenneth E. Scott

Manfred Selz

Hirotake Shinde

Dennis Soderberg

Yuri Tatarnikov

Victor Toporkov

Tatiana Trondora

Kerry Veenstra
Eugenio Villar

Gary S. Porter
Adam Postula

Jean Pouilly

Shiv Prakash

Paolo Prinetto

Jan Pukite

Hemant G. Rotithor

Jacques Rouillard

Ray Ryan

Johan Sandstrom

Larry F. Saunders

Quentin Schmierer

Kenneth E. Scott

Francesco Sforza

Moe Shahdad

Ravi Shankar

Balmukund Sharma

Charles Shelor

Raj Singh

Supreet Singh

David W. Smith

William Bong H. Soon

Alec G. Stanculescu

Balsha R. Stanisic

Michael F. Sullivan

Charles Swart

Peter Trajmar
Fatehy El-Turky
Cary Ussery
James H. Vellenga
Ranganadha R. Vemuri
Venkat V. Venkataraman

Eugenio Villar

Martin J. Walter

Greg Ward

Ronald Waxman

Alan Whittaker

John C. Willis

Alex N. Zamfirescu

Reinhard Zippelius

Mark Zwolinski

When the IEEE Standards Board approved this standard on 20 March 1997, it had the following membership:

Donald C. Loughry, *Chair*

Clyde R. Camp
Stephen L. Diamond
Harold E. Epstein
Donald C. Fleckenstein
Jay Forster*
Thomas F. Garrity
Donald N. Heirman
Jim Isaak
Ben C. Johnson

*Member Emeritus

Richard J. Holleman, *Vice Chair*

Andrew G. Salem, *Secretary*

Lowell Johnson
Robert Kennelly
E. G. "Al" Kiener
Joseph L. Koepfinger*
Stephen R. Lambert
Lawrence V. McCall
L. Bruce McClung

Louis-François Pau
Gerald H. Peterson
John W. Pope
Jose R. Ramos
Ronald H. Reimer
Ingo Rüsch
John S. Ryan

Marco W. Migliaro

Chee Kiow Tan
Howard L. Wolfman

Also included are the following nonvoting IEEE Standards Board liaisons:

Satish K. Aggarwal
Alan H. Cookson

Kim Breitfelder
IEEE Standards Project Editor

Λ

Contents

IEEE Standard VHDL Synthesis Packages

1. Overview

1.1 Scope

This standard defines standard practices for synthesizing binary digital electronic circuits from VHDL source code. It includes the following:

a) The hardware interpretation of values belonging to the BIT and BOOLEAN types defined by IEEE Std 1076-1993[1] and to the STD_ULOGIC type defined by IEEE Std 1164-1993.

b) A function (STD_MATCH) that provides "don't care" or "wild card" testing of values based on the STD_ULOGIC type.

c) Standard functions for representing sensitivity to the edge of a signal.

d) Two packages that define vector types for representing signed and unsigned arithmetic values, and

that define arithmetic, shift, and type conversion operations on those types.

This standard is designed for use with IEEE Std 1076-1993. Modifications that may be made to the packages for use with the previous edition, IEEE Std 1076-1987, are described in 7.2.

1.2 Terminology

The word *shall* indicates mandatory requirements strictly to be followed in order to conform to the standard and from which no deviation is permitted (*shall* equals *is required to*). The word *should* is used to indicate that a certain course of action is preferred but not necessarily required; or that (in the negative form) a certain course of action is deprecated but not prohibited (*should* equals *is recommended that*). The word *may* indicates a course of action permissible within the limits of the standard (*may* equals *is permitted*).

A synthesis tool is said to *accept* a VHDL construct if it allows that construct to be legal input; it is said to *interpret* the construct (or to provide an *interpretation* of the construct) by producing something that represents the construct. A synthesis tool is not required to provide an interpretation for every construct that it accepts, but only for those for which an interpretation is specified by this standard.

---

[1]Information on references can be found in Clause 2.

1.3 Conventions

This standard uses the following conventions:

a) The body of the text of this standard uses boldface to denote VHDL reserved words (such as **downto**) and upper case to denote all other VHDL identifiers (such as REVERSE_RANGE or FOO).

b) The text of the VHDL packages defined by this standard, as well as the text of VHDL examples and code fragments, is represented in a fixed-width font. All such text represents VHDL reserved words as lower case text and all other VHDL identifiers as upper case text.

c) In the body of the text, italics denote words or phrases that are being defined by the paragraph in which they occur.

d) VHDL code fragments not supported by this standard are denoted by an italic fixed-width font.

2. References

This standard shall be used in conjunction with the following publications. When the following standards are superseded by an approved revision, the revision shall apply.

IEEE Std 1076-1993, IEEE Standard VHDL Language Reference Manual (ANSI).[2]

IEEE Std 1164-1993, IEEE Standard Multivalue Logic System for VHDL Model Interoperability (Std_logic_1164) (ANSI).

3. Definitions

Terms used in this standard, but not defined in this clause, are assumed to be from IEEE Std 1076-1993 and IEEE Std 1164-1993.

3.1 argument: An expression occurring as the actual value in a function call or procedure call.

3.2 arithmetic operation: An operation for which the VHDL operator is +, -, *, /, **mod**, **rem**, **abs**, or **.

3.3 assignment reference: The occurrence of a literal or other expression as the waveform element of a signal assignment statement or as the right-hand side expression of a variable assignment statement.

3.4 don't care value: The enumeration literal '-' of the type STD_ULOGIC defined by IEEE Std 1164-1993.

3.5 equality relation: A VHDL relational expression in which the relational operator is =.

3.6 high-impedance value: The enumeration literal 'Z' of the type STD_ULOGIC defined by IEEE Std 1164-1993.

3.7 inequality relation: A VHDL relational expression in which the relational operator is /=.

3.8 logical operation: An operation for which the VHDL operator is **and**, **or**, **nand**, **nor**, **xor**, **xnor**, or **not**.

3.9 metalogical value: One of the enumeration literals 'U', 'X', 'W', or '-' of the type STD_ULOGIC

defined by IEEE Std 1164-1993.

---

[2]IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.

3.10 ordering relation: A VHDL relational expression in which the relational operator is <, <=, >, or >=.

3.11 shift operation: An operation for which the VHDL operator is **sll**, **srl**, **sla**, **sra**, **rol**, or **ror**.

3.12 standard logic type: The type STD_ULOGIC defined by IEEE Std 1164-1993, or any type derived from it, including, in particular, one-dimensional arrays of STD_ULOGIC or of one of its subtypes.

3.13 synthesis tool: Any system, process, or tool that interprets VHDL source code as a description of an electronic circuit in accordance with the terms of this standard and derives an alternate description of that circuit.

3.14 user: A person, system, process, or tool that generates the VHDL source code that a synthesis tool processes.

3.15 vector: A one-dimensional array.

3.16 well-defined: Containing no metalogical or high-impedance element values.

4. Interpretation of the standard logic types

This clause defines how a synthesis tool shall interpret values of the standard logic types defined by IEEE Std 1164-1993 and of the BIT and BOOLEAN types defined by IEEE Std 1076-1993. Simulation tools,

however, shall continue to interpret these values according to the standards in which the values are defined.

4.1 The STD_LOGIC_1164 values

IEEE Std 1164-1993 defines the standard logic type:

```
type STD_ULOGIC is ( 'U', -- Uninitialized
                     'X', -- Forcing Unknown
                     '0', -- Forcing 0
                     '1', -- Forcing 1
                     'Z', -- High Impedance
                     'W', -- Weak Unknown
                     'L', -- Weak 0
                     'H', -- Weak 1
                     '-'  -- Don't care
                   );
```

The *logical values* '1', 'H', '0', and 'L' are interpreted as representing one of two logic levels, where each logic level represents one of two distinct voltage ranges in the circuit to be synthesized.

IEEE Std 1164-1993 also defines a resolution function named RESOLVED and a subtype STD_LOGIC that is derived from STD_ULOGIC by using RESOLVED. The resolution function RESOLVED treats the values '0' and '1' as *forcing values* that override the *weak values* 'L' and 'H' when multiple sources drive the same signal.

The values 'U', 'X', 'W', and '-' are *metalogical values*; they define the behavior of the model itself rather than the behavior of the hardware being synthesized. The value 'U' represents the value of an object before it is explicitly assigned a value during simulation; the values 'X' and 'W' represent forcing and weak values,

respectively, for which the model is not able to distinguish between logic levels.

The value '-' is also called the *don't care value*. This standard treats it in the same way as the other metalogical values except when it is furnished as an argument to the STD_MATCH functions in the

IEEE.NUMERIC_STD package. The STD_MATCH functions use '-' to implement a "match all" or "wild card" matching.

The value 'Z' is called the *high-impedance value*, and represents the condition of a signal source when that source makes no effective contribution to the resolved value of the signal.

4.2 Static constant values

Wherever a synthesis tool accepts a reference to a locally static or globally static named constant, it shall treat that constant as the equivalent of the associated static expression.

4.3 Interpretation of logic values

This subclause describes the interpretations of logic values occurring as literals (or in literals) after a synthesis tool has replaced named constants by their corresponding values.

4.3.1 Interpretation of the forcing and weak values ('0', '1', 'L', 'H', FALSE, TRUE)

A synthesis tool shall interpret the following values as representing a logic value 0:

- The BIT value '0'.
- The BOOLEAN value FALSE.
- The STD_ULOGIC values '0' and 'L'.

It shall interpret the following values as representing a logic value 1:

- The BIT value '1'.
- The BOOLEAN value TRUE.
- The STD_ULOGIC value '1' and 'H'.

This standard makes no restriction as to the interpretation of the relative strength of values.

4.3.2 Interpretation of the metalogical values ('U', 'W', 'X', '-')

4.3.2.1 Metalogical values in relational expressions

If the VHDL source code includes an equality relation (=) for which one operand is a static metalogical value and for which the other operand is not a static value, a synthesis tool shall interpret the equality relation as equivalent to the BOOLEAN value FALSE. If one operand of an equality relation is a vector, and one element of that vector is a static metalogical value, a synthesis tool shall interpret the entire equality relation as equivalent to the BOOLEAN value FALSE.

A synthesis tool shall interpret an inequality relation (/=) for which one operand is or contains a static metalogical value, and for which the other operand is not a static value, as equivalent to the BOOLEAN value TRUE.

A synthesis tool shall treat an ordering relation for which at least one operand is or contains a static metalogical value as an error.

4.3.2.2 Metalogical values as a choice in a case statement

If a metalogical value occurs as a choice, or as an element of a choice, in a case statement that is interpreted by a synthesis tool, the synthesis tool shall interpret the choice as one that can never occur. That is, the inter-

pretation that is generated is not required to contain any constructs corresponding to the presence or absence of the sequence of statements associated with the choice.

Whenever a synthesis tool interprets a case statement alternative that associates multiple choices with a single sequence of statements, it shall produce an interpretation consistent with associating the sequence of statements with each choice individually.

Whenever a synthesis tool interprets a selected signal assignment statement, it shall interpret the selected signal assignment statement as if it were the case statement in the equivalent process as defined by IEEE Std 1076-1993.

4.3.2.3 Metalogical values in logical, arithmetic, and shift operations

When a static metalogical value occurs as all of, or one element of, an operand to a logical, arithmetic, or shift operation, and when the other operand to the operation is not a static value, a synthesis tool shall treat the operation as an error.

4.3.2.4 Metalogical values in concatenate operations

If a static metalogical value occurs as all of, or as one element of, an operand to the concatenate (&) operator, a synthesis tool shall treat it as if it had occurred as the corresponding element of the expression formed by the concatenate operation.

4.3.2.5 Metalogical values in type conversion and sign-extension functions

If a static metalogical value occurs as all of, or as one element of, the value argument to a type conversion or sign-extension function, a synthesis tool shall treat it as if it had occurred as the corresponding element of the expression formed by the function call.

4.3.2.6 Metalogical values used in assignment references

A synthesis tool shall accept a static metalogical value used as all of, or as one element of, an assignment reference, but is not required to provide any particular interpretation of that metalogical value.

4.3.3 Interpretation of the high-impedance value ('Z')

If the static value 'Z' occurs as an assignment reference in a signal assignment statement, a synthesis tool shall interpret the assignment as implying the equivalent of a three-state buffer that is disabled when the conditions under which the assignment occurs is true. The output of the three-state buffer is the target of the assignment. The input of the three-state buffer is the logic network that represents the value of the target apart from any assignments to 'Z'.

If the 'Z' occurs as one or more elements of an assignment reference in a signal assignment statement, a synthesis tool shall interpret each such occurrence as implying the equivalent of a three-state buffer in the man-

ner defined by the preceding paragraph.

This standard does not specify an interpretation when a static value 'Z' occurs as all of, or one bit of, an assignment reference in a variable assignment statement.

Whenever a static high-impedance value occurs in any context other than an assignment reference, a synthesis tool shall treat it as equivalent to a static metalogical value.

NOTE—A signal assignment statement that assigns one or more bits of a signal to 'Z' unconditionally implies the equivalent of a three-state buffer that is always disabled. A synthesis tool may choose to ignore such assignments.

5. The STD_MATCH function

The NUMERIC_STD package defined by this standard defines functions named STD_MATCH to provide wild card matching for the don't care value. Whenever the STD_MATCH function compares two arguments which are STD_ULOGIC values, it returns TRUE if and only if:

- Both values are well-defined and the values are the same, or
- One value is '0' and the other is 'L', or
- One value is '1' and the other is 'H', or
- At least one of the values is the don't care value ('-').

Whenever the STD_MATCH function compares two arguments which are vectors whose elements belong to the STD_ULOGIC type or to one of its subtypes, it returns TRUE if and only if:

a) The operands have the same length, and
b) STD_MATCH applied to each pair of matching elements returns TRUE.

When one of the arguments to the STD_MATCH function is a static value and the other is not, a synthesis tool shall interpret the call to the STD_MATCH function as equivalent to an equality test on matching elements of the arguments, excepting those elements of the static value which are equal to '-'.

NOTE—If any argument value passed to STD_MATCH is or contains a metalogical or high-impedance value other than '-', the function returns FALSE.

6. Signal edge detection

Wherever a synthesis tool interprets a particular expression as the edge of a signal, it shall also interpret the function RISING_EDGE as representing a rising edge and the function FALLING_EDGE as representing a falling edge, where RISING_EDGE and FALLING_EDGE are the functions declared either by the package STD_LOGIC_1164 of IEEE Std 1164-1993 or by the NUMERIC_BIT package of this standard.

7. Standard arithmetic packages

Two VHDL packages are defined by this standard. The NUMERIC_BIT package is based on the VHDL type BIT, while the second package, NUMERIC_STD, is based on the subtype STD_LOGIC of the type STD_ULOGIC. Simulations based on the subprograms of the NUMERIC_BIT package ordinarily require less execution time, because the subprograms do not have to deal with operands containing metalogical or high-impedance values. Use of the subprograms of the NUMERIC_STD package allow simulation to detect the propagation or generation of metalogical values.

Each package defines a vector type named SIGNED and a vector type named UNSIGNED. The type UNSIGNED represents an unsigned binary integer with the most significant bit on the left, while the type SIGNED represents a two's-complement binary integer with the most significant bit on the left. In particular, a one-element SIGNED vector represents the integer values –1 and 0.

The two packages are mutually incompatible, and only one shall be used in any given design unit. To facilitate changing from one package to the other, most of the subprograms declared in one package are also declared for corresponding arguments in the other. Exceptions are when:

a) The NUMERIC_BIT package declares the functions RISING_EDGE and FALLING_EDGE; the corresponding functions for STD_ULOGIC are declared by the STD_LOGIC_1164 package.

b) The NUMERIC_STD package declares the STD_MATCH functions, which give special treatment to the don't care value, whereas the BIT-based types of the NUMERIC_BIT package have no don't care values.

c) The NUMERIC_STD package declares the TO_01 functions, which may be applied to SIGNED and UNSIGNED vector values, and which map the element values of the vectors to the STD_ULOGIC values '0' and '1' and to a third value representing metalogical or high-impedance values.

Table 1 shows the order of the function declarations within the package declarations.

Table 1—Order of functions within packages

Function Id(s)	NUMERIC_BIT	NUMERIC_STD
A.1 A.2	**abs** unary –	**abs** unary –
A.3–A.8 A.9–A.14 A.15–A.20 A.21–A.26 A.27–A.32 A.33–A.38	binary + binary – * / **rem** **mod**	binary + binary – * / **rem** **mod**
C.1–C.6 C.7–C.12 C.13–C.18 C.19–C.24 C.25–C.30 C.31–C.36	>> < <= >= = /=	>> < <= >= = /=
S.1, S.3 S.2, S.4 S.5, S.7 S.6, S.8	SHIFT_LEFT SHIFT_RIGHT ROTATE_LEFT ROTATE_RIGHT	SHIFT_LEFT SHIFT_RIGHT ROTATE_LEFT ROTATE_RIGHT
S.9, S.10 S.11, S.12 S.13, S.14 S.15, S.16	(predefined in VHDL)	**sll** **srl** **rol** **ror**
R.1–R.2	RESIZE	RESIZE
D.1-2 D.3 D.4	TO_INTEGER TO_UNSIGNED TO_SIGNED	TO_INTEGER TO_UNSIGNED TO_SIGNED

D.4	TO_SIGNED	TO_SIGNED
E.1 E.2	RISING_EDGE FALLING_EDGE	(defined by the STD_LOGIC_1164 package)
L.1, L.8 L.2, L.9 L.3, L.10 L.4, L.11 L.5, L.12 L.6, L.13 L.7, L.14	**not** **and** **or** **nand** **nor** **xor** **xnor**	**not** **and** **or** **nand** **nor** **xor** **xnor**
M.1–M.5		STD_MATCH
T.1–T.2		TO_01

If a null array is furnished as an input argument to any subprogram declared by NUMERIC_BIT or NUMERIC_STD, a synthesis tool shall treat it as an error.

All vector return values that are not null array values are normalized so that the direction of the index range is **downto** and the right bound is 0. A vector return value that is a null array has the index range "0 **downto** 1".

The package declarations use the following format to declare each function:

```
    -- Id: <id_nr>
function <designator> (<formal_parameter_list>) return <type_mark>;
  -- Result Subtype: <subtype_indication>
  -- Result: <description of function>
```

The elements of this format have the following meanings:

<id_nr>
A unique identifier of the form *letter.number*. A corresponding identifier appears at the beginning of the corresponding function body in the package body for the same package.

<designator>
The function designator as defined by IEEE Std 1076-1993.

<formal_parameter_list>
The formal parameter list for the function as defined by IEEE Std 1076-1993.

<type_mark>
A type mark denoting the result subtype of the function as defined by IEEE Std 1076-1993.

<subtype_indication>
The subtype of the value returned by the function. If the result subtype of the function denotes an unconstrained vector subtype, <subtype_indication> also includes an index constraint defining the index range of the returned value in terms of sizes and values of the input parameters. <subtype_indication> is syntactically a subtype indication as defined by IEEE Std 1076-1993.

<description of function>
An English language description of the operation performed by the function.

Both packages shall be analyzed into the library symbolically named IEEE.

7.1 Allowable modifications

Vendors of tools conforming to this standard shall not modify the package declarations for NUMERIC_BIT or NUMERIC_STD. However, a vendor may provide package bodies for either or both packages in which subprograms are rewritten for more efficient simulation or synthesis, provided that the behavior of the rewritten subprograms remains the same under simulation. The behavior of the original and rewritten subprograms are the same if, for any combination of input values, they return the same return values. The text of messages associated with assertions may differ in the rewritten subprogram.

The package bodies for both packages declare a constant named NO_WARNING that has the value FALSE. A user may set NO_WARNING to TRUE and reanalyze the package body to suppress warning messages generated by calls to the functions in these packages. For this reason:

— A tool vendor who rewrites the package body shall preserve the declaration of the NO_WARNING constant to allow a user to suppress warnings by editing and reanalyzing the package body.

— A simulation tool vendor who provides a preanalyzed version of the package body should also provide a mechanism for suppressing warning messages generated by the package functions.

7.2 Compatibility with IEEE Std 1076-1987

The following functions from the NUMERIC_STD package are compatible with IEEE Std 1076-1993 but not with the previous edition, IEEE Std 1076-1987:

a) "xnor"
b) "sll"
c) "srl"
d) "rol"
e) "ror"

To use these functions with a VHDL-based system that has not yet been upgraded to be compatible with IEEE Std 1076-1993, a user or vendor may comment out the subprogram declarations and subprogram bodies.

In addition, IEEE Std 1076-1993 supports a character set that includes the copyright symbol (©). However, IEEE Std 1076-1987 does not support this same character set. Therefore, in order to use the NUMERIC_BIT and NUMERIC_STD packages with a system that has not yet been upgraded to be compatible with IEEE Std 1076-1993, a user or vendor may replace the copyright symbol within the sources of those packages by a left parenthesis, a lowercase “c,” and a right parenthesis.

7.3 The package texts

The texts of the NUMERIC_BIT and NUMERIC_STD packages (both package declarations and package

bodies) are on the diskette that is included with this standard. Those texts are an official part of this standard. For the convenience of users, the package declarations are also included in the printed form of the standard. Please consult the diskette for the contents of the package bodies.

7.3.1 Package declaration for NUMERIC_BIT

```
-- ------------------------------------------------------------------------
--
-- Copyright © 1997 by IEEE. All rights reserved.
--
-- This source file is an essential part of IEEE Std 1076.3-1997,
-- IEEE Standard VHDL Synthesis Packages. This source file may not be
-- copied, sold, or included with software that is sold without written
```

```
-- permission from the IEEE Standards Department. This source file may
-- be used to implement this standard and may be distributed in compiled
-- form in any manner so long as the compiled form does not allow direct
-- decompilation of the original source file. This source file may be
-- copied for individual use between licensed users. This source file is
-- provided on an AS IS basis. The IEEE disclaims ANY WARRANTY EXPRESS OR
-- IMPLIED INCLUDING ANY WARRANTY OF MERCHANTABILITY AND FITNESS FOR USE
-- FOR A PARTICULAR PURPOSE. The user of the source file shall indemnify
-- and hold IEEE harmless from any damages or liability arising out of the
-- use thereof.
--
-- This package may be modified to include additional data required by tools,
-- but it must in no way change the external interfaces or simulation behavior
-- of the description. It is permissible to add comments and/or attributes to
-- the package declarations, but not to change or delete any original lines of
-- the package declaration. The package body may be changed only in accordance
-- with the terms of 7.1 and 7.2 of this standard.
--
-- Title      : Standard VHDL Synthesis Packages (IEEE Std 1076.3-1997, NUMERIC_BIT)
--
-- Library    : This package shall be compiled into a library symbolically
--            : named IEEE.
--
-- Developers : IEEE DASC Synthesis Working Group.
--
-- Purpose    : This package defines numeric types and arithmetic functions
--            : for use with synthesis tools. Two numeric types are defined:
--            : -- > UNSIGNED: represents an UNSIGNED number in vector form
--            : -- > SIGNED: represents a SIGNED number in vector form
--            : The base element type is type BIT.
```

```
--              : The leftmost bit is treated as the most significant bit.
--              : Signed vectors are represented in two's complement form.
--              : This package contains overloaded arithmetic operators on
--              : the SIGNED and UNSIGNED types. The package also contains
--              : useful type conversions functions, clock detection
--              : functions, and other utility functions.
--              :
--              : If any argument to a function is a null array, a null array is
--              : returned (exceptions, if any, are noted individually).
--
-- Note         : No declarations or definitions shall be included in, or
--              : excluded from, this package. The "package declaration" defines
--              : the types, subtypes, and declarations of NUMERIC_BIT. The
--              : NUMERIC_BIT package body shall be considered the formal
--              : definition of the semantics of this package. Tool developers
--              : may choose to implement the package body in the most efficient
--              : manner available to them.
--              :
-- ---------------------------------------------------------------------------
-- Version      : 2.4
-- Date         : 12 April 1995
-- ---------------------------------------------------------------------------
```

```
package NUMERIC_BIT is
  constant CopyRightNotice: STRING
      := "Copyright © 1997 IEEE. All rights reserved.";

  --============================================================================
  -- Numeric Array Type Definitions
  --============================================================================

  type UNSIGNED is array (NATURAL range <> ) of BIT;
  type SIGNED is array (NATURAL range <> ) of BIT;

  --============================================================================
  -- Arithmetic Operators:
  --============================================================================

  -- Id: A.1
  function "abs" (ARG: SIGNED) return SIGNED;
  -- Result subtype: SIGNED(ARG'LENGTH-1 downto 0)
  -- Result: Returns the absolute value of a SIGNED vector ARG.

  -- Id: A.2
  function "-" (ARG: SIGNED) return SIGNED;
  -- Result subtype: SIGNED(ARG'LENGTH-1 downto 0)
  -- Result: Returns the value of the unary minus operation on a
  --         SIGNED vector ARG.

  --============================================================================

  -- Id: A.3
```

```
function "+" (L, R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(MAX(L'LENGTH, R'LENGTH)-1 downto 0)
-- Result: Adds two UNSIGNED vectors that may be of different lengths.

-- Id: A.4
function "+" (L, R: SIGNED) return SIGNED;
-- Result subtype: SIGNED(MAX(L'LENGTH, R'LENGTH)-1 downto 0)
-- Result: Adds two SIGNED vectors that may be of different lengths.

-- Id: A.5
function "+" (L: UNSIGNED; R: NATURAL) return UNSIGNED;
-- Result subtype: UNSIGNED(L'LENGTH-1 downto 0)
-- Result: Adds an UNSIGNED vector, L, with a nonnegative INTEGER, R.

-- Id: A.6
function "+" (L: NATURAL; R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(R'LENGTH-1 downto 0)
-- Result: Adds a nonnegative INTEGER, L, with an UNSIGNED vector, R.

-- Id: A.7
function "+" (L: INTEGER; R: SIGNED) return SIGNED;
-- Result subtype: SIGNED(R'LENGTH-1 downto 0)
-- Result: Adds an INTEGER, L(may be positive or negative), to a SIGNED
-- vector, R.

-- Id: A.8
function "+" (L: SIGNED; R: INTEGER) return SIGNED;
-- Result subtype: SIGNED(L'LENGTH-1 downto 0)
-- Result: Adds a SIGNED vector, L, to an INTEGER, R.

--============================================================================
```

```
-- Id: A.9
function "-" (L, R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(MAX(L'LENGTH, R'LENGTH)-1 downto 0)
-- Result: Subtracts two UNSIGNED vectors that may be of different lengths.

-- Id: A.10
function "-" (L, R: SIGNED) return SIGNED;
-- Result subtype: SIGNED(MAX(L'LENGTH, R'LENGTH)-1 downto 0)
-- Result: Subtracts a SIGNED vector, R, from another SIGNED vector, L,
--         that may possibly be of different lengths.

-- Id: A.11
function "-" (L: UNSIGNED; R: NATURAL) return UNSIGNED;
-- Result subtype: UNSIGNED(L'LENGTH-1 downto 0)
-- Result: Subtracts a nonnegative INTEGER, R, from an UNSIGNED vector, L.

-- Id: A.12
function "-" (L: NATURAL; R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(R'LENGTH-1 downto 0)
-- Result: Subtracts an UNSIGNED vector, R, from a nonnegative INTEGER, L.
```

```
-- Id: A.13
function "-" (L: SIGNED; R: INTEGER) return SIGNED;
-- Result subtype: SIGNED(L'LENGTH-1 downto 0)
-- Result: Subtracts an INTEGER, R, from a SIGNED vector, L.

-- Id: A.14
function "-" (L: INTEGER; R: SIGNED) return SIGNED;
-- Result subtype: SIGNED(R'LENGTH-1 downto 0)
-- Result: Subtracts a SIGNED vector, R, from an INTEGER, L.

--============================================================================

-- Id: A.15
function "*" (L, R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED((L'LENGTH+R'LENGTH-1) downto 0)
-- Result: Performs the multiplication operation on two UNSIGNED vectors
--         that may possibly be of different lengths.

-- Id: A.16
function "*" (L, R: SIGNED) return SIGNED;
-- Result subtype: SIGNED((L'LENGTH+R'LENGTH-1) downto 0)
-- Result: Multiplies two SIGNED vectors that may possibly be of
--         different lengths.

-- Id: A.17
function "*" (L: UNSIGNED; R: NATURAL) return UNSIGNED;
-- Result subtype: UNSIGNED((L'LENGTH+L'LENGTH-1) downto 0)
-- Result: Multiplies an UNSIGNED vector, L, with a nonnegative
--         INTEGER, R. R is converted to an UNSIGNED vector of
--         size L'LENGTH before multiplication.
```

```
-- Id: A.18
function "*" (L: NATURAL; R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED((R'LENGTH+R'LENGTH-1) downto 0)
-- Result: Multiplies an UNSIGNED vector, R, with a nonnegative
--         INTEGER, L. L is converted to an UNSIGNED vector of
--         size R'LENGTH before multiplication.

-- Id: A.19
```



```
function "*" (L: SIGNED; R: INTEGER) return SIGNED;
-- Result subtype: SIGNED((L'LENGTH+L'LENGTH-1) downto 0)
-- Result: Multiplies a SIGNED vector, L, with an INTEGER, R. R is
--         converted to a SIGNED vector of size L'LENGTH before
--         multiplication.

-- Id: A.20
function "*" (L: INTEGER; R: SIGNED) return SIGNED;
-- Result subtype: SIGNED((R'LENGTH+R'LENGTH-1) downto 0)
-- Result: Multiplies a SIGNED vector, R, with an INTEGER, L. L is
--         converted to a SIGNED vector of size R'LENGTH before
--         multiplication.
```

```
--============================================================================
--
-- NOTE: If second argument is zero for "/" operator, a severity level
--       of ERROR is issued.

-- Id: A.21
function "/" (L, R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(L'LENGTH-1 downto 0)
-- Result: Divides an UNSIGNED vector, L, by another UNSIGNED vector, R.

-- Id: A.22
function "/" (L, R: SIGNED) return SIGNED;
-- Result subtype: SIGNED(L'LENGTH-1 downto 0)
-- Result: Divides an SIGNED vector, L, by another SIGNED vector, R.

-- Id: A.23
function "/" (L: UNSIGNED; R: NATURAL) return UNSIGNED;
-- Result subtype: UNSIGNED(L'LENGTH-1 downto 0)
-- Result: Divides an UNSIGNED vector, L, by a nonnegative INTEGER, R.
--         If NO_OF_BITS(R) > L'LENGTH, result is truncated to L'LENGTH.

-- Id: A.24
function "/" (L: NATURAL; R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(R'LENGTH-1 downto 0)
-- Result: Divides a nonnegative INTEGER, L, by an UNSIGNED vector, R.
--         If NO_OF_BITS(L) > R'LENGTH, result is truncated to R'LENGTH.

-- Id: A.25
function "/" (L: SIGNED; R: INTEGER) return SIGNED;
-- Result subtype: SIGNED(L'LENGTH-1 downto 0)
```

```
-- Result: Divides a SIGNED vector, L, by an INTEGER, R.
--         If NO_OF_BITS(R) > L'LENGTH, result is truncated to L'LENGTH.

-- Id: A.26
function "/" (L: INTEGER; R: SIGNED) return SIGNED;
-- Result subtype: SIGNED(R'LENGTH-1 downto 0)
-- Result: Divides an INTEGER, L, by a SIGNED vector, R.
--         If NO_OF_BITS(L) > R'LENGTH, result is truncated to R'LENGTH.

--============================================================================
--
-- NOTE: If second argument is zero for "rem" operator, a severity level
--       of ERROR is issued.

-- Id: A.27
function "rem" (L, R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(R'LENGTH-1 downto 0)
-- Result: Computes "L rem R" where L and R are UNSIGNED vectors.
```



```
-- Id: A.28
function "rem" (L, R: SIGNED) return SIGNED;
-- Result subtype: SIGNED(R'LENGTH-1 downto 0)
```

```
-- Result: Computes "L rem R" where L and R are SIGNED vectors.

-- Id: A.29
function "rem" (L: UNSIGNED; R: NATURAL) return UNSIGNED;
-- Result subtype: UNSIGNED(L'LENGTH-1 downto 0)
-- Result: Computes "L rem R" where L is an UNSIGNED vector and R is a
--         nonnegative INTEGER.
--         If NO_OF_BITS(R) > L'LENGTH, result is truncated to L'LENGTH.

-- Id: A.30
function "rem" (L: NATURAL; R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(R'LENGTH-1 downto 0)
-- Result: Computes "L rem R" where R is an UNSIGNED vector and L is a
--         nonnegative INTEGER.
-- If NO_OF_BITS(L) > R'LENGTH, result is truncated to R'LENGTH.

-- Id: A.31
function "rem" (L: SIGNED; R: INTEGER) return SIGNED;
-- Result subtype: SIGNED(L'LENGTH-1 downto 0)
-- Result: Computes "L rem R" where L is SIGNED vector and R is an INTEGER.
--         If NO_OF_BITS(R) > L'LENGTH, result is truncated to L'LENGTH.

-- Id: A.32
function "rem" (L: INTEGER; R: SIGNED) return SIGNED;
-- Result subtype: SIGNED(R'LENGTH-1 downto 0)
-- Result: Computes "L rem R" where R is SIGNED vector and L is an INTEGER.
--         If NO_OF_BITS(L) > R'LENGTH, result is truncated to R'LENGTH.

--============================================================================
--
```

```
-- NOTE: If second argument is zero for "mod" operator, a severity level
--       of ERROR is issued.

-- Id: A.33
function "mod" (L, R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(R'LENGTH-1 downto 0)
-- Result: Computes "L mod R" where L and R are UNSIGNED vectors.

-- Id: A.34
function "mod" (L, R: SIGNED) return SIGNED;
-- Result subtype: SIGNED(R'LENGTH-1 downto 0)
-- Result: Computes "L mod R" where L and R are SIGNED vectors.

-- Id: A.35
function "mod" (L: UNSIGNED; R: NATURAL) return UNSIGNED;
-- Result subtype: UNSIGNED(L'LENGTH-1 downto 0)
-- Result: Computes "L mod R" where L is an UNSIGNED vector and R
--         is a nonnegative INTEGER.
--         If NO_OF_BITS(R) > L'LENGTH, result is truncated to L'LENGTH.

-- Id: A.36
function "mod" (L: NATURAL; R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(R'LENGTH-1 downto 0)
-- Result: Computes "L mod R" where R is an UNSIGNED vector and L
--         is a nonnegative INTEGER.
--         If NO_OF_BITS(L) > R'LENGTH, result is truncated to R'LENGTH.
```

```
-- Id: A.37
function "mod" (L: SIGNED; R: INTEGER) return SIGNED;
-- Result subtype: SIGNED(L'LENGTH-1 downto 0)
-- Result: Computes "L mod R" where L is a SIGNED vector and
--         R is an INTEGER.
--         If NO_OF_BITS(R) > L'LENGTH, result is truncated to L'LENGTH.

-- Id: A.38
function "mod" (L: INTEGER; R: SIGNED) return SIGNED;
-- Result subtype: SIGNED(R'LENGTH-1 downto 0)
-- Result: Computes "L mod R" where L is an INTEGER and
--         R is a SIGNED vector.
--         If NO_OF_BITS(L) > R'LENGTH, result is truncated to R'LENGTH.

--============================================================================
-- Comparison Operators
--============================================================================

-- Id: C.1
function ">" (L, R: UNSIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L > R" where L and R are UNSIGNED vectors possibly
--         of different lengths.
```

```
-- Id: C.2
function ">" (L, R: SIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L > R" where L and R are SIGNED vectors possibly
--         of different lengths.

-- Id: C.3
function ">" (L: NATURAL; R: UNSIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L > R" where L is a nonnegative INTEGER and
--         R is an UNSIGNED vector.

-- Id: C.4
function ">" (L: INTEGER; R: SIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L > R" where L is a INTEGER and
--         R is a SIGNED vector.

-- Id: C.5
function ">" (L: UNSIGNED; R: NATURAL) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L > R" where L is an UNSIGNED vector and
--         R is a nonnegative INTEGER.

-- Id: C.6
function ">" (L: SIGNED; R: INTEGER) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L > R" where L is a SIGNED vector and
--         R is a INTEGER.
```

```
--============================================================================

-- Id: C.7
function "<" (L, R: UNSIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L < R" where L and R are UNSIGNED vectors possibly
--         of different lengths.
```



```
-- Id: C.8
function "<" (L, R: SIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L < R" where L and R are SIGNED vectors possibly
--         of different lengths.

-- Id: C.9
function "<" (L: NATURAL; R: UNSIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L < R" where L is a nonnegative INTEGER and
--         R is an UNSIGNED vector.

-- Id: C.10
function "<" (L: INTEGER; R: SIGNED) return BOOLEAN;
```

```
-- Result subtype: BOOLEAN
-- Result: Computes "L < R" where L is an INTEGER and
--         R is a SIGNED vector.

-- Id: C.11
function "<" (L: UNSIGNED; R: NATURAL) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L < R" where L is an UNSIGNED vector and
--         R is a nonnegative INTEGER.

-- Id: C.12
function "<" (L: SIGNED; R: INTEGER) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L < R" where L is a SIGNED vector and
--         R is an INTEGER.

--=============================================================================

-- Id: C.13
function "<=" (L, R: UNSIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L <= R" where L and R are UNSIGNED vectors possibly
--         of different lengths.

-- Id: C.14
function "<=" (L, R: SIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L <= R" where L and R are SIGNED vectors possibly
--         of different lengths.
```

```
-- Id: C.15
function "<=" (L: NATURAL; R: UNSIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L <= R" where L is a nonnegative INTEGER and
--         R is an UNSIGNED vector.

-- Id: C.16
function "<=" (L: INTEGER; R: SIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L <= R" where L is an INTEGER and
--         R is a SIGNED vector.

-- Id: C.17
function "<=" (L: UNSIGNED; R: NATURAL) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L <= R" where L is an UNSIGNED vector and
```



```
--         R is a nonnegative INTEGER.

-- Id: C.18
function "<=" (L: SIGNED; R: INTEGER) return BOOLEAN;
-- Result subtype: BOOLEAN
```

```
-- Result: Computes "L <= R" where L is a SIGNED vector and
--         R is an INTEGER.

--============================================================================

-- Id: C.19
function ">=" (L, R: UNSIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L >= R" where L and R are UNSIGNED vectors possibly
--         of different lengths.

-- Id: C.20
function ">=" (L, R: SIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L >= R" where L and R are SIGNED vectors possibly
--         of different lengths.

-- Id: C.21
function ">=" (L: NATURAL; R: UNSIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L >= R" where L is a nonnegative INTEGER and
--         R is an UNSIGNED vector.

-- Id: C.22
function ">=" (L: INTEGER; R: SIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L >= R" where L is an INTEGER and
--         R is a SIGNED vector.

-- Id: C.23
```

```
function ">=" (L: UNSIGNED; R: NATURAL) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L >= R" where L is an UNSIGNED vector and
--         R is a nonnegative INTEGER.

-- Id: C.24
function ">=" (L: SIGNED; R: INTEGER) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L >= R" where L is a SIGNED vector and
--         R is an INTEGER.

--============================================================================

-- Id: C.25
function "=" (L, R: UNSIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L = R" where L and R are UNSIGNED vectors possibly
--         of different lengths.

-- Id: C.26
function "=" (L, R: SIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L = R" where L and R are SIGNED vectors possibly
--         of different lengths.

-- Id: C.27
```

```
function "=" (L: NATURAL; R: UNSIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L = R" where L is a nonnegative INTEGER and
--         R is an UNSIGNED vector.

-- Id: C.28
function "=" (L: INTEGER; R: SIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L = R" where L is an INTEGER and
--         R is a SIGNED vector.

-- Id: C.29
function "=" (L: UNSIGNED; R: NATURAL) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L = R" where L is an UNSIGNED vector and
--         R is a nonnegative INTEGER.

-- Id: C.30
function "=" (L: SIGNED; R: INTEGER) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L = R" where L is a SIGNED vector and
--         R is an INTEGER.

--============================================================================
```

```
-- Id: C.31
function "/=" (L, R: UNSIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L /= R" where L and R are UNSIGNED vectors possibly
--         of different lengths.

-- Id: C.32
function "/=" (L, R: SIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L /= R" where L and R are SIGNED vectors possibly
--         of different lengths.

-- Id: C.33
function "/=" (L: NATURAL; R: UNSIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L /= R" where L is a nonnegative INTEGER and
--         R is an UNSIGNED vector.

-- Id: C.34
function "/=" (L: INTEGER; R: SIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L /= R" where L is an INTEGER and
--         R is a SIGNED vector.

-- Id: C.35
function "/=" (L: UNSIGNED; R: NATURAL) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L /= R" where L is an UNSIGNED vector and
--         R is a nonnegative INTEGER.
```

```
-- Id: C.36
function "/=" (L: SIGNED; R: INTEGER) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L /= R" where L is a SIGNED vector and
--         R is an INTEGER.
```



```
--============================================================================
-- Shift and Rotate Functions
--============================================================================

-- Id: S.1
function SHIFT_LEFT (ARG: UNSIGNED; COUNT: NATURAL) return UNSIGNED;
-- Result subtype: UNSIGNED(ARG'LENGTH-1 downto 0)
-- Result: Performs a shift-left on an UNSIGNED vector COUNT times.
--         The vacated positions are filled with Bit '0'.
--         The COUNT leftmost bits are lost.

-- Id: S.2
function SHIFT_RIGHT (ARG: UNSIGNED; COUNT: NATURAL) return UNSIGNED;
-- Result subtype: UNSIGNED(ARG'LENGTH-1 downto 0)
-- Result: Performs a shift-right on an UNSIGNED vector COUNT times.
```

```
--         The vacated positions are filled with Bit '0'.
--         The COUNT rightmost bits are lost.

-- Id: S.3
function SHIFT_LEFT (ARG: SIGNED; COUNT: NATURAL) return SIGNED;
-- Result subtype: SIGNED(ARG'LENGTH-1 downto 0)
-- Result: Performs a shift-left on a SIGNED vector COUNT times.
--         The vacated positions are filled with Bit '0'.
--         The COUNT leftmost bits are lost.

-- Id: S.4
function SHIFT_RIGHT (ARG: SIGNED; COUNT: NATURAL) return SIGNED;
-- Result subtype: SIGNED(ARG'LENGTH-1 downto 0)
-- Result: Performs a shift-right on a SIGNED vector COUNT times.
--         The vacated positions are filled with the leftmost bit, ARG'LEFT.
--         The COUNT rightmost bits are lost.

--============================================================================

-- Id: S.5
function ROTATE_LEFT (ARG: UNSIGNED; COUNT: NATURAL) return UNSIGNED;
-- Result subtype: UNSIGNED(ARG'LENGTH-1 downto 0)
-- Result: Performs a rotate-left of an UNSIGNED vector COUNT times.

-- Id: S.6
function ROTATE_RIGHT (ARG: UNSIGNED; COUNT: NATURAL) return UNSIGNED;
-- Result subtype: UNSIGNED(ARG'LENGTH-1 downto 0)
-- Result: Performs a rotate-right of an UNSIGNED vector COUNT times.

-- Id: S.7
function ROTATE_LEFT (ARG: SIGNED; COUNT: NATURAL) return SIGNED;
```

```
-- Result subtype: SIGNED(ARG'LENGTH-1 downto 0)
-- Result: Performs a logical rotate-left of a SIGNED vector COUNT times.

-- Id: S.8
function ROTATE_RIGHT (ARG: SIGNED; COUNT: NATURAL) return SIGNED;
-- Result subtype: SIGNED(ARG'LENGTH-1 downto 0)
-- Result: Performs a logical rotate-right of a SIGNED vector COUNT times.

--============================================================================

----------------------------------------------------------------------------
-- Note: Function S.9 is not compatible with IEEE Std 1076-1987. Comment
-- out the function (declaration and body) for IEEE Std 1076-1987 compatibility.
----------------------------------------------------------------------------
-- Id: S.9
```



```
function "sll" (ARG: UNSIGNED; COUNT: INTEGER) return UNSIGNED;
-- Result subtype: UNSIGNED(ARG'LENGTH-1 downto 0)
-- Result: SHIFT_LEFT(ARG, COUNT)

----------------------------------------------------------------------------
-- Note: Function S.10 is not compatible with IEEE Std 1076-1987. Comment
-- out the function (declaration and body) for IEEE Std 1076-1987 compatibility.
```

```
--------------------------------------------------------------------------------
-- Id: S.10
function "sll" (ARG: SIGNED; COUNT: INTEGER) return SIGNED;
-- Result subtype: SIGNED(ARG'LENGTH-1 downto 0)
-- Result: SHIFT_LEFT(ARG, COUNT)

--------------------------------------------------------------------------------
-- Note: Function S.11 is not compatible with IEEE Std 1076-1987. Comment
-- out the function (declaration and body) for IEEE Std 1076-1987 compatibility.
--------------------------------------------------------------------------------
-- Id: S.11
function "srl" (ARG: UNSIGNED; COUNT: INTEGER) return UNSIGNED;
-- Result subtype: UNSIGNED(ARG'LENGTH-1 downto 0)
-- Result: SHIFT_RIGHT(ARG, COUNT)

--------------------------------------------------------------------------------
-- Note: Function S.12 is not compatible with IEEE Std 1076-1987. Comment
-- out the function (declaration and body) for IEEE Std 1076-1987 compatibility.
--------------------------------------------------------------------------------
-- Id: S.12
function "srl" (ARG: SIGNED; COUNT: INTEGER) return SIGNED;
-- Result subtype: SIGNED(ARG'LENGTH-1 downto 0)
-- Result: SIGNED(SHIFT_RIGHT(UNSIGNED(ARG), COUNT))

--------------------------------------------------------------------------------
-- Note: Function S.13 is not compatible with IEEE Std 1076-1987. Comment
-- out the function (declaration and body) for IEEE Std 1076-1987 compatibility.
--------------------------------------------------------------------------------
-- Id: S.13
function "rol" (ARG: UNSIGNED; COUNT: INTEGER) return UNSIGNED;
-- Result subtype: UNSIGNED(ARG'LENGTH-1 downto 0)
```

```
-- Result: ROTATE_LEFT(ARG, COUNT)

------------------------------------------------------------------------
-- Note: Function S.14 is not compatible with IEEE Std 1076-1987. Comment
-- out the function (declaration and body) for IEEE Std 1076-1987 compatibility.
------------------------------------------------------------------------
-- Id: S.14
function "rol" (ARG: SIGNED; COUNT: INTEGER) return SIGNED;
-- Result subtype: SIGNED(ARG'LENGTH-1 downto 0)
-- Result: ROTATE_LEFT(ARG, COUNT)

------------------------------------------------------------------------
-- Note: Function S.15 is not compatible with IEEE Std 1076-1987. Comment
-- out the function (declaration and body) for IEEE Std 1076-1987 compatibility.
------------------------------------------------------------------------
-- Id: S.15
function "ror" (ARG: UNSIGNED; COUNT: INTEGER) return UNSIGNED;
-- Result subtype: UNSIGNED(ARG'LENGTH-1 downto 0)
-- Result: ROTATE_RIGHT(ARG, COUNT)

------------------------------------------------------------------------
-- Note: Function S.16 is not compatible with IEEE Std 1076-1987. Comment
-- out the function (declaration and body) for IEEE Std 1076-1987 compatibility.
```

```
--------------------------------------------------------------------------
-- Id: S.16
function "ror" (ARG: SIGNED; COUNT: INTEGER) return SIGNED;
-- Result subtype: SIGNED(ARG'LENGTH-1 downto 0)
-- Result: ROTATE_RIGHT(ARG, COUNT)

--==========================================================================
-- RESIZE Functions
--==========================================================================

-- Id: R.1
function RESIZE (ARG: SIGNED; NEW_SIZE: NATURAL) return SIGNED;
-- Result subtype: SIGNED(NEW_SIZE-1 downto 0)
-- Result: Resizes the SIGNED vector ARG to the specified size.
--         To create a larger vector, the new [leftmost] bit positions
--         are filled with the sign bit (ARG'LEFT). When truncating,
--         the sign bit is retained along with the rightmost part.

-- Id: R.2
function RESIZE (ARG: UNSIGNED; NEW_SIZE: NATURAL) return UNSIGNED;
-- Result subtype: UNSIGNED(NEW_SIZE-1 downto 0)
-- Result: Resizes the UNSIGNED vector ARG to the specified size.
--         To create a larger vector, the new [leftmost] bit positions
--         are filled with '0'. When truncating, the leftmost bits
--         are dropped.

--==========================================================================
-- Conversion Functions
--==========================================================================
```

```
-- Id: D.1
function TO_INTEGER (ARG: UNSIGNED) return NATURAL;
-- Result subtype: NATURAL. Value cannot be negative since parameter is an
--         UNSIGNED vector.
-- Result: Converts the UNSIGNED vector to an INTEGER.

-- Id: D.2
function TO_INTEGER (ARG: SIGNED) return INTEGER;
-- Result subtype: INTEGER
-- Result: Converts a SIGNED vector to an INTEGER.

-- Id: D.3
function TO_UNSIGNED (ARG, SIZE: NATURAL) return UNSIGNED;
-- Result subtype: UNSIGNED(SIZE-1 downto 0)
-- Result: Converts a nonnegative INTEGER to an UNSIGNED vector with
--         the specified size.

-- Id: D.4
function TO_SIGNED (ARG: INTEGER; SIZE: NATURAL) return SIGNED;
-- Result subtype: SIGNED(SIZE-1 downto 0)
-- Result: Converts an INTEGER to a SIGNED vector of the specified size.

--============================================================================
-- Logical Operators
--============================================================================

-- Id: L.1
function "not" (L: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(L'LENGTH-1 downto 0)
```

```
-- Result: Termwise inversion
```



```
-- Id: L.2
function "and" (L, R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(L'LENGTH-1 downto 0)
-- Result: Vector AND operation

-- Id: L.3
function "or" (L, R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(L'LENGTH-1 downto 0)
-- Result: Vector OR operation

-- Id: L.4
function "nand" (L, R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(L'LENGTH-1 downto 0)
-- Result: Vector NAND operation

-- Id: L.5
function "nor" (L, R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(L'LENGTH-1 downto 0)
-- Result: Vector NOR operation
```

```
-- Id: L.6
function "xor" (L, R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(L'LENGTH-1 downto 0)
-- Result: Vector XOR operation

--------------------------------------------------------------------------------
-- Note: Function L.7 is not compatible with IEEE Std 1076-1987. Comment
-- out the function (declaration and body) for IEEE Std 1076-1987 compatibility.
--------------------------------------------------------------------------------
-- Id: L.7
function "xnor" (L, R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(L'LENGTH-1 downto 0)
-- Result: Vector XNOR operation

-- Id: L.8
function "not" (L: SIGNED) return SIGNED;
-- Result subtype: SIGNED(L'LENGTH-1 downto 0)
-- Result: Termwise inversion

-- Id: L.9
function "and" (L, R: SIGNED) return SIGNED;
-- Result subtype: SIGNED(L'LENGTH-1 downto 0)
-- Result: Vector AND operation

-- Id: L.10
function "or" (L, R: SIGNED) return SIGNED;
-- Result subtype: SIGNED(L'LENGTH-1 downto 0)
-- Result: Vector OR operation

-- Id: L.11
```

```
function "nand" (L, R: SIGNED) return SIGNED;
-- Result subtype: SIGNED(L'LENGTH-1 downto 0)
-- Result: Vector NAND operation

-- Id: L.12
function "nor" (L, R: SIGNED) return SIGNED;
-- Result subtype: SIGNED(L'LENGTH-1 downto 0)
-- Result: Vector NOR operation

-- Id: L.13
function "xor" (L, R: SIGNED) return SIGNED;
```



```
-- Result subtype: SIGNED(L'LENGTH-1 downto 0)
-- Result: Vector XOR operation

------------------------------------------------------------------------
-- Note: Function L.14 is not compatible with IEEE Std 1076-1987. Comment
-- out the function (declaration and body) for IEEE Std 1076-1987 compatibility.
------------------------------------------------------------------------
-- Id: L.14
function "xnor" (L, R: SIGNED) return SIGNED;
-- Result subtype: SIGNED(L'LENGTH-1 downto 0)
-- Result: Vector XNOR operation
```

```
  --=============================================================================
  -- Edge Detection Functions
  --=============================================================================

  -- Id: E.1
  function RISING_EDGE (signal S: BIT) return BOOLEAN;
  -- Result subtype: BOOLEAN
  -- Result: Returns TRUE if an event is detected on signal S and the
  --         value changed from a '0' to a '1'.

  -- Id: E.2
  function FALLING_EDGE (signal S: BIT) return BOOLEAN;
  -- Result subtype: BOOLEAN
  -- Result: Returns TRUE if an event is detected on signal S and the
  --         value changed from a '1' to a '0'.

end NUMERIC_BIT;
```

7.3.2 Package declaration for NUMERIC_STD

```
-- ------------------------------------------------------------------
--
-- Copyright © 1997 by IEEE. All rights reserved.
--
-- This source file is an essential part of IEEE Std 1076.3-1997,
-- IEEE Standard VHDL Synthesis Packages. This source file may not be
-- copied, sold, or included with software that is sold without written
-- permission from the IEEE Standards Department. This source file may
-- be used to implement this standard and may be distributed in compiled
-- form in any manner so long as the compiled form does not allow direct
-- decompilation of the original source file. This source file may be
-- copied for individual use between licensed users. This source file is
-- provided on an AS IS basis. The IEEE disclaims ANY WARRANTY EXPRESS OR
-- IMPLIED INCLUDING ANY WARRANTY OF MERCHANTABILITY AND FITNESS FOR USE
-- FOR A PARTICULAR PURPOSE. The user of the source file shall indemnify
-- and hold IEEE harmless from any damages or liability arising out of the
-- use thereof.
--
```

```
-- This package may be modified to include additional data required by tools,
-- but it must in no way change the external interfaces or simulation behavior
-- of the description. It is permissible to add comments and/or attributes to
-- the package declarations, but not to change or delete any original lines of
-- the package declaration. The package body may be changed only in accordance
-- with the terms of 7.1 and 7.2 of this standard.
--
-- Title      : Standard VHDL Synthesis Packages (IEEE Std 1076.3-1997, NUMERIC_STD)
--
-- Library    : This package shall be compiled into a library symbolically
--            : named IEEE.
--
-- Developers : IEEE DASC Synthesis Working Group.
--
-- Purpose    : This package defines numeric types and arithmetic functions
--            : for use with synthesis tools. Two numeric types are defined:
--            : -- > UNSIGNED: represents UNSIGNED number in vector form
--            : -- > SIGNED: represents a SIGNED number in vector form
--            : The base element type is type STD_LOGIC.
--            : The leftmost bit is treated as the most significant bit.
--            : Signed vectors are represented in two's complement form.
--            : This package contains overloaded arithmetic operators on
--            : the SIGNED and UNSIGNED types. The package also contains
--            : useful type conversions functions.
--            :
--            : If any argument to a function is a null array, a null array is
--            : returned (exceptions, if any, are noted individually).
--
-- Note       : No declarations or definitions shall be included in, or
--            : excluded from this package. The "package declaration" defines
```

```
--              : the types, subtypes and declarations of NUMERIC_STD. The
--              : NUMERIC_STD package body shall be considered the formal
--              : definition of the semantics of this package. Tool developers
--              : may choose to implement the package body in the most efficient
--              : manner available to them.
--
-- ------------------------------------------------------------------
--   Modification History :
-- ------------------------------------------------------------------
--   Version:  2.4
--   Date    :  12 April 1995
```



```
-- ------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;

package NUMERIC_STD is
  constant CopyRightNotice: STRING
      := "Copyright © 1997 IEEE. All rights reserved.";

  --============================================================
  -- Numeric Array Type Definitions
  --============================================================
```

```
type UNSIGNED is array (NATURAL range <>) of STD_LOGIC;
type SIGNED is array (NATURAL range <>) of STD_LOGIC;

--============================================================================
-- Arithmetic Operators:
--============================================================================

-- Id: A.1
function "abs" (ARG: SIGNED) return SIGNED;
-- Result subtype: SIGNED(ARG'LENGTH-1 downto 0)
-- Result: Returns the absolute value of a SIGNED vector ARG.

-- Id: A.2
function "-" (ARG: SIGNED) return SIGNED;
-- Result subtype: SIGNED(ARG'LENGTH-1 downto 0)
-- Result: Returns the value of the unary minus operation on a
--         SIGNED vector ARG.

--============================================================================

-- Id: A.3
function "+" (L, R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(MAX(L'LENGTH, R'LENGTH)-1 downto 0)
-- Result: Adds two UNSIGNED vectors that may be of different lengths.

-- Id: A.4
function "+" (L, R: SIGNED) return SIGNED;
-- Result subtype: SIGNED(MAX(L'LENGTH, R'LENGTH)-1 downto 0)
-- Result: Adds two SIGNED vectors that may be of different lengths.
```

```
-- Id: A.5
function "+" (L: UNSIGNED; R: NATURAL) return UNSIGNED;
-- Result subtype: UNSIGNED(L'LENGTH-1 downto 0)
-- Result: Adds an UNSIGNED vector, L, with a nonnegative INTEGER, R.

-- Id: A.6
function "+" (L: NATURAL; R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(R'LENGTH-1 downto 0)
-- Result: Adds a nonnegative INTEGER, L, with an UNSIGNED vector, R.

-- Id: A.7
function "+" (L: INTEGER; R: SIGNED) return SIGNED;
-- Result subtype: SIGNED(R'LENGTH-1 downto 0)
-- Result: Adds an INTEGER, L(may be positive or negative), to a SIGNED
--         vector, R.

-- Id: A.8
function "+" (L: SIGNED; R: INTEGER) return SIGNED;
-- Result subtype: SIGNED(L'LENGTH-1 downto 0)
```



```
-- Result: Adds a SIGNED vector, L, to an INTEGER, R.

--============================================================================
```

```
-- Id: A.9
function "-" (L, R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(MAX(L'LENGTH, R'LENGTH)-1 downto 0)
-- Result: Subtracts two UNSIGNED vectors that may be of different lengths.

-- Id: A.10
function "-" (L, R: SIGNED) return SIGNED;
-- Result subtype: SIGNED(MAX(L'LENGTH, R'LENGTH)-1 downto 0)
-- Result: Subtracts a SIGNED vector, R, from another SIGNED vector, L,
--         that may possibly be of different lengths.

-- Id: A.11
function "-" (L: UNSIGNED;R: NATURAL) return UNSIGNED;
-- Result subtype: UNSIGNED(L'LENGTH-1 downto 0)
-- Result: Subtracts a nonnegative INTEGER, R, from an UNSIGNED vector, L.

-- Id: A.12
function "-" (L: NATURAL; R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(R'LENGTH-1 downto 0)
-- Result: Subtracts an UNSIGNED vector, R, from a nonnegative INTEGER, L.

-- Id: A.13
function "-" (L: SIGNED; R: INTEGER) return SIGNED;
-- Result subtype: SIGNED(L'LENGTH-1 downto 0)
-- Result: Subtracts an INTEGER, R, from a SIGNED vector, L.

-- Id: A.14
function "-" (L: INTEGER; R: SIGNED) return SIGNED;
-- Result subtype: SIGNED(R'LENGTH-1 downto 0)
```

```
-- Result: Subtracts a SIGNED vector, R, from an INTEGER, L.

--============================================================================

-- Id: A.15
function "*" (L, R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED((L'LENGTH+R'LENGTH-1) downto 0)
-- Result: Performs the multiplication operation on two UNSIGNED vectors
--         that may possibly be of different lengths.

-- Id: A.16
function "*" (L, R: SIGNED) return SIGNED;
-- Result subtype: SIGNED((L'LENGTH+R'LENGTH-1) downto 0)
-- Result: Multiplies two SIGNED vectors that may possibly be of
--         different lengths.

-- Id: A.17
function "*" (L: UNSIGNED; R: NATURAL) return UNSIGNED;
-- Result subtype: UNSIGNED((L'LENGTH+L'LENGTH-1) downto 0)
-- Result: Multiplies an UNSIGNED vector, L, with a nonnegative
--         INTEGER, R. R is converted to an UNSIGNED vector of
--         SIZE L'LENGTH before multiplication.

-- Id: A.18
function "*" (L: NATURAL; R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED((R'LENGTH+R'LENGTH-1) downto 0)
-- Result: Multiplies an UNSIGNED vector, R, with a nonnegative
--         INTEGER, L. L is converted to an UNSIGNED vector of
```

```
--          SIZE R'LENGTH before multiplication.

-- Id: A.19
function "*" (L: SIGNED; R: INTEGER) return SIGNED;
-- Result subtype: SIGNED((L'LENGTH+L'LENGTH-1) downto 0)
-- Result: Multiplies a SIGNED vector, L, with an INTEGER, R. R is
--         converted to a SIGNED vector of SIZE L'LENGTH before
--         multiplication.

-- Id: A.20
function "*" (L: INTEGER; R: SIGNED) return SIGNED;
-- Result subtype: SIGNED((R'LENGTH+R'LENGTH-1) downto 0)
-- Result: Multiplies a SIGNED vector, R, with an INTEGER, L. L is
--         converted to a SIGNED vector of SIZE R'LENGTH before
--         multiplication.

--============================================================================
--
-- NOTE: If second argument is zero for "/" operator, a severity level
--       of ERROR is issued.

-- Id: A.21
function "/" (L, R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(L'LENGTH-1 downto 0)
```

```
-- Result: Divides an UNSIGNED vector, L, by another UNSIGNED vector, R.

-- Id: A.22
function "/" (L, R: SIGNED) return SIGNED;
-- Result subtype: SIGNED(L'LENGTH-1 downto 0)
-- Result: Divides an SIGNED vector, L, by another SIGNED vector, R.

-- Id: A.23
function "/" (L: UNSIGNED; R: NATURAL) return UNSIGNED;
-- Result subtype: UNSIGNED(L'LENGTH-1 downto 0)
-- Result: Divides an UNSIGNED vector, L, by a nonnegative INTEGER, R.
--         If NO_OF_BITS(R) > L'LENGTH, result is truncated to L'LENGTH.

-- Id: A.24
function "/" (L: NATURAL; R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(R'LENGTH-1 downto 0)
-- Result: Divides a nonnegative INTEGER, L, by an UNSIGNED vector, R.
--         If NO_OF_BITS(L) > R'LENGTH, result is truncated to R'LENGTH.

-- Id: A.25
function "/" (L: SIGNED; R: INTEGER) return SIGNED;
-- Result subtype: SIGNED(L'LENGTH-1 downto 0)
-- Result: Divides a SIGNED vector, L, by an INTEGER, R.
--         If NO_OF_BITS(R) > L'LENGTH, result is truncated to L'LENGTH.

-- Id: A.26
function "/" (L: INTEGER; R: SIGNED) return SIGNED;
-- Result subtype: SIGNED(R'LENGTH-1 downto 0)
-- Result: Divides an INTEGER, L, by a SIGNED vector, R.
--         If NO_OF_BITS(L) > R'LENGTH, result is truncated to R'LENGTH.
```

```
--============================================================================
--
-- NOTE: If second argument is zero for "rem" operator, a severity level
--       of ERROR is issued.
```



```
-- Id: A.27
function "rem" (L, R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(R'LENGTH-1 downto 0)
-- Result: Computes "L rem R" where L and R are UNSIGNED vectors.

-- Id: A.28
function "rem" (L, R: SIGNED) return SIGNED;
-- Result subtype: SIGNED(R'LENGTH-1 downto 0)
-- Result: Computes "L rem R" where L and R are SIGNED vectors.

-- Id: A.29
function "rem" (L: UNSIGNED; R: NATURAL) return UNSIGNED;
-- Result subtype: UNSIGNED(L'LENGTH-1 downto 0)
-- Result: Computes "L rem R" where L is an UNSIGNED vector and R is a
--         nonnegative INTEGER.
--         If NO_OF_BITS(R) > L'LENGTH, result is truncated to L'LENGTH.
```

```
-- Id: A.30
function "rem" (L: NATURAL; R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(R'LENGTH-1 downto 0)
-- Result: Computes "L rem R" where R is an UNSIGNED vector and L is a
--         nonnegative INTEGER.
--         If NO_OF_BITS(L) > R'LENGTH, result is truncated to R'LENGTH.

-- Id: A.31
function "rem" (L: SIGNED; R: INTEGER) return SIGNED;
-- Result subtype: SIGNED(L'LENGTH-1 downto 0)
-- Result: Computes "L rem R" where L is SIGNED vector and R is an INTEGER.
--         If NO_OF_BITS(R) > L'LENGTH, result is truncated to L'LENGTH.

-- Id: A.32
function "rem" (L: INTEGER; R: SIGNED) return SIGNED;
-- Result subtype: SIGNED(R'LENGTH-1 downto 0)
-- Result: Computes "L rem R" where R is SIGNED vector and L is an INTEGER.
--         If NO_OF_BITS(L) > R'LENGTH, result is truncated to R'LENGTH.

--============================================================================
--
-- NOTE: If second argument is zero for "mod" operator, a severity level
--       of ERROR is issued.

-- Id: A.33
function "mod" (L, R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(R'LENGTH-1 downto 0)
-- Result: Computes "L mod R" where L and R are UNSIGNED vectors.
```

```
-- Id: A.34
function "mod" (L, R: SIGNED) return SIGNED;
-- Result subtype: SIGNED(R'LENGTH-1 downto 0)
-- Result: Computes "L mod R" where L and R are SIGNED vectors.

-- Id: A.35
function "mod" (L: UNSIGNED; R: NATURAL) return UNSIGNED;
-- Result subtype: UNSIGNED(L'LENGTH-1 downto 0)
-- Result: Computes "L mod R" where L is an UNSIGNED vector and R
--         is a nonnegative INTEGER.
--         If NO_OF_BITS(R) > L'LENGTH, result is truncated to L'LENGTH.

-- Id: A.36
function "mod" (L: NATURAL; R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(R'LENGTH-1 downto 0)
```



```
-- Result: Computes "L mod R" where R is an UNSIGNED vector and L
--         is a nonnegative INTEGER.
--         If NO_OF_BITS(L) > R'LENGTH, result is truncated to R'LENGTH.

-- Id: A.37
function "mod" (L: SIGNED; R: INTEGER) return SIGNED;
```

```
-- Result subtype: SIGNED(L'LENGTH-1 downto 0)
-- Result: Computes "L mod R" where L is a SIGNED vector and
--         R is an INTEGER.
--         If NO_OF_BITS(R) > L'LENGTH, result is truncated to L'LENGTH.

-- Id: A.38
function "mod" (L: INTEGER; R: SIGNED) return SIGNED;
-- Result subtype: SIGNED(R'LENGTH-1 downto 0)
-- Result: Computes "L mod R" where L is an INTEGER and
--         R is a SIGNED vector.
--         If NO_OF_BITS(L) > R'LENGTH, result is truncated to R'LENGTH.

--============================================================================
-- Comparison Operators
--============================================================================

-- Id: C.1
function ">" (L, R: UNSIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L > R" where L and R are UNSIGNED vectors possibly
--         of different lengths.

-- Id: C.2
function ">" (L, R: SIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L > R" where L and R are SIGNED vectors possibly
--         of different lengths.

-- Id: C.3
function ">" (L: NATURAL; R: UNSIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
```

```
-- Result: Computes "L > R" where L is a nonnegative INTEGER and
--         R is an UNSIGNED vector.

-- Id: C.4
function ">" (L: INTEGER; R: SIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L > R" where L is a INTEGER and
--         R is a SIGNED vector.

-- Id: C.5
function ">" (L: UNSIGNED; R: NATURAL) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L > R" where L is an UNSIGNED vector and
--         R is a nonnegative INTEGER.

-- Id: C.6
function ">" (L: SIGNED; R: INTEGER) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L > R" where L is a SIGNED vector and
--         R is a INTEGER.

--============================================================================
```

```
-- Id: C.7
function "<" (L, R: UNSIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L < R" where L and R are UNSIGNED vectors possibly
--         of different lengths.

-- Id: C.8
function "<" (L, R: SIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L < R" where L and R are SIGNED vectors possibly
--         of different lengths.

-- Id: C.9
function "<" (L: NATURAL; R: UNSIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L < R" where L is a nonnegative INTEGER and
--         R is an UNSIGNED vector.

-- Id: C.10
function "<" (L: INTEGER; R: SIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L < R" where L is an INTEGER and
--         R is a SIGNED vector.

-- Id: C.11
function "<" (L: UNSIGNED; R: NATURAL) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L < R" where L is an UNSIGNED vector and
--         R is a nonnegative INTEGER.
```

```
-- Id: C.12
function "<" (L: SIGNED; R: INTEGER) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L < R" where L is a SIGNED vector and
--         R is an INTEGER.

--============================================================================

-- Id: C.13
function "<=" (L, R: UNSIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L <= R" where L and R are UNSIGNED vectors possibly
--         of different lengths.

-- Id: C.14
function "<=" (L, R: SIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L <= R" where L and R are SIGNED vectors possibly
--         of different lengths.

-- Id: C.15
function "<=" (L: NATURAL; R: UNSIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L <= R" where L is a nonnegative INTEGER and
--         R is an UNSIGNED vector.

-- Id: C.16
function "<=" (L: INTEGER; R: SIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
```

```
-- Result: Computes "L <= R" where L is an INTEGER and
--         R is a SIGNED vector.
```



```
-- Id: C.17
function "<=" (L: UNSIGNED; R: NATURAL) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L <= R" where L is an UNSIGNED vector and
--         R is a nonnegative INTEGER.

-- Id: C.18
function "<=" (L: SIGNED; R: INTEGER) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L <= R" where L is a SIGNED vector and
--         R is an INTEGER.

--============================================================================

-- Id: C.19
function ">=" (L, R: UNSIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L >= R" where L and R are UNSIGNED vectors possibly
--         of different lengths.
```

```
-- Id: C.20
function ">=" (L, R: SIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L >= R" where L and R are SIGNED vectors possibly
--         of different lengths.

-- Id: C.21
function ">=" (L: NATURAL; R: UNSIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L >= R" where L is a nonnegative INTEGER and
--         R is an UNSIGNED vector.

-- Id: C.22
function ">=" (L: INTEGER; R: SIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L >= R" where L is an INTEGER and
--         R is a SIGNED vector.

-- Id: C.23
function ">=" (L: UNSIGNED; R: NATURAL) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L >= R" where L is an UNSIGNED vector and
--         R is a nonnegative INTEGER.

-- Id: C.24
function ">=" (L: SIGNED; R: INTEGER) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L >= R" where L is a SIGNED vector and
--         R is an INTEGER.
```

```
--============================================================================

-- Id: C.25
function "=" (L, R: UNSIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L = R" where L and R are UNSIGNED vectors possibly
--         of different lengths.

-- Id: C.26
function "=" (L, R: SIGNED) return BOOLEAN;
```



```
-- Result subtype: BOOLEAN
-- Result: Computes "L = R" where L and R are SIGNED vectors possibly
--         of different lengths.

-- Id: C.27
function "=" (L: NATURAL; R: UNSIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L = R" where L is a nonnegative INTEGER and
--         R is an UNSIGNED vector.

-- Id: C.28
```

```
function "=" (L: INTEGER; R: SIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L = R" where L is an INTEGER and
--         R is a SIGNED vector.

-- Id: C.29
function "=" (L: UNSIGNED; R: NATURAL) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L = R" where L is an UNSIGNED vector and
--         R is a nonnegative INTEGER.

-- Id: C.30
function "=" (L: SIGNED; R: INTEGER) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L = R" where L is a SIGNED vector and
--         R is an INTEGER.

--=============================================================================

-- Id: C.31
function "/=" (L, R: UNSIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L /= R" where L and R are UNSIGNED vectors possibly
--         of different lengths.

-- Id: C.32
function "/=" (L, R: SIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L /= R" where L and R are SIGNED vectors possibly
--         of different lengths.
```

```
-- Id: C.33
function "/=" (L: NATURAL; R: UNSIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L /= R" where L is a nonnegative INTEGER and
--         R is an UNSIGNED vector.

-- Id: C.34
function "/=" (L: INTEGER; R: SIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L /= R" where L is an INTEGER and
--         R is a SIGNED vector.

-- Id: C.35
function "/=" (L: UNSIGNED; R: NATURAL) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: Computes "L /= R" where L is an UNSIGNED vector and
--         R is a nonnegative INTEGER.

-- Id: C.36
```



```
function "/=" (L: SIGNED; R: INTEGER) return BOOLEAN;
-- Result subtype: BOOLEAN
```

```
-- Result: Computes "L /= R" where L is a SIGNED vector and
--         R is an INTEGER.

--============================================================================
-- Shift and Rotate Functions
--============================================================================

-- Id: S.1
function SHIFT_LEFT (ARG: UNSIGNED; COUNT: NATURAL) return UNSIGNED;
-- Result subtype: UNSIGNED(ARG'LENGTH-1 downto 0)
-- Result: Performs a shift-left on an UNSIGNED vector COUNT times.
--         The vacated positions are filled with '0'.
--         The COUNT leftmost elements are lost.

-- Id: S.2
function SHIFT_RIGHT (ARG: UNSIGNED; COUNT: NATURAL) return UNSIGNED;
-- Result subtype: UNSIGNED(ARG'LENGTH-1 downto 0)
-- Result: Performs a shift-right on an UNSIGNED vector COUNT times.
--         The vacated positions are filled with '0'.
--         The COUNT rightmost elements are lost.

-- Id: S.3
function SHIFT_LEFT (ARG: SIGNED; COUNT: NATURAL) return SIGNED;
-- Result subtype: SIGNED(ARG'LENGTH-1 downto 0)
-- Result: Performs a shift-left on a SIGNED vector COUNT times.
--         The vacated positions are filled with '0'.
--         The COUNT leftmost elements are lost.

-- Id: S.4
function SHIFT_RIGHT (ARG: SIGNED; COUNT: NATURAL) return SIGNED;
```

```
-- Result subtype: SIGNED(ARG'LENGTH-1 downto 0)
-- Result: Performs a shift-right on a SIGNED vector COUNT times.
--         The vacated positions are filled with the leftmost
--         element, ARG'LEFT. The COUNT rightmost elements are lost.

--============================================================================

-- Id: S.5
function ROTATE_LEFT (ARG: UNSIGNED; COUNT: NATURAL) return UNSIGNED;
-- Result subtype: UNSIGNED(ARG'LENGTH-1 downto 0)
-- Result: Performs a rotate-left of an UNSIGNED vector COUNT times.

-- Id: S.6
function ROTATE_RIGHT (ARG: UNSIGNED; COUNT: NATURAL) return UNSIGNED;
-- Result subtype: UNSIGNED(ARG'LENGTH-1 downto 0)
-- Result: Performs a rotate-right of an UNSIGNED vector COUNT times.

-- Id: S.7
function ROTATE_LEFT (ARG: SIGNED; COUNT: NATURAL) return SIGNED;
-- Result subtype: SIGNED(ARG'LENGTH-1 downto 0)
-- Result: Performs a logical rotate-left of a SIGNED
--         vector COUNT times.

-- Id: S.8
function ROTATE_RIGHT (ARG: SIGNED; COUNT: NATURAL) return SIGNED;
-- Result subtype: SIGNED(ARG'LENGTH-1 downto 0)
-- Result: Performs a logical rotate-right of a SIGNED
--         vector COUNT times.
```

```
--============================================================================

--============================================================================

------------------------------------------------------------------------------
-- Note: Function S.9 is not compatible with IEEE Std 1076-1987. Comment
-- out the function (declaration and body) for IEEE Std 1076-1987 compatibility.
------------------------------------------------------------------------------
-- Id: S.9
function "sll" (ARG: UNSIGNED; COUNT: INTEGER) return UNSIGNED;
-- Result subtype: UNSIGNED(ARG'LENGTH-1 downto 0)
-- Result: SHIFT_LEFT(ARG, COUNT)

------------------------------------------------------------------------------
-- Note: Function S.10 is not compatible with IEEE Std 1076-1987. Comment
-- out the function (declaration and body) for IEEE Std 1076-1987 compatibility.
------------------------------------------------------------------------------
-- Id: S.10
function "sll" (ARG: SIGNED; COUNT: INTEGER) return SIGNED;
-- Result subtype: SIGNED(ARG'LENGTH-1 downto 0)
-- Result: SHIFT_LEFT(ARG, COUNT)

------------------------------------------------------------------------------
```

```
-- Note: Function S.11 is not compatible with IEEE Std 1076-1987. Comment
-- out the function (declaration and body) for IEEE Std 1076-1987 compatibility.
--------------------------------------------------------------------------------
-- Id: S.11
function "srl" (ARG: UNSIGNED; COUNT: INTEGER) return UNSIGNED;
-- Result subtype: UNSIGNED(ARG'LENGTH-1 downto 0)
-- Result: SHIFT_RIGHT(ARG, COUNT)

--------------------------------------------------------------------------------
-- Note: Function S.12 is not compatible with IEEE Std 1076-1987. Comment
-- out the function (declaration and body) for IEEE Std 1076-1987 compatibility.
--------------------------------------------------------------------------------
-- Id: S.12
function "srl" (ARG: SIGNED; COUNT: INTEGER) return SIGNED;
-- Result subtype: SIGNED(ARG'LENGTH-1 downto 0)
-- Result: SIGNED(SHIFT_RIGHT(UNSIGNED(ARG), COUNT))

--------------------------------------------------------------------------------
-- Note: Function S.13 is not compatible with IEEE Std 1076-1987. Comment
-- out the function (declaration and body) for IEEE Std 1076-1987 compatibility.
--------------------------------------------------------------------------------
-- Id: S.13
function "rol" (ARG: UNSIGNED; COUNT: INTEGER) return UNSIGNED;
-- Result subtype: UNSIGNED(ARG'LENGTH-1 downto 0)
-- Result: ROTATE_LEFT(ARG, COUNT)

--------------------------------------------------------------------------------
-- Note: Function S.14 is not compatible with IEEE Std 1076-1987. Comment
-- out the function (declaration and body) for IEEE Std 1076-1987 compatibility.
--------------------------------------------------------------------------------
```

```
-- Id: S.14
function "rol" (ARG: SIGNED; COUNT: INTEGER) return SIGNED;
-- Result subtype: SIGNED(ARG'LENGTH-1 downto 0)
-- Result: ROTATE_LEFT(ARG, COUNT)

--------------------------------------------------------------------------------
-- Note: Function S.15 is not compatible with IEEE Std 1076-1987. Comment
-- out the function (declaration and body) for IEEE Std 1076-1987 compatibility.
```



```
--------------------------------------------------------------------------------
-- Id: S.15
function "ror" (ARG: UNSIGNED; COUNT: INTEGER) return UNSIGNED;
-- Result subtype: UNSIGNED(ARG'LENGTH-1 downto 0)
-- Result: ROTATE_RIGHT(ARG, COUNT)

--------------------------------------------------------------------------------
-- Note: Function S.16 is not compatible with IEEE Std 1076-1987. Comment
-- out the function (declaration and body) for IEEE Std 1076-1987 compatibility.
--------------------------------------------------------------------------------
-- Id: S.16
function "ror" (ARG: SIGNED; COUNT: INTEGER) return SIGNED;
-- Result subtype: SIGNED(ARG'LENGTH-1 downto 0)
-- Result: ROTATE_RIGHT(ARG, COUNT)
```

```
--============================================================================
-- RESIZE Functions
--============================================================================

-- Id: R.1
function RESIZE (ARG: SIGNED; NEW_SIZE: NATURAL) return SIGNED;
-- Result subtype: SIGNED(NEW_SIZE-1 downto 0)
-- Result: Resizes the SIGNED vector ARG to the specified size.
--         To create a larger vector, the new [leftmost] bit positions
--         are filled with the sign bit (ARG'LEFT). When truncating,
--         the sign bit is retained along with the rightmost part.

-- Id: R.2
function RESIZE (ARG: UNSIGNED; NEW_SIZE: NATURAL) return UNSIGNED;
-- Result subtype: UNSIGNED(NEW_SIZE-1 downto 0)
-- Result: Resizes the SIGNED vector ARG to the specified size.
--         To create a larger vector, the new [leftmost] bit positions
--         are filled with '0'. When truncating, the leftmost bits
--         are dropped.

--============================================================================
-- Conversion Functions
--============================================================================

-- Id: D.1
function TO_INTEGER (ARG: UNSIGNED) return NATURAL;
-- Result subtype: NATURAL. Value cannot be negative since parameter is an
--           UNSIGNED vector.
-- Result: Converts the UNSIGNED vector to an INTEGER.
```

```
-- Id: D.2
function TO_INTEGER (ARG: SIGNED) return INTEGER;
-- Result subtype: INTEGER
-- Result: Converts a SIGNED vector to an INTEGER.

-- Id: D.3
function TO_UNSIGNED (ARG, SIZE: NATURAL) return UNSIGNED;
-- Result subtype: UNSIGNED(SIZE-1 downto 0)
-- Result: Converts a nonnegative INTEGER to an UNSIGNED vector with
--         the specified SIZE.

-- Id: D.4
function TO_SIGNED (ARG: INTEGER; SIZE: NATURAL) return SIGNED;
-- Result subtype: SIGNED(SIZE-1 downto 0)
-- Result: Converts an INTEGER to a SIGNED vector of the specified SIZE.
```



```
--============================================================================
-- Logical Operators
--============================================================================

-- Id: L.1
```

```
function "not" (L: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(L'LENGTH-1 downto 0)
-- Result: Termwise inversion

-- Id: L.2
function "and" (L, R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(L'LENGTH-1 downto 0)
-- Result: Vector AND operation

-- Id: L.3
function "or" (L, R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(L'LENGTH-1 downto 0)
-- Result: Vector OR operation

-- Id: L.4
function "nand" (L, R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(L'LENGTH-1 downto 0)
-- Result: Vector NAND operation

-- Id: L.5
function "nor" (L, R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(L'LENGTH-1 downto 0)
-- Result: Vector NOR operation

-- Id: L.6
function "xor" (L, R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(L'LENGTH-1 downto 0)
-- Result: Vector XOR operation

-- ------------------------------------------------------------------
```

```
-- Note: Function L.7 is not compatible with IEEE Std 1076-1987. Comment
-- out the function (declaration and body) for IEEE Std 1076-1987 compatibility.
-- ----------------------------------------------------------------------
-- Id: L.7
function "xnor" (L, R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(L'LENGTH-1 downto 0)
-- Result: Vector XNOR operation

-- Id: L.8
function "not" (L: SIGNED) return SIGNED;
-- Result subtype: SIGNED(L'LENGTH-1 downto 0)
-- Result: Termwise inversion

-- Id: L.9
function "and" (L, R: SIGNED) return SIGNED;
-- Result subtype: SIGNED(L'LENGTH-1 downto 0)
-- Result: Vector AND operation

-- Id: L.10
function "or" (L, R: SIGNED) return SIGNED;
-- Result subtype: SIGNED(L'LENGTH-1 downto 0)
-- Result: Vector OR operation

-- Id: L.11
function "nand" (L, R: SIGNED) return SIGNED;
-- Result subtype: SIGNED(L'LENGTH-1 downto 0)
```

```
  -- Result: Vector NAND operation

  -- Id: L.12
  function "nor" (L, R: SIGNED) return SIGNED;
  -- Result subtype: SIGNED(L'LENGTH-1 downto 0)
  -- Result: Vector NOR operation

  -- Id: L.13
  function "xor" (L, R: SIGNED) return SIGNED;
  -- Result subtype: SIGNED(L'LENGTH-1 downto 0)
  -- Result: Vector XOR operation

  -- ------------------------------------------------------------------
  -- Note: Function L.14 is not compatible with IEEE Std 1076-1987. Comment
  -- out the function (declaration and body) for IEEE Std 1076-1987 compatibility.
  -- ------------------------------------------------------------------
  -- Id: L.14
  function "xnor" (L, R: SIGNED) return SIGNED;
  -- Result subtype: SIGNED(L'LENGTH-1 downto 0)
  -- Result: Vector XNOR operation

  --==================================================================
  -- Match Functions
  --==================================================================
```

```
-- Id: M.1
function STD_MATCH (L, R: STD_ULOGIC) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: terms compared per STD_LOGIC_1164 intent

-- Id: M.2
function STD_MATCH (L, R: UNSIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: terms compared per STD_LOGIC_1164 intent

-- Id: M.3
function STD_MATCH (L, R: SIGNED) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: terms compared per STD_LOGIC_1164 intent

-- Id: M.4
function STD_MATCH (L, R: STD_LOGIC_VECTOR) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: terms compared per STD_LOGIC_1164 intent

-- Id: M.5
function STD_MATCH (L, R: STD_ULOGIC_VECTOR) return BOOLEAN;
-- Result subtype: BOOLEAN
-- Result: terms compared per STD_LOGIC_1164 intent

--============================================================================
-- Translation Functions
--============================================================================

-- Id: T.1
```

```
  function TO_01 (S: UNSIGNED; XMAP: STD_LOGIC := '0') return UNSIGNED;
  -- Result subtype: UNSIGNED(S'RANGE)
  -- Result: Termwise, 'H' is translated to '1', and 'L' is translated
  --         to '0'. If a value other than '0'|'1'|'H'|'L' is found,
  --         the array is set to (others => XMAP), and a warning is
  --         issued.
```



```
  -- Id: T.2
  function TO_01 (S: SIGNED; XMAP: STD_LOGIC := '0') return SIGNED;
  -- Result subtype: SIGNED(S'RANGE)
  -- Result: Termwise, 'H' is translated to '1', and 'L' is translated
  --         to '0'. If a value other than '0'|'1'|'H'|'L' is found,
  --         the array is set to (others => XMAP), and a warning is
  --         issued.

end NUMERIC_STD;
```