BRITISH STANDARD

BS EN 61691-2:2001
IEC 61691-2:2001

Behavioural languages —

Part 2: VHDL multilogic system for model interoperability

The European Standard EN 61691-2:2001 has the status of a British Standard

ICS 35.240.50

National foreword

This British Standard is the official English language version of EN 61691-2:2001. It is identical with IEC 61691-2:2001.

The UK participation in its preparation was entrusted to Technical Committee GEL/93, Design automation, which has the responsibility to:

- aid enquirers to understand the text;
- present to the responsible international/European committee any enquiries on the interpretation, or proposals for change, and keep the UK interests informed;
- monitor related international and European developments and promulgate them in the UK.

A list of organizations represented on this committee can be obtained on request to its secretary.

From 1 January 1997, all IEC publications have the number 60000 added to the old number. For instance, IEC 27-1 has been renumbered as IEC 60027-1. For a period of time during the change over from one numbering system to the other, publications may contain identifiers from both systems.

Cross-references

The British Standards which implement international or European publications referred to in this document may be found in the BSI Standards Catalogue under the section entitled "International Standards Correspondence Index", or by using the "Find" facility of the BSI Standards Electronic Catalogue.

A British Standard does not purport to include all the necessary provisions of a contract. Users of British Standards are responsible for their correct application.

Compliance with a British Standard does not of itself confer immunity from legal obligations.

This British Standard, having been prepared under the direction of the Electrotechnical Sector Policy and Strategy Committee, was published under the authority of the Standards Policy and Strategy Committee on 2 April 2002

Summary of pages

This document comprises a front cover, an inside front cover, the EN title page, the EN foreword page, the IEC title page, pages 2 to 23 and a back cover.

The BSI copyright date displayed in this document indicates when the document was last issued.

Amendments issued since publication

Amd. No.	Date	Comments

ISBN 0 580 39266 X

EUROPEAN STANDARD **EN 61691-2**

NORME EUROPÉENNE

EUROPÄISCHE NORM December 2001

ICS 35.240.50

English version

Behavioural languages
Part 2: VHDL multilogic system for model interoperability
(IEC 61691-2:2001)

Langages relatifs au comportement
Partie 2: Système multilogique en VHDL permettant l'interopérabilité des modèles
(CEI 61691-2:2001)

Verhaltensebenensprache
Teil 2: System für mehrwertige Logik für das VHDL-Interoperabilitätsmodell
(IEC 61691-2:2001)

This European Standard was approved by CENELEC on 2001-09-01. CENELEC members are bound to comply with the CEN/CENELEC Internal Regulations which stipulate the conditions for giving this European Standard the status of a national standard without any alteration.

Up-to-date lists and bibliographical references concerning such national standards may be obtained on application to the Central Secretariat or to any CENELEC member.

This European Standard exists in three official versions (English, French, German). A version in any other language made by translation under the responsibility of a CENELEC member into its own language and notified to the Central Secretariat has the same status as the official versions.

CENELEC members are the national electrotechnical committees of Austria, Belgium, Czech Republic, Denmark, Finland, France, Germany, Greece, Iceland, Ireland, Italy, Luxembourg, Malta, Netherlands, Norway, Portugal, Spain, Sweden, Switzerland and United Kingdom.

CENELEC
European Committee for Electrotechnical Standardization
Comité Européen de Normalisation Electrotechnique
Europäisches Komitee für Elektrotechnische Normung

Central Secretariat: rue de Stassart 35, B - 1050 Brussels

Ref. No. EN 61691-2:2001 E

Foreword

The text of document 93/130/FDIS, future edition 1 of IEC 61691-2, prepared by IEC TC 93, Design automation, was submitted to the IEC-CENELEC parallel vote and was approved by CENELEC as EN 61691-2 on 2001-09-01.

The following dates were fixed:

– latest date by which the EN has to be implemented at national level by publication of an identical national standard or by endorsement	(dop)	2002-06-01
– latest date by which the national standards conflicting with the EN have to be withdrawn	(dow)	2004-09-01

This standard is based on IEEE Std 1164:1993, Multivalue logic system for VHDL model interoperability.

Endorsement notice

The text of the International Standard IEC 61691-2:2001 was approved by CENELEC as a European Standard without any modification.

EN 61691–2:2001

INTERNATIONAL STANDARD

IEC 61691-2

First edition
2001-06

Behavioural languages –

Part 2: VHDL multilogic system for model interoperability

Reference number
IEC 61691-2:2001(E)

BEHAVIOURAL LANGUAGES - Part 2: VHDL multilogic system for model interoperatibility

1. Overview

1.1 Scope

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

1.2 Conformance with this standard

The following conformance rules shall apply as they

a) No modifications shall be made to the package declaration
b) The Std_logic_1164 package body represents the formal Std_logic_1164 package declaration. Implementers of this package body as it is; or they may choose to implement to the user. Users shall not implement a semantic that

2. Std_logic_1164 package declaration

```
--
-- 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 packages defines a standard for designers
--           : to use in describing the
--           : used in VHDL modeling.
--           :
```

```
--   Limitation:  The logic system defined in this package may
--            :  be insufficient for modeling switched
--            :  since such a requirement is out of the
--            :  effort. Furthermore, mathematics, primitives,
--            :  timing standards, etc. are considered
--            :  issues in relation to this package and
--            :  beyond the scope of this effort.
--            :
--   Note     :  No declarations or definitions shall be
--            :  or excluded from, this package. The
--            :  defines the types, subtypes, and
--            :  Std_logic_1164. The Std_logic_1164
--            :  considered the formal definition of the
--            :  this package. Tool developers may
--            :  the package body in the most efficient
--            :  to them.
--            :
--
--   modification history :
--
-- version | mod. date:|
--  v4.200 | 01/02/92  |
--
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

  TYPE std_ulogic_vector IS ARRAY ( NATURAL RANGE <> )

  -- 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

  TYPE std_logic_vector IS ARRAY ) NATURAL RANGE <>) OF
```

```
    -- common subtypes

    SUBTYPE X01     IS resolved std_ulogic RANGE ‘
    SUBTYPE X01Z    IS resolved std_ulogic RANGE “‘Z’)
    SUBTYPE UX01    IS resolved std_ulogic RANGE “‘1’)
    SUBTYPE UX01Z   IS resolved std_ulogic RANGE “‘1’, ‘Z’)

    -- overloaded logical operators

    FUNCTION “and”   ( l : std_ulogic; r :
    FUNCTION “nand”  ( l : std_ulogic; r :
    FUNCTION “or”    ( l : std_ulogic; r :
    FUNCTION “nor”   ( l : std_ulogic; r :
    FUNCTION “xor”   ( l : std_ulogic; r :
    FUNCTION “xnor”  ( l : std_ulogic; r :
    FUNCTION “not”   ( l : std_ulogic

    -- vectorized overloaded logical operators

    FUNCTION “and”   ( l, r : std_logic_vector  )
    FUNCTION “and”   ( l, r : std_ulogic_vector )
    FUNCTION “nand”  ( l, r : std_logic_vector  )
    FUNCTION “nand”  ( l, r : std_ulogic_vector )
    FUNCTION “or”    ( l, r : std_logic_vector  )
    FUNCTION “or”    ( l, r : std_ulogic_vector )
    FUNCTION “nor”   ( l, r : std_logic_vector  )
    FUNCTION “nor”   ( l, r : std_ulogic_vector )
    FUNCTION “xor”   ( l, r : std_logic_vector  )
    FUNCTION “xor”   ( l, r : std_ulogic_vector )
--
-- Note : The declaration and implementation of the “
-- specifically commented until a time at which the VHDL
-- officially adopted as containing such a function. At
-- the following comments may be removed along with this
-- further “official” balloting of this
-- the intent of this effort to provide such a function
-- available in the VHDL standard.
--
-- FUNCTION “xnor” ( l, r : std_logic_vector  )
-- FUNCTION “xnor” ( l, r : std_ulogic_vector )
    FUNCTION “not”  ( l : std_logic_vector  )
    FUNCTION “not”  ( l : std_ulogic_vector )

    -- conversion functions

    FUNCTION To_bit       ( s : std_ulogic;        xmap :
    FUNCTION To_bitvector ( s : std_logic_vector ; xmap : BIT_VECTOR;
    FUNCTION To_bitvector ( s : std_ulogic_vector; xmap : BIT_VECTOR;
    FUNCTION To_StdULogic       ( b : BIT              )
    FUNCTION To_StdLogicVector  ( b : BIT_VECTOR       )
    FUNCTION To_StdLogicVector  ( s : std_ulogic_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
  FUNCTION To_X01  ( s : std_ulogic_vector ) RETURN
  FUNCTION To_X01  ( s : std_ulogic        ) RETURN X01;
  FUNCTION To_X01  ( b : BIT_VECTOR        ) RETURN
  FUNCTION To_X01  ( b : BIT_VECTOR        ) RETURN
  FUNCTION To_X01  ( b : BIT               ) RETURN X01;
  FUNCTION To_X01Z ( s : std_logic_vector  ) RETURN
  FUNCTION To_X01Z ( s : std_ulogic_vector ) RETURN
  FUNCTION To_X01Z ( s : std_ulogic        ) RETURN X01Z;
  FUNCTION To_X01Z ( b : BIT_VECTOR        ) RETURN
  FUNCTION To_X01Z ( b : BIT_VECTOR        ) RETURN
  FUNCTION To_X01Z ( b : BIT               ) RETURN X01Z;
  FUNCTION To_UX01 ( s : std_logic_vector  ) RETURN
  FUNCTION To_UX01 ( s : std_ulogic_vector ) RETURN
  FUNCTION To_UX01 ( s : std_ulogic        ) RETURN UX01;
  FUNCTION To_UX01 ( b : BIT_VECTOR        ) RETURN
  FUNCTION To_UX01 ( b : BIT_VECTOR        ) RETURN
  FUNCTION To_UX01 ( b : BIT               ) 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
--            :  to use in describing the interconnection
--            :  used in VHDL modeling.
--            :
--  Limitation:  The logic system defined in this package may
--            :  be insufficient for modeling switched
--            :  since such a requirement is out of the
--            :  effort. Furthermore, mathematics, primitives,
--            :  timing standards, etc., are considered
--            :  issues in relation to this package and
```

```
--          :  beyond the scope of this effort.
--          :
--   Note     :  No declarations or definitions shall be
--          :  or excluded from this package. The "
--          :  defines the types, subtypes and declarations of
--          :  Std_logic_1164. The Std_logic_1164
--          :  considered the formal definition of the
--          :  this package. Tool developers may choose
--          :  the package body in the most efficient
--          :  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)

   -- resolution function

   CONSTANT resolution_table : stdlogic_table := (
   --
   --      | U    X    0    1    Z    W    L    H    -
   --
         ( 'U', 'U', 'U', '
         ( 'U', 'X', 'X', '
         ( 'U', 'X', '0', '
         ( 'U', 'X', 'X', '
         ( 'U', 'X', '0', '
         ( 'U', 'X', '0', '
         ( 'U', 'X', '0', '
         ( 'U', 'X', '0', '
         ( 'U', 'X', 'X', '
      );

   FUNCTION resolved ( s : std_ulogic_vector ) RETURN
      VARIABLE result : std_ulogic := 'Z'; --
   BEGIN
      -- the test for a single driver is essential;
      -- loop would return 'X' for a single
      -- would conflict with the value of a single
      -- 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   X   0   1   Z   W   L   H   -
--     --------------------------------------------------
       ( 'U', 'U', '0', '
       ( 'U', 'X', '0', '
       ( '0', '0', '0', '
       ( 'U', 'X', '0', '
       ( 'U', 'X', '0', '
       ( 'U', 'X', '0', '
       ( '0', '0', '0', '
       ( 'U', 'X', '0', '
       ( 'U', 'X', '0', '
);
-- truth table for "or" function
CONSTANT or_table : stdlogic_table := (
--     --------------------------------------------------
--     | U   X   0   1   Z   W   L   H   -
--     --------------------------------------------------
       ( 'U', 'U', 'U', '
       ( 'U', 'X', 'X', '
       ( 'U', 'X', '0', '
       ( '1', '1', '1', '
       ( 'U', 'X', 'X', '
       ( 'U', 'X', 'X', '
       ( 'U', 'X', '0', '
       ( '1'  '1', '1', '
       ( 'U', 'X', 'X', '
);
-- truth table for "xor" function
CONSTANT xor_table : stdlogic_table := (
--     --------------------------------------------------
--     | U   X   0   1   Z   W   L   H   -
--     --------------------------------------------------
       ( 'U', 'U', 'U', '
       ( 'U', 'X', 'X', '
       ( 'U', 'X', '0', '
       ( 'U', 'X', '1', '
       ( 'U', 'X', 'X', '
       ( 'U', 'X', 'X', '
       ( 'U', 'X', '0', '
       ( 'U', 'X', '1', '
       ( 'U', 'X', 'X', '
);
-- truth table for "not" function
CONSTANT not_table: stdlogic_1d :=
--  ------------------------------------------------
-- |  U   X   0   1   Z   W   L   H   -  |
```

```
-- -------------------------------------------------
( 'U', 'X', '1', '0',

-- overloaded logical operators ( with optimizing hints )

FUNCTION "and"  ( l : std_ulogic; r :
BEGIN
   RETURN (and_table(l, r));
END "and";
FUNCTION "nand" ( l : std_ulogic; r :
BEGIN
   RETURN (not_table ( and_table(l, r)));
END "nand";
FUNCTION "or"   ( l : std_ulogic; r :
BEGIN
   RETURN (or_table(l, r));
END "or";
FUNCTION "nor"  ( l : std_ulogic; r :
BEGIN
   RETURN (not_table ( or_table( l, r )));
END "nor";
FUNCTION "xor"  ( l : std_ulogic; r :
BEGIN
   RETURN (xor_table(l, r));
END "xor";
--  FUNCTION "xnor"  ( l : std_ulogic; r :
--  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";

-- and

FUNCTION "and"  ( l,r : std_logic_vector )
   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 'length"
      SEVERITY FAILURE;
   ELSE
      FOR i IN result'RANGE LOOP
         result(i) := and_table (lv(i), rv(i));
      END LOOP;
   END IF;
   RETURN result;
END "and";

FUNCTION "and"  ( l,r : std_ulogic_vector )
```

```
   ALIAS lv : std_ulogic_vector ( 1 To l'LENGTH ) IS l;
   ALIAS rv : std_ulogic_vector ( 1 TO r'LENGTH ) IS r;
   VARIABLE result : std_ulogic_vector ( 1 TO l'LENGTH );
BEGIN
   IF ( l'LENGTH /= r'LENGTH ) THEN
      ASSERT FALSE
      REPORT "arguments of overloaded 'length"
      SEVERITY FAILURE;
   ELSE
      FOR i IN result'RANGE LOOP
         result(i) := and_table (lv(i), rv(i));
      END LOOP;
   END IF;
   RETURN result;
END "and";

-- nand

FUNCTION "nand"  ( l,r : std_logic_vector )
   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 'length"
      SEVERITY FAILURE;
   ELSE
      FOR i IN result'RANGE LOOP
         result(i) := not_table(and_table (lv(i), rv(i)));
      END LOOP;
   END IF;
   RETURN result;
END "nand";

FUNCTION "nand"  ( l,r : std_ulogic_vector )
   ALIAS lv : std_ulogic_vector ( 1 TO l'LENGTH ) IS l;
   ALIAS rv : std_ulogic_vector ( 1 TO r'LENGTH ) IS r;
   VARIABLE result : std_ulogic_vector ( 1 TO l'LENGTH );
BEGIN
   IF ( l'LENGTH /= r'LENGTH ) THEN
      ASSERT FALSE
      REPORT "arguments of overloaded 'length"
      SEVERITY FAILURE;
   ELSE
      FOR i IN result'RANGE LOOP
         result(i) := not_table(and_table (lv(i), rv(i)));
      END LOOP;
   END IF;
   RETURN result;
END "nand";

-- or
```

```
FUNCTION “or”  ( l,r : std_logic_vector )
   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 ‘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 )
   ALIAS lv : std_ulogic_vector ( 1 TO l'LENGTH ) IS l;
   ALIAS rv : std_ulogic_vector ( 1 TO r'LENGTH ) IS r;
   VARIABLE result : std_ulogic_vector ( 1 TO l'LENGTH );
BEGIN
   IF ( l'LENGTH /= r'LENGTH ) THEN
      ASSERT FALSE
      REPORT “arguments of overloaded ‘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’;

-- nor

FUNCTION “nor”  ( l,r : std_logic_vector )
   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 ‘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_ulogic_vector )
```

```
    ALIAS lv : std_ulogic_vector ( 1 TO l'LENGTH ) IS l;
    ALIAS rv : std_ulogic_vector ( 1 TO r'LENGTH ) IS r;
    VARIABLE result : std_ulogic_vector ( 1 TO l'LENGTH );
  BEGIN
    IF ( l'LENGTH /= r'LENGTH ) THEN
      ASSERT FALSE
      REPORT "arguments of overloaded '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";

  -- xor

  FUNCTION "xor"  ( l,r : std_logic_vector )
    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 '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 )
    ALIAS lv : std_ulogic_vector ( 1 TO l'LENGTH ) IS l;
    ALIAS rv : std_ulogic_vector ( 1 TO r'LENGTH ) IS r;
    VARIABLE result : std_ulogic_vector ( 1 TO l'LENGTH );
  BEGIN
    IF ( l'LENGTH /= r'LENGTH ) THEN
      ASSERT FALSE
      REPORT "arguments of overloaded '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";
--
--  -- xnor
--
```

```
--  Note : The declaration and implementation of the “
--  specifically commented until a time at which the VHDL
--  officially adopted as containing such a function. At
--  the following comments may be removed along with this
--  further “official” balloting of this
--  the intent of this effort to provide such a function
--  available in the VHDL standard.
--
--  FUNCTION “xnor” ( l, r : std_logic_vector )
--      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 ‘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 )
--      alias lv : std_ulogic_vector ) 1 to l'length ) is l;
--      alias rv : std_ulogic_vector ) 1 to r'length ) is r;
--      variable result : std_ulogic_vector ( 1 to l'length );
--  begin
--      if ( l'length /= r'length ) then
--         assert false
--         report “arguments of overloaded ‘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”;

    -- not

    FUNCTION “not”  ( l : std_logic_vector )
       ALIAS lv : std_logic_vector ( 1 TO l'LENGTH ) IS l;
       VARIABLE result : std_logic_vector ( 1 To
    BEGIN
       FOR i IN result'RANGE LOOP
          result(i) := not_table( lv(i) );
       END LOOP;
       RETURN result;
    END;
```

```
FUNCTION "not" ( l : std_ulogic_vector )
   ALIAS 1v : std_ulogic_vector ) 1 TO l'LENGTH ) IS l;
   VARIABLE result : std_ulogic_vector ( 1 TO
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
TYPE logic_x01z_table IS ARRAY (std_ulogic'LOW TO
TYPE logic_ux01_table IS ARRAY (std_ulogic'LOW TO
------------------------------------------------------
-- 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) = '
--
------------------------------------------------------
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) = '
--
------------------------------------------------------
CONSTANT cvt_to_x01z : logic_x01z_table := (
                'X',  -- 'U'
                'X',  -- 'X'
                '0',  -- '0'
                '1',  -- '1'
```

```
                    'Z',  -- 'Z'
                    'X',  -- '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) = '

 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
 BEGIN
        CASE s IS
             WHEN '0' | 'L' =>
             WHEN '1' | 'H' =>
             WHEN OTHERS => RETURN xmap;
        END CASE;
 END;

 FUNCTION To_bitvector ( s : std_logic_vector ; xmap : BIT_VECTOR_IS
    ALIAS sv : std_logic_vector ( s'LENGTH-1 DOWNTO
    VARIABLE result : BIT_VECTOR (s'LENGTH-1 DOWNTO 0 );
 BEGIN
    FOR i IN result'RANGE LOOP
        CASE sv(i) IS
           WHEN '0' | 'L' =>
           WHEN '1' | 'H' =>
           WHEN OTHERS => result(i) := xmap;
        END CASE;
    END LOOP;
    RETURN result;
 END;

 FUNCTION To_bitvector ( s : std_ulogic_vector; xmap : BIT_VECTOR_IS
    ALIAS sv : std_logic_vector ( s'LENGTH-1 DOWNTO
```

```
   VARIABLE result : BIT_VECTOR (s'LENGTH-1 DOWNTO 0 );
BEGIN
   FOR i IN result'RANGE LOOP
      CASE sv(i) IS
         WHEN '0' | 'L' =>
         WHEN '1' | 'H' =>
         WHEN OTHERS => result(i) := xmap;
      END CASE;
   END LOOP;
   RETURN result;
END;

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

FUNCTION To_StdlogicVector ( b : BIT_VECTOR  ) RETURN
   ALIAS bv : BIT_VECTOR (b'LENGTH-1 DOWNTO 0 ) IS b;
   VARIABLE result : std_logic_vector (b'LENGTH-1
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
   VARIABLE result : std_logic_vector ( s'LENGTH-1
BEGIN
   FOR i IN RESULT'RANGE LOOP
      result(i) := sv(i)
   END LOOP;
   RETURN result;
END;

FUNCTION To_StdULogicVector ( b : BIT_VECTOR       ) IS
   ALIAS bv : BIT_VECTOR ( b'LENGTH-1 DOWNTO 0 ) IS b;
   VARIABLE result : std_ulogic_vector ( b'LENGTH-1
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
   VARIABLE result : std_ulogic_vector ( s'LENGTH-1
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
   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
   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_ulogic ) RETURN X01 IS
BEGIN
   RETURN (cvt_to_x01(s));
END;

FUNCTION To_X01  ( b : BIT_VECTOR ) RETURN
   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
  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
  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  ( s : std_ulogic_vector ) RETURN
  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  ( s : std_ulogic ) RETURN X01Z IS
BEGIN
  RETURN (cvt_to_x01z(s));
END;

FUNCTION To_X01Z  ( b : BIT_VECTOR ) RETURN
  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  ( b : BIT_VECTOR ) RETURN
    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
    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  ( s : std_ulogic_vector ) RETURN
    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
   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_VECTOR ) RETURN
    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_UX01  ( b : BIT ) RETURN UX01 IS
BEGIN
   CASE b IS
      WHEN '0' => RETURN('0');
      WHEN '1' => RETURN('l');
   END CASE;
END;

-- edge detection

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

-- 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' | '
```

```
            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' | '
            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'
         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 a means of building models that interoperate, provided typing imposed by the VHDL language.

A.1 Value system

The value system in Std_logic_1164 was developed to model the logic system is named "std_ulogic" where the comprising the type have a specified semantic and a interoperate, one must interpret the meaning of each of

```
Type std_ulogic is (
        'U',        Uninitialized state
        'X',        Forcing Unknown etc.
        '0',        Forcing Zero
        '1',        Forcing One
        'Z',        High Impedance
        'W',        Weak Unknown
        'L',        Weak Zero
        'H',        Weak One
        '-'         Don't Care modeling
);
```

A.2 Handling strengths

Behavioral modeling techniques rarely require knowledge "strength stripper" functions have been designed "forcing" strength counterparts.

Once in forcing strength, the model can simply respond to stripping is done by using one of the following functions:

```
To_X01 (...)      converts 'L' and 'H' to '0' and
To_UX0 1 (...)    converts 'L' and 'H' to '0' and to 'X'.
```

A.3 Use of the uninitialized value

The 'U' value is located in the first position of automatically initialized to 'U' unless expressly

Uninitialized values were designed to provide a means of uninitialized state since the time of system XNOR, and NOT have been designed to propagate 'U'

The propagation of 'U's through a circuit gives properly initialized. The AND gate example that follows

A.4 Behavioral modeling for 'U' propagation

For behavioral modeling where 'U' propagation is system, as far as the modeler is concerned, thereby

A.5 'U's related to conditional expressions

Case statements, "if" expressions, and selected path for 'U' state propagation in order to

A.6 Structural modeling with logical tables

The logical tables are designed to generate output values of the nine-state system passes through any of the arises for a weak or floating strength to be propagated model developer shall be certain to assign the

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

In assignments, the ‘X’ and ‘-’ values means that synthesis tools are allowed to generate either ‘X’ usually appears during transitions or as a conditions, such as in the following waveform assignment:

S <= ‘X’ after 1 ns, ‘1’ after 5 ns

where the current value of S becomes indeterminate after

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 order to simulate) real circuits. The former deals with function of a circuit from an electrical point of view. assumption that the VHDL models will be logical function of the logic type to logical function. The motivation for do not specify the behavior of the circuit to be built, such simulation artifacts to remain in models for these references, the user is assuming only the kind of occur in hardware.

A.8.2 Semantics of ‘-’

In designing the resolution function and the various syntactic shorthand for ‘X’, provided for becomes ‘X’ as soon as it is operated upon and value represents either a ‘1’ or a ‘0’ as

A.9 Resolution function

In digital logic design, there are a number of occasions together. The most common of which is tri-state™[1] buses in which memory data ports are connected to each to controlling microprocessors. Another common case is loaded signal path. In each of these cases, the VHDL devices be “resolved” signal types.

Focusing on resolution: when two signals' values are that wire. For example, if two parallel buffers both is in the high-impedance state ‘Z’ and another signal values will result in a value of ‘1’

The resolution function built into Std_logic_1164 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

a) Does the simulator run slower when using a resolved type simulator optimized for the std_logic data types?
b) What should be done to insure interoperability of models

Each of these is considered, in order, below:

[1]Tri-state is a trademark of National Semiconductor.

Most simulator vendors, in approving this standard, formal semantics of the package, but wanted to be allowed manner. Consequently, a great number of simulator vendors performance for signals declared of the resolved type.

In the case of two unity buffers, wired in parallel and signal (i.e., std_logic) and the type of the unity driver work properly. But, suppose a user developed a model of ports as eight element arrays of std_logic just to each and every buffer element. In this scenario, the user std_logic_vector as the array type of the buffer port. are by definition incompatible. Therefore, if the user to a microprocessor address or data bus unless that Since the user may have not developed the microprocessor and might prefer not to use a type conversion function in order to have 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.

BSI — British Standards Institution

BSI is the independent national body responsible for preparing British Standards. It presents the UK view on standards in Europe and at the international level. It is incorporated by Royal Charter.

Revisions

British Standards are updated by amendment or revision. Users of British Standards should make sure that they possess the latest amendments or editions.

It is the constant aim of BSI to improve the quality of our products and services. We would be grateful if anyone finding an inaccuracy or ambiguity while using this British Standard would inform the Secretary of the technical committee responsible, the identity of which can be found on the inside front cover. Tel: +44 (0)20 8996 9000. Fax: +44 (0)20 8996 7400.

BSI offers members an individual updating service called PLUS which ensures that subscribers automatically receive the latest editions of standards.

Buying standards

Orders for all BSI, international and foreign standards publications should be addressed to Customer Services. Tel: +44 (0)20 8996 9001. Fax: +44 (0)20 8996 7001. Email: orders@bsi-global.com. Standards are also available from the BSI website at http://www.bsi-global.com.

In response to orders for international standards, it is BSI policy to supply the BSI implementation of those that have been published as British Standards, unless otherwise requested.

Information on standards

BSI provides a wide range of information on national, European and international standards through its Library and its Technical Help to Exporters Service. Various BSI electronic information services are also available which give details on all its products and services. Contact the Information Centre. Tel: +44 (0)20 8996 7111. Fax: +44 (0)20 8996 7048. Email: info@bsi-global.com.

Subscribing members of BSI are kept up to date with standards developments and receive substantial discounts on the purchase price of standards. For details of these and other benefits contact Membership Administration. Tel: +44 (0)20 8996 7002. Fax: +44 (0)20 8996 7001. Email: membership@bsi-global.com.

Information regarding online access to British Standards via British Standards Online can be found at http://www.bsi-global.com/bsonline.

Further information about BSI is available on the BSI website at http://www.bsi-global.com.

Copyright

BSI
389 Chiswick High Road
London
W4 4AL