IEEE Standard Multivalule Logic System for VHDL Model Interoperability (Std_logic_1164)

Abstract: This standard is embodied in the Std_logic_1164 package declaration and the semantics of the Std_logic_1164 body. An annex is provided to suggest ways in which one might use this package.

Keywords: Std_logic_1164, VHDL model interoperability
IEEE Standards documents are developed within the Technical Committees of 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 technical 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

IEEE Standards documents are adopted by the Institute of Electrical and Electronics Engineers without regard to whether their adoption may involve patents on articles, materials, or processes. Such adoption does not assume any liability to any patent owner, nor does it assume any obligation whatever to parties adopting the standards documents.
Introduction

This package provides a standard datatype system for the declaration of ports and signals used in modeling digital components in VHDL. Use of this package with its defined logic values and operators is intended to provide a mechanism for writing VHDL component models that have well-defined behavior when connected to other models adhering to this standard.

Development of the Std_logic_1164 package:

The work of this committee is the culmination of efforts by several groups with the same goals working over a period of over three years. The EIA (Electronic Industries Association) and the VDEG (VHDL Design Exchange Group) have been working on the problem of interoperable VHDL component models since the standardization of VHDL by the IEEE in 1987. The work at the EIA has been guided by John Wilner, Jack Kinn, and Len Finegold in their efforts to produce a specification for procuring interoperable VHDL component models. The work at the VDEG was guided by Moe Shahdad, Ghulam Nurie, and its last chair, Victor Berman, who merged this group with the IEEE Model Standards Group in order to promote a unified standard. The VDEG group has since disbanded. At present there is agreement by the IEEE P1164 and EIA 567 groups on this standard.

Between 1989 and this date, many individuals made valuable contributions to the development of this standard. At the time of approval of the standard, the members of the working group were:

William Billowitch, Chair

David Ackley  Andrew Guyler  Zainalabedin Navabi
Gordon Adshead  William A. Hanna  Sivaram Nayudu
Shishir Agarwal  John Hillawi  Wolfgang W. Nebel
David G. Agnew  Robert Hillman  Lawrence J. O’Connell
James R. Armstrong  Frederick Hinchliffe  Jan Pukite
Daniel S. Barclay  John Hines  Eric John Purslow
Victor Berman  Elchanan Herzog  SrinivasRaghvendra
Thomas H. Borgstrom  Andreas Hohl  Paul Ramondetta
Mark Brown  Andy Huang  Deborah L. Rooney
Walter H. Buckhardt  Gongwen Huang  Jacques Rouillard
Scott Calhoun  Mitsuaki Ishikawa  Ashraf M. Salem
David M. Cantwell  Takashi Kambe  Larry F. Saunders
Steven Carlson  Stanley J. Krolkoski  Paul Scheidt
Harold W. Carter  Stephen Kun  Kenneth E. Scott
Moon Jung Chung  Howard K. Lane  Moe Shadad
David Coelho  Rick Lazansky  Lee A. Shombert
Teddi Cormann  Jean Lebrun  David W. Smith
Allen Dewey  Oz Levi  Alec G. Stanculescu
Michael Dukes  Alfred Lowenstein  Balsha R. Stanisic
Len Finegold  Joseph F.P. Luhukay  Jose A. Torres
Jacques P. Flandrois  Don MacMillen  Joseph G. Tront
Alain Fonkoua  F. Eric Marschner  Cary Ussery
Geoffrey Frank  William S. McKinney  Radha Vaidyanathan
Gary Gaugler  Paul J. Menchini  James H. Vellenga
Alfred S. Gilman  Jean Mermet  Ranganadha Vemuri
Emil Girczyce  Gerald T. Michael  Karen E. Watkins
Rita Glover  Gabe Moretti  Ronald Waxman
Brent Gregory  Wolfgang Mueller  Francis Wiest
Brian Griffin
Lawrence T. Groves  Alex Zamfirescu
The following persons were on the balloting committee that approved this document for submission to the IEEE Standards Board:

- David Ackley
- Gordon Adshead
- Shishir Agarwal
- David G. Agnew
- James R. Armstrong
- Daniel S. Barclay
- Victor Berman
- Thomas H. Borgstrom
- Mark Brown
- Walter H. Buckhardt
- Scott Calhoun
- David M. Cantwell
- Steven Carlson
- Harold W. Carter
- Moon Jung Chung
- David Coelho
- Tedd Corman
- Allen Dewey
- Michael Dukes
- Len Finegold
- Jacques P. Flandrois
- Alain Fonkoua
- Geoffrey Frank
- Gary Gaugler
- Alfred S. Gilman
- Emil Girczyc
- Rita Glover
- Brent Gregory
- Brian Griffin
- Lawrence T. Groves
- Andrew Guyler
- William A. Hanna
- John Hillavi
- Robert Hillman
- Frederick Hinchcliffe
- John Hines
- Elchanaan Herzog
- Andreas Hohl
- Andy Huang
- Gongwen Huang
- Mitsuaki Ishikawa
- Takashi Kambe
- Stanley J. Krolkoski
- Stephen Kun
- Howard K. Lane
- Rick Lazansky
- Jean Lebrun
- Oz Levi
- Alfred Lowenstein
- Joseph F.P. Luhukay
- Don MacMillen
- F. Eric Marschner
- William S. McKinney
- Paul J. Menchini
- Jean Mermet
- Gerald T. Michael
- Gabe Moretti
- Wolfgang Mueller
- Marco W. Migliaro, Chair
- Donald C. Loughry, Vice Chair
- Andrew G. Salem, Secretary
- Dennis Bodson
- Paul L. Borrill
- Clyde Camp
- Donald C. Fleckenstein
- Jay Forster*
- David F. Franklin
- Ramiro Garcia
- Thomas L. Hannan
- Donald N. Heirman
- Ben C. Johnson
- Walter J. Karplus
- Ivor N. Knight
- Joseph Koepfinger*
- Irving Kolodny
- D. N. “Jim” Logothetis
- Lawrence V. McCall
- T. Don Michael*
- John L. Rankine
- Wallace S. Read
- Ronald H. Reimer
- Gary S. Robinson
- Martin V. Schneider
- Terrance R. Whittemore
- Donald W. Zipse
- *Member Emeritus

Also included are the following nonvoting IEEE Standards Board liaisons:

- Satish K. Aggarwal
- James Beall
- Richard B. Engelman
- David E. Soffrin
- Stanley Warshaw
- Adam H. Sicker

IEEE Standards Project Editor
# Contents

<table>
<thead>
<tr>
<th>CLAUSE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Overview</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Scope</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Conformance with this standard</td>
<td>1</td>
</tr>
<tr>
<td>2. Std_logic_1164 package declaration</td>
<td>2</td>
</tr>
<tr>
<td>3. Std_logic_1164 package body</td>
<td>4</td>
</tr>
<tr>
<td>Annex A Using the Std_logic_1164 Package</td>
<td>15</td>
</tr>
<tr>
<td>A.1 Value system</td>
<td>15</td>
</tr>
<tr>
<td>A.2 Handling strengths</td>
<td>15</td>
</tr>
<tr>
<td>A.3 Use of the uninitialized value</td>
<td>15</td>
</tr>
<tr>
<td>A.4 Behavioral modeling for 'U' propagation</td>
<td>16</td>
</tr>
<tr>
<td>A.5 'U's related to conditional expressions</td>
<td>16</td>
</tr>
<tr>
<td>A.6 Structural modeling with logical tables</td>
<td>16</td>
</tr>
<tr>
<td>A.7 X-handling: assignment of X’s</td>
<td>16</td>
</tr>
<tr>
<td>A.8 Modeling with don’t care’s</td>
<td>16</td>
</tr>
<tr>
<td>A.9 Resolution function</td>
<td>17</td>
</tr>
<tr>
<td>A.10 Using Std_ulogic vs. Std_logic</td>
<td>17</td>
</tr>
</tbody>
</table>
1. Overview

1.1 Scope

This standard is embodied in the Std_logic_1164 package declaration and the semantics of the Std_logic_1164 package body along with this clause 1 documentation. The information annex A is a guide to users and is not part of this standard, but suggests ways in which one might use this package.

1.2 Conformance with this standard

The following conformance rules shall apply as they pertain to the use and implementation of this standard:

a) No modifications shall be made to the package declaration whatsoever.
b) The Std_logic_1164 package body represents the formal semantics of the implementation of the Std_logic_1164 package declaration. Implementers of this package body may choose to simply compile the package body as it is; or they may choose to implement the package body in the most efficient form available to the user. Users shall not implement a semantic that differs from the formal semantic provided herein.
2. Std_logic_1164 package declaration

```
-- Title : Std_logic_1164 multivalue logic system
-- Library : This package shall be compiled into a library symbolic name IEEE.
-- Developers: IEEE model standards group (par 1164)
-- Purpose : This package defines a standard for designers to use in describing the
--           interconnection data types used in VHDL modeling.
-- Limitation: The logic system defined in this package may be insufficient for
--             modeling switched transistors, since such a requirement is out of
--             the scope of this effort. Furthermore, mathematics, primitives,
--             timing standards, etc. are considered orthogonal issues in relation
--             to this package and are therefore beyond the scope of this effort.
-- Note : No declarations or definitions shall be included in, or excluded from, this
--        package. The "package declaration" defines the types, subtypes, and
--        declarations of Std_logic_1164. The Std_logic_1164 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.

PACKAGE Std_logic_1164 IS
  -- logic state system (unresolved)
  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
  );

  -- unconstrained array of std_ulogic for use with the resolution function
  TYPE std_ulogic_vector IS ARRAY ( NATURAL RANGE <> ) OF std_ulogic;

  -- resolution function
  FUNCTION resolved ( s : std_ulogic_vector ) RETURN std_ulogic;

  -- *** industry standard logic type ***
  SUBTYPE std_logic IS resolved std_ulogic;

  -- unconstrained array of std_logic for use in declaring signal arrays
  TYPE std_logic_vector IS ARRAY ( NATURAL RANGE <> ) OF std_logic;

  -- common subtypes
  SUBTYPE X01 IS resolved std_logic RANGE 'X' TO '1'; -- ('X','0','1')
  SUBTYPE X01Z IS resolved std_logic RANGE 'X' TO 'Z'; -- ('X','0','1','Z')
  SUBTYPE UX01 IS resolved std_logic RANGE 'U' TO '1'; -- ('U','X','0','1')
  SUBTYPE UX01Z IS resolved std_logic RANGE 'U' TO 'Z'; -- ('U','X','0','1','Z')

  -- overloaded logical operators
  FUNCTION "and" ( l : std_logic; r : std_logic ) RETURN UX01;
  FUNCTION "nand" ( l : std_logic; r : std_logic ) RETURN UX01;
  FUNCTION "or" ( l : std_logic; r : std_logic ) RETURN UX01;
  FUNCTION "nor" ( l : std_logic; r : std_logic ) RETURN UX01;
```
FUNCTION "xor" ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
---
FUNCTION "xnor" ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
FUNCTION "not" ( l : std_ulogic ) RETURN UX01;
--- vectorized overloaded logical operators
---
FUNCTION "and" ( l, r : std_logic_vector ) RETURN std_logic_vector;
FUNCTION "and" ( l, r : std_ulogic_vector ) RETURN std_ulogic_vector;
FUNCTION "nand" ( l, r : std_logic_vector ) RETURN std_logic_vector;
FUNCTION "or" ( l, r : std_logic_vector ) RETURN std_logic_vector;
FUNCTION "nor" ( l, r : std_logic_vector ) RETURN std_logic_vector;
FUNCTION "xor" ( l, r : std_logic_vector ) RETURN std_logic_vector;
FUNCTION "xor" ( l, r : std_ulogic_vector ) RETURN std_ulogic_vector;
---
-- Note: The declaration and implementation of the "xnor" function is
-- specifically commented until a time at which the VHDL language has been
-- officially adopted as containing such a function. At such a point,
-- the following comments may be removed along with this notice without
-- further "official" balloting of this Std_logic_1164 package. It is
-- the intent of this effort to provide such a function once it becomes
-- available in the VHDL standard.
---
FUNCTION "xnor" ( l, r : std_logic_vector ) RETURN std_logic_vector;
FUNCTION "xnor" ( l, r : std_ulogic_vector ) RETURN std_ulogic_vector;
FUNCTION "not" ( l : std_logic_vector ) RETURN std_logic_vector;
FUNCTION "not" ( l : std_ulogic_vector ) RETURN std_ulogic_vector;
--- conversion functions
---
FUNCTION To_bit       ( s : std_ulogic; xmap : BIT := '0') RETURN BIT;
FUNCTION To_bitvector ( s : std_logic_vector ; xmap : BIT := '0') RETURN BIT_VECTOR;
FUNCTION To_bitvector ( s : std_ulogic_vector; xmap : BIT := '0') RETURN BIT_VECTOR;
FUNCTION To_StdULogic       ( b : BIT               ) RETURN std_ulogic;
FUNCTION To_StdLogicVector  ( b : BIT_VECTOR        ) RETURN std_logic_vector;
FUNCTION To_StdLogicVector  ( s : std_logic_vector  ) RETURN std_logic_vector;
FUNCTION To_StdULogicVector ( b : BIT_VECTOR        ) RETURN std_ulogic_vector;
FUNCTION To_StdULogicVector ( s : std_logic_vector  ) RETURN std_ulogic_vector;
--- strength strippers and type converters
---
FUNCTION To_X01  ( s : std_logic_vector ) RETURN  std_logic_vector;
FUNCTION To_X01  ( s : std_ulogic_vector ) RETURN  std_ulogic_vector;
FUNCTION To_X01  ( s : std_ulogic        ) RETURN  X01;
FUNCTION To_X01Z ( s : std_logic_vector ) RETURN  std_logic_vector;
FUNCTION To_X01Z ( s : std_ulogic_vector ) RETURN  std_ulogic_vector;
FUNCTION To_X01Z ( s : std_ulogic        ) RETURN  X01Z;
FUNCTION To_UX01 ( s : std_logic_vector ) RETURN  std_logic_vector;
FUNCTION To_UX01 ( s : std_ulogic_vector ) RETURN  std_ulogic_vector;
FUNCTION To_UX01 ( s : std_ulogic        ) RETURN  UX01;
--- edge detection
---
FUNCTION rising_edge  (SIGNAL s : std_ulogic) RETURN BOOLEAN;
FUNCTION falling_edge (SIGNAL s : std_ulogic) RETURN BOOLEAN;
--- object contains an unknown
---
FUNCTION Is_X ( s : std_ulogic_vector ) RETURN BOOLEAN;
FUNCTION Is_X ( s : std_logic_vector ) RETURN BOOLEAN;
FUNCTION Is_X ( s : std_ulogic        ) RETURN BOOLEAN;
END Std_logic_1164;
3. Std_logic_1164 package body

-- Title : Std_logic_1164 multivalue logic system
-- Library : This package shall be compiled into a library symbolically named IEEE.
-- Developers: IEEE model standards group (par 1164)
-- Purpose : This package defines a standard for designers used in describing the interconnection data types
--           : in VHDL modeling.
-- Limitation: The logic system defined in this package may be insufficient for modeling switched transistors,
--             : since such a requirement is out of the scope of this effort. Furthermore, mathematics, primitives,
--             : timing standards, etc., are considered orthogonal issues in relation to this package and are therefore
--             : beyond the scope of this effort.
-- Note : No declarations or definitions shall be included in, or excluded from this package. The "package declaration"
--        : defines the types, subtypes and declarations of Std_logic_1164. The Std_logic_1164 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    mod. date:   |
-- v4.200     01/02/91   |
-- --

PACKAGE BODY Std_logic_1164 IS
-------------------------------------------------------------------
-- local types
-------------------------------------------------------------------
TYPE stdlogic_1d IS ARRAY (std_ulogic) OF std_ulogic;
TYPE stdlogic_table IS ARRAY(std_ulogic, std_ulogic) OF std_ulogic;
-------------------------------------------------------------------
-- resolution function
-------------------------------------------------------------------
CONSTANT resolution_table : stdlogic_table := (
  --       ---------------------------------------------------------
  --       |  U    X    0    1    Z    W    L    H    -        |   |
  --       ---------------------------------------------------------
  --      |  U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), -- | U |
  -- ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- | X |
  -- ( 'U', 'X', '0', 'X', '0', '0', '0', '0', 'X' ), -- | 0 |
  -- ( 'U', 'X', '0', '1', '1', '1', '1', '1', 'X' ), -- | 1 |
  -- ( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X' ), -- | Z |
  -- ( 'U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X' ), -- | W |
  -- ( 'U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X' ), -- | L |
  -- ( 'U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X' ), -- | H |
  -- ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' )  -- | - |
);

FUNCTION resolved ( s : std_ulogic_vector ) RETURN std_ulogic IS
  VARIABLE result : std_ulogic := 'Z';  -- weakest state default
BEGIN
  -- the test for a single driver is essential; otherwise, the
  -- loop would return 'X' for a single driver of '-' and that
  -- would conflict with the value of a single driver unresolved
  -- signal.
  IF (s'LENGTH = 1) THEN    RETURN s(s'LOW);
ELSE
  FOR i IN s'RANGE LOOP
    result := resolution_table(result, s(i));
  END LOOP;
  END IF;
  RETURN result;
END resolved;
-- tables for logical operations
-------------------------------------------------------------------
-- truth table for "and" function
CONSTANT and_table : stdlogic_table := (  
  ( 'U', 'U', '0', '1', 'Z', 'W', 'L', 'H', 'U' ), -- U  
  ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ), -- X  
  ( '0', '0', '0', '0', '0', '0', '0', '0', '0' ), -- 0  
  ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ), -- 1  
  ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ), -- Z  
  ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ), -- W  
  ( '0', '0', '0', '0', '0', '0', '0', '0', '0' ), -- L  
  ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ), -- H  
  ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' )   -- -  
);

-- truth table for "or" function
CONSTANT or_table : stdlogic_table := (  
  ( 'U', 'U', 'U', '1', 'U', 'U', 'U', '1', 'U' ), -- U  
  ( 'U', 'X', 'X', '1', 'X', 'X', 'X', '1', 'X' ), -- X  
  ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ), -- 0  
  ( '1', '1', '1', '1', '1', '1', '1', '1', '1' ), -- 1  
  ( 'U', 'X', 'X', '1', 'X', 'X', '0', '1', 'X' ), -- Z  
  ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ), -- W  
  ( '1', '1', '1', '1', '1', '1', '1', '1', '1' ), -- L  
  ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ), -- H  
  ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' )   -- -  
);

-- truth table for "xor" function
CONSTANT xor_table : stdlogic_table := (  
  ( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), -- U  
  ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- X  
  ( 'U', 'X', '1', '0', 'X', 'X', '1', '0', 'X' ), -- 0  
  ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- Z  
  ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- W  
  ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- L  
  ( 'U', 'X', '1', '0', 'X', 'X', '1', '0', 'X' ), -- H  
  ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' )   -- -  
);

-- truth table for "not" function
CONSTANT not_table: stdlogic_1d :=  
  ( 'U', 'X', '1', '0', 'X', 'X', '1', '0', 'X' );
-------------------------------------------------------------------
-- overloaded logical operators ( with optimizing hints )
FUNCTION "and"  ( l : std_ulogic; r : std_ulogic ) RETURN UX01 IS  
BEGIN  
RETURN (and_table(l, r));  
END "and";

FUNCTION "nand" ( l : std_ulogic; r : std_ulogic ) RETURN UX01 IS  
BEGIN  
RETURN  (not_table ( and_table(l, r)));  
END "nand";

FUNCTION "or"   ( l : std_ulogic; r : std_ulogic ) RETURN UX01 IS  
BEGIN  
RETURN (or_table(l, r));  
END "or";

FUNCTION "nor"  ( l : std_ulogic; r : std_ulogic ) RETURN UX01 IS  
BEGIN  
RETURN  (not_table ( or_table( l, r )));  
END "nor";

FUNCTION "xor"  ( l : std_ulogic; r : std_ulogic ) RETURN UX01 IS  
BEGIN  
RETURN (xor_table(l, r));  
END "xor";
-- FUNCTION "xnor" ( l : std_ulogic; r : std_ulogic ) RETURN UX01 is
-- begin
--   return not_table(xor_table(l, r));
-- end "xnor";
FUNCTION "not" ( l : std_ulogic ) RETURN UX01 IS
BEGIN
   RETURN (not_table(l));
END "not";

FUNCTION "and" ( l,r : std_logic_vector ) RETURN std_logic_vector IS
   ALIAS lv : std_logic_vector ( 1 TO l'LENGTH ) IS l;
   ALIAS rv : std_logic_vector ( 1 TO r'LENGTH ) IS r;
   VARIABLE result : std_logic_vector ( 1 TO l'LENGTH );
BEGIN
   IF ( l'LENGTH /= r'LENGTH ) THEN
      ASSERT FALSE
      REPORT "arguments of overloaded 'and' operator are not of the same length"
      SEVERITY FAILURE;
   ELSE
      FOR i IN result'RANGE LOOP
         result(i) := and_table (lv(i), rv(i));
      END LOOP;
      RETURN result;
   END IF;
END "and";

FUNCTION "nand" ( l,r : std_logic_vector ) RETURN std_logic_vector IS
   ALIAS lv : std_logic_vector ( 1 TO l'LENGTH ) IS l;
   ALIAS rv : std_logic_vector ( 1 TO r'LENGTH ) IS r;
   VARIABLE result : std_logic_vector ( 1 TO l'LENGTH );
BEGIN
   IF ( l'LENGTH /= r'LENGTH ) THEN
      ASSERT FALSE
      REPORT "arguments of overloaded 'nand' operator are not of the same length"
      SEVERITY FAILURE;
   ELSE
      FOR i IN result'RANGE LOOP
         result(i) := not_table(and_table (lv(i), rv(i)));
      END LOOP;
      RETURN result;
   END IF;
END "nand";
FUNCTION "or" ( l,r : std_logic_vector ) RETURN std_logic_vector IS  
  ALIAS lv : std_logic_vector ( l'LENGTH ) IS l;  
  ALIAS rv : std_logic_vector ( r'LENGTH ) IS r;  
  VARIABLE result : std_logic_vector ( l'LENGTH );  
BEGIN  
  IF ( l'LENGTH /= r'LENGTH ) THEN  
    ASSERT FALSE
    REPORT "arguments of overloaded 'or' operator are not of the same length"  
    SEVERITY FAILURE;
  ELSE  
    FOR i IN result'RANGE LOOP  
      result(i) := or_table (lv(i), rv(i));
    END LOOP;
  END IF;
  RETURN result;
END "or";

FUNCTION "or" ( l,r : std_ulogic_vector ) RETURN std_ulogic_vector IS  
  ALIAS lv : std_ulogic_vector ( l'LENGTH ) IS l;  
  ALIAS rv : std_ulogic_vector ( r'LENGTH ) IS r;  
  VARIABLE result : std_ulogic_vector ( l'LENGTH );  
BEGIN  
  IF ( l'LENGTH /= r'LENGTH ) THEN  
    ASSERT FALSE
    REPORT "arguments of overloaded 'or' operator are not of the same length"  
    SEVERITY FAILURE;
  ELSE  
    FOR i IN result'RANGE LOOP  
      result(i) := or_table (lv(i), rv(i));
    END LOOP;
  END IF;
  RETURN result;
END "or";

FUNCTION "nor" ( l,r : std_logic_vector ) RETURN std_logic_vector IS  
  ALIAS lv : std_logic_vector ( l'LENGTH ) IS l;  
  ALIAS rv : std_logic_vector ( r'LENGTH ) IS r;  
  VARIABLE result : std_logic_vector ( l'LENGTH );  
BEGIN  
  IF ( l'LENGTH /= r'LENGTH ) THEN  
    ASSERT FALSE
    REPORT "arguments of overloaded 'nor' operator are not of the same length"  
    SEVERITY FAILURE;
  ELSE  
    FOR i IN result'RANGE LOOP  
      result(i) := not_table(or_table (lv(i), rv(i)));  
    END LOOP;
  END IF;
  RETURN result;
END "nor";

FUNCTION "nor" ( l,r : std_logic_vector ) RETURN std_logic_vector IS  
  ALIAS lv : std_logic_vector ( l'LENGTH ) IS l;  
  ALIAS rv : std_logic_vector ( r'LENGTH ) IS r;  
  VARIABLE result : std_logic_vector ( l'LENGTH );  
BEGIN  
  IF ( l'LENGTH /= r'LENGTH ) THEN  
    ASSERT FALSE
    REPORT "arguments of overloaded 'nor' operator are not of the same length"  
    SEVERITY FAILURE;
  ELSE  
    FOR i IN result'RANGE LOOP  
      result(i) := not_table(or_table (lv(i), rv(i)));  
    END LOOP;
  END IF;
  RETURN result;
END "nor";
FUNCTION "xor" ( l,r : std_logic_vector ) RETURN std_logic_vector IS
  ALIAS lv : std_logic_vector ( 1 TO l'LENGTH ) IS l;
  ALIAS rv : std_logic_vector ( 1 TO r'LENGTH ) IS r;
  VARIABLE result : std_logic_vector ( 1 TO l'LENGTH );
BEGIN
  IF ( l'LENGTH /= r'LENGTH ) THEN
    ASSERT FALSE
    REPORT "arguments of overloaded 'xor' operator are not of the same length"
    SEVERITY FAILURE;
  ELSE
    FOR i IN result'RANGE LOOP
      result(i) := xor_table (lv(i), rv(i));
    END LOOP;
  END IF;
  RETURN result;
END "xor";

FUNCTION "xor" ( l,r : std_ulogic_vector ) RETURN std_logic_vector IS
  ALIAS lv : std_logic_vector ( 1 TO l'LENGTH ) IS l;
  ALIAS rv : std_logic_vector ( 1 TO r'LENGTH ) IS r;
  VARIABLE result : std_logic_vector ( 1 TO l'LENGTH );
BEGIN
  IF ( l'LENGTH /= r'LENGTH ) THEN
    ASSERT FALSE
    REPORT "arguments of overloaded 'xor' operator are not of the same length"
    SEVERITY FAILURE;
  ELSE
    FOR i IN result'RANGE LOOP
      result(i) := xor_table (lv(i), rv(i));
    END LOOP;
  END IF;
  RETURN result;
END "xor";

-- xor

FUNCTION "xnor" ( l,r : std_logic_vector ) RETURN std_logic_vector IS
  ALIAS lv : std_logic_vector ( 1 TO l'LENGTH ) IS l;
  ALIAS rv : std_logic_vector ( 1 TO r'LENGTH ) IS r;
  VARIABLE result : std_logic_vector ( 1 TO l'LENGTH );
BEGIN
  IF ( l'LENGTH /= r'LENGTH ) THEN
    ASSERT FALSE
    REPORT "arguments of overloaded 'xnor' operator are not of the same length"
    SEVERITY FAILURE;
  ELSE
    FOR i IN result'RANGE LOOP
      result(i) := not_table(xor_table (lv(i), rv(i)));
    END LOOP;
  END IF;
  RETURN result;
END "xnor";

FUNCTION "xnor" ( l,r : std_ulogic_vector ) RETURN std_logic_vector IS
  ALIAS lv : std_logic_vector ( 1 TO l'LENGTH ) IS l;
  ALIAS rv : std_logic_vector ( 1 TO r'LENGTH ) IS r;
  VARIABLE result : std_logic_vector ( 1 TO l'LENGTH );
BEGIN
  IF ( l'LENGTH /= r'LENGTH ) THEN
    ASSERT FALSE
    REPORT "arguments of overloaded 'xnor' operator are not of the same length"
    SEVERITY FAILURE;
  ELSE
    FOR i IN result'RANGE LOOP
      result(i) := not_table(xor_table (lv(i), rv(i)));
    END LOOP;
  END IF;
  RETURN result;
END "xnor";

-- xnor

Note: The declaration and implementation of the "xnor" function is
-- specifically commented until a time at which the VHDL language has been
-- officially adopted as containing such a function. At such a point,
-- the following comments may be removed along with this notice without
-- further "official" balloting of this std_logic_1164 package. It is
-- the intent of this effort to provide such a function once it becomes
-- available in the VHDL standard.
FUNCTION "not"  ( l : std_logic_vector ) RETURN std_logic_vector IS
  ALIAS lv : std_logic_vector ( 1 TO l'LENGTH ) IS l;
  VARIABLE result : std_logic_vector ( 1 TO l'LENGTH ) := (OTHERS => 'X');
BEGIN
  FOR i IN result'RANGE LOOP
    result(i) := not_table( lv(i) );
  END LOOP;
  RETURN result;
END;
---------------------------------------------------------------------
FUNCTION "not"  ( l : std_ulogic_vector ) RETURN std_ulogic_vector IS
  ALIAS lv : std_ulogic_vector ( 1 TO l'LENGTH ) IS l;
  VARIABLE result : std_ulogic_vector ( 1 TO l'LENGTH ) := (OTHERS => 'X');
BEGIN
  FOR i IN result'RANGE LOOP
    result(i) := not_table( lv(i) );
  END LOOP;
  RETURN result;
END;
-------------------------------------------------------------------
-- conversion tables
-------------------------------------------------------------------
TYPE logic_x01_table IS ARRAY (std_ulogic'LOW TO std_ulogic'HIGH) OF X01;
TYPE logic_x01z_table IS ARRAY (std_ulogic'LOW TO std_ulogic'HIGH) OF X01Z;
TYPE logic_ux01_table IS ARRAY (std_ulogic'LOW TO std_ulogic'HIGH) OF UX01;
-------------------------------------------------------------------
-- table name : cvt_to_x01
-- parameters :
--        in  :  std_ulogic  -- some logic value
-- returns    :  x01         -- state value of logic value
-- purpose    :  to convert state-strength to state only
-- example    : if (cvt_to_x01 (input_signal) = '1' ) then ...
-------------------------------------------------------------------
CONSTANT cvt_to_x01 : logic_x01_table := (
  'X', -- 'U'
  'X', -- 'X'
  '0', -- '0'
  '1', -- '1'
  'X', -- 'Z'
  'X', -- 'W'
  '0', -- 'L'
  '1', -- 'H'
  'X'   -- '-'
);
-------------------------------------------------------------------
-- table name : cvt_to_x01z
-- parameters :
--        in  :  std_ulogic  -- some logic value
-- returns    :  x01z        -- state value of logic value
-- purpose    :  to convert state-strength to state only
-- example    : if (cvt_to_x01z (input_signal) = '1' ) then ...
-------------------------------------------------------------------
CONSTANT cvt_to_x01z : logic_x01z_table := (
  'X', -- 'U'
  'X', -- 'X'
  '0', -- '0'
  '1', -- '1'
  'Z', -- 'Z'
  'W', -- 'W'
  '0', -- 'L'
  '1', -- 'H'
  'X'   -- '-'
);
-------------------------------------------------------------------
-- table name : cvt_to_ux01
-- parameters :
--        in  :  std_ulogic  -- some logic value
-- returns    :  ux01        -- state value of logic value
-- purpose    :  to convert state-strength to state only
-- example    : if (cvt_to_ux01 (input_signal) = '1' ) then ...
-------------------------------------------------------------------
CONSTANT cvt_to_ux01 : logic_ux01_table := (
  'X', -- 'U'
  'X', -- 'X'
  '0', -- '0'
  '1', -- '1'
  'Z', -- 'Z'
  'W', -- 'W'
  '0', -- 'L'
  '1', -- 'H'
  'X'   -- '-'
);
CONSTANT cvt_to_ux01 : logic_ux01_table := ( 
'U', -- 'U'
'X', -- 'X'
'0', -- '0'
'1', -- '1'
'X', -- 'Z'
'X', -- 'W'
'0', -- 'L'
'1', -- 'H'
'X', -- '-' 
);
-------------------------------------------------------------------
-- conversion functions
-------------------------------------------------------------------
FUNCTION To_bit       ( s : std_ulogic;        xmap : BIT := '0') RETURN BIT IS
BEGIN
CASE s IS
WHEN '0' | 'L' => RETURN ('0');
WHEN '1' | 'H' => RETURN ('1');
WHEN OTHERS => RETURN xmap;
END CASE;
END;
--------------------------------------------------------------------
FUNCTION To_bitvector ( s : std_logic_vector ; xmap : BIT := '0') RETURN BIT_VECTOR IS
ALIAS sv : std_logic_vector ( s'LENGTH-1 DOWNTO 0 ) IS s;
VARIABLE result : BIT_VECTOR ( s'LENGTH-1 DOWNTO 0 );
BEGIN
FOR i IN result'RANGE LOOP
CASE sv(i) IS
WHEN '0' | 'L' => result(i) := '0';
WHEN '1' | 'H' => result(i) := '1';
WHEN OTHERS => result(i) := xmap;
END CASE;
END LOOP;
RETURN result;
END;
--------------------------------------------------------------------
FUNCTION To_bitvector ( s : std_ulogic_vector; xmap : BIT := '0') RETURN BIT_VECTOR IS
ALIAS sv : std_ulogic_vector ( s'LENGTH-1 DOWNTO 0 ) IS s;
VARIABLE result : BIT_VECTOR ( s'LENGTH-1 DOWNTO 0 );
BEGIN
FOR i IN result'RANGE LOOP
CASE sv(i) IS
WHEN '0' | 'L' => result(i) := '0';
WHEN '1' | 'H' => result(i) := '1';
WHEN OTHERS => result(i) := xmap;
END CASE;
END LOOP;
RETURN result;
END;
--------------------------------------------------------------------
FUNCTION To_StdULogic       ( b : BIT               ) RETURN std_ulogic IS
BEGIN
CASE b IS
WHEN '0' => RETURN '0';
WHEN '1' => RETURN '1';
END CASE;
END;
--------------------------------------------------------------------
FUNCTION To_StdLogicVector  ( b : BIT_VECTOR        ) RETURN std_logic_vector IS
ALIAS bv : BIT_VECTOR ( b'LENGTH-1 DOWNTO 0 ) IS b;
VARIABLE result : std_logic_vector ( b'LENGTH-1 DOWNTO 0 );
BEGIN
FOR i IN result'RANGE LOOP
CASE bv(i) IS
WHEN '0' => result(i) := '0';
WHEN '1' => result(i) := '1';
END CASE;
END LOOP;
RETURN result;
END;
--------------------------------------------------------------------
FUNCTION To_StdLogicVector  ( s : std_ulogic_vector ) RETURN std_logic_vector IS
ALIAS sv : std_ulogic_vector ( s'LENGTH-1 DOWNTO 0 ) IS s;
VARIABLE result : std_logic_vector ( s'LENGTH-1 DOWNTO 0 );
BEGIN
FOR i IN result'RANGE LOOP
result(i) := sv(i);
END LOOP;
RETURN result;
END;
FUNCTION To_StdULogicVector ( b : BIT_VECTOR ) RETURN std_ulogic_vector IS
  ALIAS bv : BIT_VECTOR ( b'LENGTH-1 DOWNTO 0 ) IS b;
  VARIABLE result : std_ulogic_vector ( b'LENGTH-1 DOWNTO 0 );
BEGIN
  FOR i IN result'Range LOOP
    CASE bv(i) IS
      WHEN '0' => result(i) := '0';
      WHEN '1' => result(i) := '1';
    END CASE;
  END LOOP;
  RETURN result;
END;

FUNCTION To_StdULogicVector ( s : std_logic_vector ) RETURN std_ulogic_vector IS
  ALIAS sv : std_logic_vector ( s'LENGTH-1 DOWNTO 0 ) IS s;
  VARIABLE result : std_ulogic_vector ( s'LENGTH-1 DOWNTO 0 );
BEGIN
  FOR i IN result'Range LOOP
    result(i) := sv(i);
  END LOOP;
  RETURN result;
END;

-- strength strippers and type convertors

-- to_x01

FUNCTION To_X01 ( s : std_logic_vector ) RETURN std_logic_vector IS
  ALIAS sv : std_logic_vector ( 1 TO s'LENGTH ) IS s;
  VARIABLE result : std_logic_vector ( 1 TO s'LENGTH );
BEGIN
  FOR i IN result'Range LOOP
    result(i) := cvt_to_x01 (sv(i));
  END LOOP;
  RETURN result;
END;

FUNCTION To_X01 ( s : std_ulogic_vector ) RETURN std_ulogic_vector IS
  ALIAS sv : std_ulogic_vector ( 1 TO s'LENGTH ) IS s;
  VARIABLE result : std_ulogic_vector ( 1 TO s'LENGTH );
BEGIN
  FOR i IN result'Range LOOP
    result(i) := cvt_to_x01 (sv(i));
  END LOOP;
  RETURN result;
END;

FUNCTION To_X01 ( s : std_logic ) RETURN X01 IS
BEGIN
  RETURN (cvt_to_x01(s));
END;

FUNCTION To_X01 ( b : BIT_VECTOR ) RETURN std_logic_vector IS
  ALIAS bv : BIT_VECTOR ( 1 TO b'LENGTH ) IS b;
  VARIABLE result : std_logic_vector ( 1 TO b'LENGTH );
BEGIN
  FOR i IN result'Range LOOP
    CASE bv(i) IS
      WHEN '0' => result(i) := '0';
      WHEN '1' => result(i) := '1';
    END CASE;
  END LOOP;
  RETURN result;
END;

FUNCTION To_X01 ( b : BIT_VECTOR ) RETURN std_ulogic_vector IS
  ALIAS bv : BIT_VECTOR ( 1 TO b'LENGTH ) IS b;
  VARIABLE result : std_ulogic_vector ( 1 TO b'LENGTH );
BEGIN
  FOR i IN result'Range LOOP
    CASE bv(i) IS
      WHEN '0' => result(i) := '0';
      WHEN '1' => result(i) := '1';
    END CASE;
  END LOOP;
  RETURN result;
END;
FUNCTION To_X01 ( b : BIT ) RETURN X01 IS
BEGIN
  CASE b IS
    WHEN '0' => RETURN('0');
    WHEN '1' => RETURN('1');
  END CASE;
END;

-- to_x01z

FUNCTION To_X01Z ( s : std_logic_vector ) RETURN std_logic_vector IS
  ALIAS sv : std_logic_vector (1 TO s'LENGTH) IS s;
  VARIABLE result : std_logic_vector (1 TO s'LENGTH);
BEGIN
  FOR i IN result'RANGE LOOP
    result(i) := cvt_to_x01z(sv(i));
  END LOOP;
  RETURN result;
END;

FUNCTION To_X01Z ( b : BIT_VECTOR ) RETURN std_logic_vector IS
  ALIAS bv : BIT_VECTOR (1 TO b'LENGTH) IS b;
  VARIABLE result : std_logic_vector (1 TO b'LENGTH);
BEGIN
  FOR i IN result'RANGE LOOP
    CASE bv(i) IS
      WHEN '0' => result(i) := '0';
      WHEN '1' => result(i) := '1';
    END CASE;
  END LOOP;
  RETURN result;
END;

FUNCTION To_X01Z ( s : std_ulogic_vector ) RETURN std_ulogic_vector IS
  ALIAS sv : std_ulogic_vector (1 TO s'LENGTH) IS s;
  VARIABLE result : std_ulogic_vector (1 TO s'LENGTH);
BEGIN
  FOR i IN result'RANGE LOOP
    result(i) := cvt_to_x01z(sv(i));
  END LOOP;
  RETURN result;
END;

FUNCTION To_X01Z ( b : BIT_VECTOR ) RETURN std_ulogic_vector IS
  ALIAS bv : BIT_VECTOR (1 TO b'LENGTH) IS b;
  VARIABLE result : std_ulogic_vector (1 TO b'LENGTH);
BEGIN
  FOR i IN result'RANGE LOOP
    CASE bv(i) IS
      WHEN '0' => result(i) := '0';
      WHEN '1' => result(i) := '1';
    END CASE;
  END LOOP;
  RETURN result;
END;

FUNCTION To_X01Z ( b : BIT ) RETURN X01Z IS
BEGIN
  CASE b IS
    WHEN '0' => RETURN('0');
    WHEN '1' => RETURN('1');
  END CASE;
END;

-- to_ux01

FUNCTION To_UX01 ( s : std_logic_vector ) RETURN std_logic_vector IS
  ALIAS sv : std_logic_vector (1 TO s'LENGTH) IS s;
  VARIABLE result : std_logic_vector (1 TO s'LENGTH);
BEGIN
  FOR i IN result'RANGE LOOP
    result(i) := cvt_to_ux01(sv(i));
  END LOOP;
  RETURN result;
END;

FUNCTION To_UX01 ( b : BIT_VECTOR ) RETURN std_logic_vector IS
  ALIAS bv : BIT_VECTOR (1 TO b'LENGTH) IS b;
  VARIABLE result : std_logic_vector (1 TO b'LENGTH);
BEGIN
  FOR i IN result'RANGE LOOP
    CASE bv(i) IS
      WHEN '0' => result(i) := '0';
      WHEN '1' => result(i) := '1';
    END CASE;
  END LOOP;
  RETURN result;
END;

FUNCTION To_UX01 ( b : BIT ) RETURN X01Z IS
BEGIN
  CASE b IS
    WHEN '0' => RETURN('0');
    WHEN '1' => RETURN('1');
  END CASE;
END;
FUNCTION To_UX01 ( s : std_ulogic_vector ) RETURN std_ulogic_vector IS
  ALIAS sv : std_ulogic_vector ( 1 TO s'LENGTH ) IS s;
  VARIABLE result : std_ulogic_vector ( 1 TO s'LENGTH );
BEGIN
  FOR i IN result'RANGE LOOP
    result(i) := cvt_to_ux01 (sv(i));
  END LOOP;
  RETURN result;
END;

FUNCTION To_UX01 ( s : std_ulogic ) RETURN UX01 IS
BEGIN
  RETURN (cvt_to_ux01(s));
END;

FUNCTION To_UX01 ( b : BIT_VECTOR ) RETURN std_logic_vector IS
  ALIAS bv : BIT_VECTOR ( 1 TO b'LENGTH ) IS b;
  VARIABLE result : std_logic_vector ( 1 TO b'LENGTH );
BEGIN
  FOR i IN result'RANGE LOOP
    CASE bv(i) IS
      WHEN '0' => result(i) := '0';
      WHEN '1' => result(i) := '1';
    END CASE;
  END LOOP;
  RETURN result;
END;

FUNCTION To_UX01 ( b : BIT ) RETURN UX01 IS
BEGIN
  CASE b IS
    WHEN '0' => RETURN('0');
    WHEN '1' => RETURN('1');
  END CASE;
END;

-- edge detection
FUNCTION rising_edge (SIGNAL s : std_ulogic) RETURN BOOLEAN IS
BEGIN
  RETURN (s'EVENT AND (To_X01(s) = '1') AND
          (To_X01(s'LAST_VALUE) = '0'));
END;

FUNCTION falling_edge (SIGNAL s : std_ulogic) RETURN BOOLEAN IS
BEGIN
  RETURN (s'EVENT AND (To_X01(s) = '0') AND
          (To_X01(s'LAST_VALUE) = '1'));
END;

-- object contains an unknown
FUNCTION Is_X ( s : std_ulogic_vector ) RETURN BOOLEAN IS
BEGIN
  FOR i IN s'RANGE LOOP
    CASE s(i) IS
      WHEN 'U' | 'X' | 'Z' | 'W' | '-' => RETURN TRUE;
      WHEN OTHERS => NULL;
    END CASE;
  END LOOP;
  RETURN FALSE;
END;
FUNCTION Is_X (s : std_logic_vector) RETURN BOOLEAN IS
BEGIN
  FOR i IN s' Range LOOP
    CASE s(i) IS
      WHEN 'U' | 'X' | 'Z' | 'W' | '-' => RETURN TRUE;
      WHEN OTHERS => NULL;
    END CASE;
  END LOOP;
  RETURN FALSE;
END;

FUNCTION Is_X (s : std_ulogic) RETURN BOOLEAN IS
BEGIN
  CASE s IS
    WHEN 'U' | 'X' | 'Z' | 'W' | '-' => RETURN TRUE;
    WHEN OTHERS => NULL;
  END CASE;
  RETURN FALSE;
END;
END std_logic_1164;
Annex A
Using the Std_logic_1164 Package

(informative)

This annex is intended to be a brief guide to using the Std_logic_1164 package. As a standard, this package provides a means of building models that interoperate, provided that the user adheres to a set of guidelines required by the strict typing imposed by the VHDL language.

A.1 Value system

The value system in Std_logic_1164 was developed to model a variety of digital device technologies. The base type of the logic system is named “std_ulogic” where the “u” in the name signifies “unresolved.” Each of the elements comprising the type have a specified semantic and a commonly used application. In order for models to properly interoperate, one must interpret the meaning of each of the elements as provided by the standard.

Type std_ulogic is {
    'U', Uninitialized state Used as a default value
    'X', Forcing Unknown Bus contentions, error conditions, etc.
    '0', Forcing Zero Transistor driven to GND
    '1', Forcing One Transistor driven to VCC
    'Z', High Impedance 3-state buffer outputs
    'W', Weak Unknown Bus terminators
    'L', Weak Zero Pull down resistors
    'H', Weak One Pull up resistors
    '-' Don’t Care Used for synthesis and advanced modeling
};

A.2 Handling strengths

Behavioral modeling techniques rarely require knowledge of the strength of a signal’s value. Therefore, a number of “strength stripper” functions have been designed to transform 'Z', 'W', 'L', 'H', and '-' into their corresponding “forcing” strength counterparts.

Once in forcing strength, the model can simply respond to X’s, 0’s, 1’s and 'U's as the need may arise. This strength stripping is done by using one of the following functions:

To_X01 (...) converts 'L' and 'H' to 0 and 1 respectively. All others are converted to 'X'.

To_UX01 (...) converts 'L' and 'H' to 0 and 1 respectively. 'U's are propagated and all others are converted to 'X'.

A.3 Use of the uninitialized value

The 'U' value is located in the first position of the type. Therefore, any object declared to be of this base type will be automatically initialized to 'U' unless expressly assigned a default expression.

Uninitialized values were designed to provide a means of detecting system values that have not changed from their uninitialized state since the time of system initialization. Hence, the logical tables for AND, OR, NAND, NOR, XOR, XNOR, and NOT have been designed to propagate 'U' states whenever encountered.

The propagation of 'U's through a circuit gives the designer an understanding of where the system has failed to be properly initialized. The AND gate example that follows illustrates the effect of 'U' propagation.
A.4 Behavioral modeling for 'U' propagation

For behavioral modeling where 'U' propagation is desired, the function To_UX01 will provide a reduction in the state system, as far as the modeler is concerned, thereby easing the modeler’s task.

A.5 'U's related to conditional expressions

Case statements, “if” expressions, and selected signal assignments need to separately treat 'U' states and provide a path for 'U' state propagation in order to propagate 'U's.

A.6 Structural modeling with logical tables

The logical tables are designed to generate output values in the range 'U', 'X', '0', and '1'. Therefore, once an element of the nine-state system passes through any of the logical tables, it will be converted to forcing strength. If the need arises for a weak or floating strength to be propagated through the remainder of a circuit or to an output port, then the model developer shall be certain to assign the appropriate value accordingly.

A.7 X-handling: assignment of X's

In assignments, the 'X' and '-' values differ minimally. The value '-', also known as “output don’t care,” explicitly means that synthesis tools are allowed to generate either a '0' or a '1', whichever leads to minimal circuitry, whereas 'X' usually appears during transitions or as a result of bus contentions or to flag model generated internal error conditions, such as in the following waveform assignment:

$$ S <= 'X' \text{ after 1 ns, } '1' \text{ after 5 ns} $$

where the current value of S becomes indeterminate after 1 ns and then reaches '1' after 5 ns have elapsed.

A.8 Modeling with don’t care’s

A.8.1 Use of the don’t care state in synthesis models

For synthesis, a VHDL program is a specification of the functionality of a design. VHDL can also be used to model (in order to simulate) real circuits. The former deals with logical function of the circuit, while the latter is concerned with function of a circuit from an electrical point of view. The nine-state logic type usage for synthesis is based on the assumption that the VHDL models will be logical function specifications and, therefore, attempts to restrict the usage of the logic type to logical function. The motivation for allowing the user to reference the values 'U' and 'X' (which do not specify the behavior of the circuit to be built, i.e., one can not build a circuit which “drives an 'X'”) is to allow such simulation artifacts to remain in models for synthesis for the sake of convenience. By having synthesis remove these references, the user is assuming only the kind of usage (of 'U' and 'X') that catches error states that should never occur in hardware.

A.8.2 Semantics of '-'

In designing the resolution function and the various logic tables in the package body, '-' is almost exclusively a syntactic shorthand for 'X', provided for compatibility with synthesis tools. This is evident from that fact that '-' becomes 'X' as soon as it is operated upon and when it is converted to subtype X01 or UX01. The “output don’t care” value represents either a '1' or a '0' as the output of combinatorial circuitry, with respect to state encoding in particular.
A.9 Resolution function

In digital logic design, there are a number of occasions in which driving outputs of more than one device are connected together. The most common of which is tri-state™ buses in which memory data ports are connected to each other and to controlling microprocessors. Another common case is one in which multiple drivers are parallel driving a heavily loaded signal path. In each of these cases, the VHDL language requires that the signals used to interconnect those devices be “resolved” signal types.

Focusing on resolution: when two signals’ values are driving the same “wire,” some resulting value will be observed on that wire. For example, if two parallel buffers both drive ‘1’ onto a signal, then the signal will be ‘1’. If a tri-state driver is in the high-impedance state ‘Z’ and another driver is in the forcing one ‘1’ state, then the combination of those two signal values will result in a value of ‘1’ appearing on the wire.

The resolution function built into Std_logic_1164 operates on the principal that weak values dominate over high-impedance values and forcing values dominate over weak values.

A.10 Using Std_ulogic vs. Std_logic

In deciding whether to use the resolved signal or unresolved signal type, a number of considerations need to be made:

a) Does the simulator run slower when using a resolved type than when using an unresolved type; or is the simulator optimized for the std_logic data types?

b) What should be done to insure interoperability of models with other model developers?

Each of these is considered, in order, below:

Most simulator vendors, in approving this standard, voiced their strong interest in having the package body reflect the formal semantics of the package, but wanted to be allowed to implement the package body in the most efficient manner. Consequently, a great number of simulator vendors will optimize their environments to maximize performance for signals declared of the resolved type.

In the case of two unity buffers, wired in parallel and driving a common signal, the driven signal shall be a resolved signal (i.e., std_logic) and the type of the unity driver output ports may be either std_ulogic or std_logic; either will work properly. But, suppose a user developed a model of an octal buffer and preferred to model the input and output ports as eight element arrays of std_logic just to benefit from the ease of wiring arrays rather than individually wiring each and every buffer element. In this scenario, the user must make a choice between std_ulogic_vector and std_logic_vector as the array type of the buffer port. Since std_logic_vector and std_ulogic_vector are TYPES, they are by definition incompatible. Therefore, if the user chooses incorrectly, he or she will not be able to wire this array to a microprocessor address or data bus unless that microprocessor model uses exactly the same data type for its ports. Since the user may have not developed the microprocessor model, he or she may not know what data type was used and might prefer not to use a type conversion function in order to have the two models interconnect. Therefore, the resolved vector type is preferred.

For scalar ports and signals, the developer may use either the std_ulogic or std_logic type.

For vector ports and signals, the developer should use the STD_LOGIC_VECTOR type.

1 Tri-state is a trademark of National Semiconductor.