SLOVENSKI STANDARD

SIST EN 61691-3-3:2007

januar 2007

Vedenjski jeziki - 3-3. del: Sinteza v VHDL (IEC 61691-3-3:2001) (istoveten EN 61691-3-3:2001)

Behavioural languages - Part 3-3: Synthesis in VHDL (IEC 61691-3-3:2001)

ICS 35.060

Referenčna številka
SIST EN 61691-3-3:2007(en)

EUROPEAN STANDARD

NORME EUROPÉENNE

EUROPÄISCHE NORM

EN 61691-3-3

December 2001

ICS 35.240.50

English version

Behavioural languages
Part 3-3: Synthesis in VHDL
(IEC 61691-3-3:2001)

Langages relatifs au comportement
Partie 3-3: Synthèse en VHDL de la norme IEEE - Progiciels
(CEI 61691-3-3:2001)

Verhaltensebenensprache
Teil 3-3: Synthese mit VHDL
(IEC 61691-3-3:2001)

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 61691-3-3:2001 E

Foreword

The text of document 93/132/FDIS, future edition 1 of IEC 61691-3-3, prepared by IEC TC 93, Design automation, was submitted to the IEC-CENELEC parallel vote and was approved by CENELEC as EN 61691-3-3 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– latest date by which the national standards conflicting with the EN have to be withdrawn	(dow) 2004-09-01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 1076-3:1997, Synthesis packages.

Endorsement notice

The text of the International Standard IEC 61691-3-3:2001 was approved by CENELEC as a European Standard without any modification.

INTERNATIONAL STANDARD

IEC 61691-3-3

First edition
2001-06

Behavioural languages –

Part 3-3: Synthesis in VHDL

International Electrotechnical Commission 3, rue de Varembé Geneva, Switzerland
Telefax: +41 22 919 0300 e-mail: inmail@iec.ch IEC web site http://www.iec.ch

Commission Electrotechnique Internationale
International Electrotechnical Commission
Международная Электротехническая Комиссия

PRICE CODE X

For price, see current catalogue

INTERNATIONAL ELECTROTECHNICAL COMMISSION

BEHAVIOURAL LANGUAGES –

Part 3-3: Synthesis in VHDL

FOREWORD

1) The IEC (International Electrotechnical Commission) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of the IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, the IEC publishes International Standards. Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. The IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of the IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested National Committees.
3) The documents produced have the form of recommendations for international use and are published in the form of standards, technical specifications, technical reports or guides and they are accepted by the National Committees in that sense.
4) In order to promote international unification, IEC National Committees undertake to apply IEC International Standards transparently to the maximum extent possible in their national and regional standards. Any divergence between the IEC Standard and the corresponding national or regional standard shall be clearly indicated in the latter.
5) The IEC provides no marking procedure to indicate its approval and cannot be rendered responsible for any equipment declared to be in conformity with one of its standards.
6) Attention is drawn to the possibility that some of the elements of this International Standard may be the subject of patent rights. The IEC shall not be held responsible for identifying any or all such patent rights.

International Standard IEC 61691-2-3 has been prepared by IEC technical committee 93: Design automation.

This standard is based on IEEE Std 1076-3 (1997: *Synthesis packages*

The text of this standard is based on the following documents:

FDIS	Report on voting
93/132/FDIS	93/142/RVD

Full information on the voting for the approval of this standard can be found in the report on voting indicated in the above table.

This standard does not follow the rules for the structure of international standards given in Part 3 of the ISO/IEC Directives.

IEC 61691 consists of the following parts, under the general title: *Behavioural languages:*

IEC 61691-1:1997, VHDL language reference manual [1)]

IEC 61691-2:2001, Part 2: VHDL multilogic system for model interoperability

---

[1)] The edition 2 with the title: VHSIC hardware description language VHDL (1076a) (under consideration) will replace it.

IEC 61691-3-1, Part 3-1: Analog description in VHDL (under consideration)

IEC 61691-3-2:2001, Part 3-2: Mathematical operation in VHDL

IEC 61691-3-3:2001, Part 3-3: Synthesis in VHDL

IEC 61691-3-4, Part 3-4: Timing expressions in VHDL (under consideration)

IEC 61691-3-5, Part 3-5: Library utilities in VHDL (under consideration)

The committee has decided that the contents of this publication will remain unchanged until 2004. At this date, the publication will be

- reconfirmed;
- withdrawn;
- replaced by a revised edition, or
- amended.

INTRODUCTION

This standard, 61691-3-3, 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.

This standard includes package bodies, as described in annex A, which are available in electronic format either on a diskette affixed to the back cover, or as a downloadable file from the IEC Web Store.

BEHAVIOURAL LANGUAGES - Part 3-3: Synthesis in VHDL

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 interpretation 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 manner 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
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