SLOVENSKI STANDARD

SIST-TP CLC/R217-020:2004

september 2004

Specifikacijski jeziki elektronskega sistema - VDHL smernice oblikovanja

Electronic system specification languages - VHDL modelling guidelines

ICS 35.060; 35.240.50

Referenčna številka
SIST-TP CLC/R217-020:2004(en)

CENELEC

R217-020

REPORT

November 2001

English version

Electronic system specification languages - VHDL modelling guidelines

This CENELEC Report has been prepared by TC 217, Electronic Design Automation (EDA). It was approved by TC 217 on 1997-07-07 and endorsed by the CENELEC Technical Board on 2001-03-06.

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. R217-020:2001 E

Foreword

This CENELEC Report has been prepared by the Technical Committee CENELEC TC 217, Electronic Design Automation (EDA). Having been approved by TC 217 in July 1997, the need for its publication was confirmed on 2000-10-16 and endorsed by the CENELEC Technical Board on 2001-03-06. The report is derived from the European Space Agency's (ESA's) *VHDL Modelling Guidelines,* reference ASIC/001 issue 1, dated September 1994.

The ESA *VHDL Modelling Guidelines* have been used in ESA development and study contracts to ensure high-quality maintainable VHDL models. They have been prepared by Peter Sinander with support from Sandi Habinc, both at the ESA/ESTEC Microelectronics and Technology Section (WSM), P.O. Box 299, 2200 AG Noordwijk, The Netherlands. The ESA *VHDL Modelling Guidelines* are based on the experience gained from ESA contracts in this area during several years, feedback from the contracting companies working for ESA, and various information found in articles, non-VHDL coding guidelines and on the Internet.

Table of contents

1 Introduction 4
1.1 Purpose and scope 4
1.2 Applicable documents 4
1.3 Reference documents 4

2 Requirements for all kinds of models 4
2.1 General 4
2.2 Names 5
2.3 Comments 6
2.4 Types 6
2.5 Files 7
2.6 Signals and ports 7
2.7 Assertions and reporting 8
2.8 Subprograms, processes, entities, architectures, component declarations 8
2.9 Configurations 9
2.10 Packages 9
2.11 Design libraries 10
2.12 Constructs to be avoided 10
2.13 Verification 11
2.14 File organisation 12

3 Additional requirements 13
3.1 Models for Component simulation 13
3.1.1 Names 13
3.1.2 Types 14
3.1.3 Model interface 14
3.2 Models for Board-level simulation 14
3.2.1 Names 15
3.2.2 Model interface 15
3.2.3 Handling of unknown values 15
3.2.4 Timing 16
3.2.5 Reporting 17
3.2.6 Verification 17
3.3 Models for System-level simulation 17
3.3.1 Model interface 18
3.3.2 Verification 18
3.4 Testbenches 18
3.4.1 Automated verification 18

Annex A: Abbreviations 20
Annex B: Compatibility between VHDL-87 and VHDL-93 21
Annex C: Calculation of VHDL line coverage 22
Annex D: VHDL code examples 23
Annex E: Selection of simulation condition 47

1 Introduction

1.1 Purpose and scope

This document defines requirements on VHDL models and testbenches. It concerns simulation and documentation aspects of VHDL models; specific aspects for logic synthesis from VHDL have not been included. Nevertheless, the requirements of this document are compatible with the use of logic synthesis. The document is focused on digital models; specific requirements for analog modelling have not been covered. The requirements are not applicable for the case when a design database is transferred in VHDL format as a netlist. The requirements are targeted to the finalised models rather than the models during the development.

The purpose of these requirements is to ensure a high quality of the developed VHDL models, so they can be efficiently used and maintained with a low effort throughout the full life-cycle of the modelled hardware.

The requirements are based on the VHDL-93 standard, to minimise future maintenance efforts for updating models. However, it is recommended to keep the models backward compatible with VHDL-87 as far as possible, since some tools have not been updated.

The requirements have been structured in a general part applicable to all VHDL models and additional requirements applicable to different kinds of models: Component Simulation, Board-level simulation and System-level simulation. In addition, VHDL code examples have been included to provide some guidance to the VHDL developer.

1.2 Applicable documents

AD1 IEEE Standard VHDL Language Reference Manual, IEEE Std 1076-1993, 6 June 1994
AD2 IEEE Standard Multivalue Logic System for VHDL Model Interoperability (std_logic_1164), IEEE Std 1164-1993, 26 May 1993

1.3 Reference documents

RD1 IEEE Standard VHDL Language Reference Manual, IEEE Std 1076-1987, 31 March 1988
RD2 IEEE Standards Interpretations: IEEE Standard VHDL Language Reference Manual, IEEE Std 1076/INT-1991 or latest version
RD3 Standard VITAL ASIC Modeling Specification, IEEE Std 1076.4-1995
RD4 Standard VHDL Mathematical Packages, IEEE Std P 1076.2
RD5 Standard VHDL Synthesis Packages, IEEE Std P 1076.3

2 Requirements for all kinds of models

2.1 General

The models shall be written in VHDL-93 as defined in AD1. All code shall be written with the intent to be simulator independent (as far as possible, using all available information); the use of non-standard constructs or supersets is not allowed. Note that the code is not necessarily correct just because it compiles and executes on one simulator without errors; many tools do not detect all possible errors. In case of ambiguities, the interpretations in RD2 shall have precedence.

The models shall be coded so run-time errors can never occur due to model itself, such as division by zero, range error etc.

All models shall be compliant with VHDL-93 as defined in AD1. To allow backward compatibility with VHDL-87, the VHDL code shall as far as possible also be compliant with RD1. The benefits of using the new features of VHDL-93 should be assessed before being employed.

All documentation, identifiers, comments, messages, file names etc. should use or be based on the English language. Exceptionally, models only intended for local or national usage for the foreseeable fixture, might use the local language when beneficial.

The code shall be consistent in writing style and naming conventions. The VHDL reserved words shall appear in uniform casing; they shall either all appear in lower-case, or all appear in upper-case. It is recommended to write identifiers using mixed casing. The same consistent casing shall be used in all the code of a model, with the exception of standard packages.

The code shall be concise and use the most straightforward and intuitive constructs. Using more code than necessary leads to poorer readability and lower simulation speed. Wherever possible, unused parts of the code shall be removed. Temporary assignments should not be used unless necessary.

The code shall emphasise good readability. It shall contain maximum one statement per line, and have maximum 80 characters per line. The code shall be properly indented, for example using 3 space characters; the indentation shall be the same in all the code. The TAB character shall not be used, being environment dependent. Related constructs should be grouped together and these groups should be separated e.g. using blank lines or lines made of dashes where this increases the readability. Identifiers, comments etc. should be aligned vertically where this improves the readability.

Automatically generated VHDL models, for example from schematics or from State Machine diagrams, often have poor readability. Before deciding to use automatically generated code in a particular case, the interoperability and maintenance aspects need be assessed, as the master design description will then be in the internal format of the tool used, requiring all interacting users to have the same tool. Furthermore, the long-term availability of tools being able to read this particular format should be evaluated.

2.2 Names

Meaningful non-cryptic identifier names shall be used, based on the English language. Exceptionally, models only intended for local or national usage for the foreseeable fixture, might use the local language when beneficial. The same identifier name as for the actual hardware and as in the data sheet or similar documentation shall be used. For signals and variables that are active low, this shall be clearly indicated by their name, for example by suffixing *_N* as in *Reset_N.* In case a name would not be legal VHDL, it should be close to the original name and a comment should be included for clarification. The VHDL-93 extended identifiers (any string enclosed by two \ characters) should be used with precaution (see also 2.1 on the usage of the new features of VHDL-93).

A name should indicate the purpose of the object and not its type.
Example: an eight-bit loadable counter used for addressing should be called *AddressCounter* (its purpose) rather than *CountLoad8* (its type).

The naming convention (e.g. how active low and internal signals are indicated if registers are indicated with a special suffix etc.) should be documented in the file header for each file.

It is recommended to write identifiers using mixed casing with consistent casing in all the code. Underscore characters may be used, though excessive usage - such as multiple occurrences in one identifier - should be avoided. It is recommended to use less than 15 characters in the normal case, though the number of characters used for an identifier shall never exceed 28, due to an NFS limitation for file names.

The VHDL name of the predefined identifiers, including the identifiers in the *Std* and *IEEE* design libraries, shall never be used for other identifiers. Note for example the form-feed character *FF* and the *Time* unit *Min.*

A constant value should generally be specified using a constant rather than using a specified value. In particular, when the same constant value or a value derived from a constant appear more than once in a model, constants shall be used.

Where constant values depend on the characteristics of a specific type, such as the length of an **array,** the predefined attributes, such as *'Length,* shall be used instead of a specified value. This makes the code less dependent on the characteristics of the particular type used.

2.3 Comments

The purpose of comments is to allow the function of a model, package or testbench to be understood by a designer not involved in the development of the VHDL code.

All models shall be fully documented with explanatory comments in English. Exceptionally, models only intended for local or national usage for the foreseeable future might use the local language when beneficial. The comments shall be placed close to the part of the code they describe; a description only in the file header without comments in the executable part is not acceptable. All comments shall be indented and aligned for good readability. The comments shall be descriptive and not just direct translations or repetitions of the VHDL code.

Each file shall include a header as a minimum containing the following information:

- name of the design unit(s) in the file;
- file name;
- purpose of the code, description of hardware or functionality modelled;
- limitations to the model and known errors, if any, including any assumptions made;
- design library where the code is intended to be compiled in;
- list of all analysis dependencies, if any;
- naming convention;
- author(s) including full address;
- simulator(s), simulator version(s) and platform(s) used;
- change list, containing version numbers, author(s), the dates and a description of all changes performed.

Each subprogram declaration, subprogram, process, block etc. shall be immediately preceded by a description of its function, including any limitations and assumptions. For subprograms, the parameters and the result shall also be described.

For port and generic clauses in entity and component declarations, there shall be one signal declaration per line, directly followed by a comment describing the signal. Describing the signals in a group of comments separate from the declarations themselves are not recommended, being likely to become inconsistent in case of modification.

Where functionality is represented by data, as for example microcode or a PLA fuse-map program, the functionality shall be fully described. This applies regardless of the data representation (e.g. hard-coded constants or data read from an ASCII file).

2.4 Types

The leftmost bit of an array shall be the most significant, regardless of the bit ordering.
Example: In *Bit_Vector(0* ***to*** *15),* bit 0 is the Most Significant Bit (MSB), whereas in *Bit_Vector(15* ***downto*** *0),* bit 0 is the Least Significant Bit (LSB).

It is recommended to write the code so it is possible to change the type of a signal or variable without changing the simulation behaviour. This implies

- avoid relying on default initialisation of a variable or a signal unless a reset policy ensures that the model is explicitly initialised (typical for synthesizable constructs),
- avoid relying on the number of type values in a type declaration,
- avoid dependencies on the order in the type declaration.

Real literals shall only be written in decimal format. Based literals shall only be specified in base 2, 8, 10 or 16 and should not have an exponent. The use of underscore characters in literals should be restricted to binary, octal and hexadecimal literals. Hexadecimal literals shall be written using uppercase characters, for example *16#9ABC#.*

2.5 Files

For portability reasons, the only allowed file type is *Std.TextIO.Text.* However, it should be noted that there are still certain variances, such as (see further AD1 section 14.3)

- line delimiters might not be readable, and therefore characters with a lower rank than the *space* character should be avoided,
- *underline* character(s) and/or an exponent may be absent or present when writing values of the *Integer, Real* and *Time* types,
- the casing of the identifier when writing values of the *Boolean* type may vary.

Consequently, in case values of the *Boolean, Integer, Real* or *Time* types are written using *Std.TextIO,* the possible impact on portability should be analysed. The same applies when characters with lower rank than the *space* character is read from a file.

The predefined file *Std.TextIO.Input* should be avoided, since its implementation on different simulators varies. In particular, it shall never be used in testbenches for automated verification, since this could preclude the verification to be performed using a script. Also note that assertions may be output to the *Std.TextIO.Output* file by some simulators, whereas others might output them to a separate file.

When data is to be read from a text file, e.g. for initialising a memory, the format of the file shall be fully and clearly specified in the VHDL code implementing the reading function. An example should also be included.

It is recommended to limit the number of characters per line in a file to be read to 80 characters. In any case, it shall never exceed 255 characters.

2.6 Signals and ports

The same name shall be used for a signal throughout all levels of the model, wherever possible. In cases where exactly the same name cannot be used, e.g. when two identical sub-components have been instantiated, names derived from the same base name should be used.

The index ordering (i.e. using **to** or **downto**) of the model top-level entity port clause signals shall be identical to the one used in the data sheet or similar documentation. It is recommended to use the same index ordering in the whole model but in case the index order is reversed within the model, this shall be clearly marked every time the index order is different w.r.t. the corresponding signal at the highest level of the hierarchy.

The **buffer** mode shall never appear in the port clause of the model's top-level entity declaration, since ports of this mode have restrictions on the mode of other ports associated to this port.

The port clause signal declarations shall appear in a logical order. It is recommended to order the signals in the port clause after their mode; first input signals, followed by bi-directional signals and last output signals. Alternatively, the signals could be grouped together according to their function, and within each such group according to their mode. Port clauses shall be commented as specified in 2.3.

Port maps for component instantiations shall use named association unless all signals in the component instantiation have the same (or derived) name as in the component declaration. The same applies to generic maps where increasing the readability.

Duplicating a signal by assignment to another signal only to rename the signal, to allow another port mode to be used or to perform a type conversion, should only be used where necessary or where clearly increasing the readability.

2.7 Assertions and reporting

Assertions shall be used to report model errors, timing violations and when signals have illegal or unknown values affecting the model behaviour. The following policy for assigning severity levels is recommended:

- *Failure*: errors in the model itself (e.g. if a statement that is believed to be impossible to execute actually is executed);
- *Error*: timing violations and invalid data affecting the control state of the model, including illegal combinations of mode signals and of control signals (e.g. unknown data on a mode input or too short reset time);
- *Warning*: timing violations and invalid data not affecting the control state of the model, but which could affect the simulation behaviour of the model (e.g. if data to be sent out from an interface is invalid);
- *Note*: essential information that is not classified in the other severity levels, such as reporting from which text file data is read, which testbench is executed, if an event is detected on an input signal whose function has not been implemented (e.g. activation of production test) etc.

A model should not issue assertions for insignificant events, for example at start, during reset or if an event has no impact on the simulation behaviour. Neither should unnecessary messages be generated, e.g. as reporting if Worst Case timing has been selected, since many insignificant messages will hide the important ones.

The assertion report shall give a clear description of the reason for the assertion and shall include the hierarchical path to the instance or package, as well as identifying the signal(s) where applicable. It is sufficient to report the hierarchical path relative to the top-level entity of the model before VHDL-93 has been fully introduced, then the predefined attribute *'Instance_Name* should be used.

Reporting using *Std.TextIO.Output* instead of assertions gives shorter messages by eliminating information such as assertion level, time etc. In cases where such additional information is not needed, *Std.TextIO.Output* could be used, e.g. for reporting operating modes, for disassembly and for testbenches. In such case, the model instance path should be inserted at the beginning of the message.

2.8 Subprograms, processes, entities, architectures, component declarations

All processes shall be associated with a descriptive label. The same applies for other concurrent statements where this will increase the readability.

A process with only one statement of the type "**wait on** *SignalName;*" - typical for synthesizable processes - should use a process sensitivity list instead of the **wait on** statement to increase the readability.

Wherever possible, all language constructs such as subprograms, package declarations and bodies, entities, architectures, processes and **loop** statements shall be qualified, i.e. the identifier associated with the construct shall also appear at its end.

Procedures that modify signals or variables not passed as parameters in the procedure call should be avoided. Nevertheless, in some cases such as testbenches, this technique could actually increase the readability of the code by hiding insignificant details. If used, it shall be clearly commented which signals and variables can be modified by the procedure call.

The top-level entity should have the same name as the device or hardware modelled. Declarations other than assertion statements, generic and port clauses should be avoided in an entity declaration in order to make a clear distinction between the model interface (i.e. the entity) and the model functionality or connectivity (i.e. the architecture).

The identifier, port clause and generic clause of a component declaration shall be identical (i.e. use the same identifiers and the same ordering) to the declarations in the corresponding entity declaration.

2.9 Configurations

All design units being part of a model which should not be reconfigured or separated (such as all design units being part of a model of a standard component) shall be directly instantiated in the architecture(s) of the model. It should not be possible to reconfigure them using a configuration specification.

For design units intended for reconfiguration, there shall be no configuration specifications within the design units themselves, since it would then not be possible to reconfigure the model using a configuration specification.

2.10 Packages

Where possible, packages approved by the IEEE should be used rather than redeveloping similar functionality in order to reduce development cost as well as the number of errors in the packages and to allow speed optimised versions to be provided with the VHDL simulators (see AD2, RD3, RD4, RD5). In case a package is used before IEEE approval, it shall be placed in the same design library as the model itself, and not be in the *IEEE* library.

Packages specific to a particular CAD tool should only be used when unavoidable. In particular, any source code distribution restrictions should be assessed, if applicable.

The number of packages used by a model shall not be excessive. There shall be no empty or almost empty packages, unless where this clearly increases code readability. It is recommended to place VHDL code concerning different functionality areas in different packages, e.g. all timing parameters in one package, all subprograms related to timing in another etc. However, there should not be a separate package for each entity where constants etc. used by that entity are defined.

The package declaration shall contain full documentation about the declared types, constants, subprograms etc.

The declarations in a package body shall appear in the same order as the corresponding declarations in the package declaration.

Each package containing one or more subprograms - except packages approved by the IEEE - shall be separately and extensively verified as specified in 2.13, using a testbench allowing automated verification as described in 3.4.1.

2.11 Design libraries

The model design units shall be placed in a design library other than *Work*. This will normally be a separate design library for each model, though families of devices, such as 74-series logic or a collection of different memories, are preferably grouped together in one design library.

This design library shall be named after the device, respectively the family, with the suffix *Lib* appended. The top-level entity to be used for simulation shall have the same name as the device.
Example: a device *XYZ* should be placed in the library *XYZ_Lib* and should be used as *XYZ_Lib.XYZ.* The situation when the same name is used in different developments should be avoided by not using generic names (larger risk for duplication) and where possible investigate already known names.

This design library shall contain all design units used by the model itself (including packages), except for the packages in design library *Std*, the packages in the *IEEE* design library and common packages used by many different models. Major criteria to decide whether or not a package is to be regarded as common are standardisation by an international organisation such as the IEEE and international acceptance as a defacto standard.

The testbench(es) used for the verification of the device shall be placed in a design library different from the device design library, such as *XYZ_TB_Lib* or *Work*. This design library should contain all hierarchical sub-components and packages used except the model to be tested (already being in a separate library) and standard and common packages as defined above.

The *IEEE* design library shall not contain any other packages than those approved by the IEEE. Neither shall these packages be modified or extended. Some CAD companies may place own defined packages in the *IEEE* design library. In case such a package is used, it shall be moved to the design library where it is used.

2.12 Constructs to be avoided

The VHDL code shall be fully deterministic when executed regardless of the simulator used. This means for example:

- There shall be no communication between different parts of the model through files;
- Resolution functions shall always be commutative and associative;
- Shared variables (VHDL-93) shall only be used when absolutely necessary. It shall then be proven by analysis that the usage is fully deterministic, which should be documented;
- Care should be taken when using floating point values, especially conversion to and from floating point values, comparisons between floating point values and events on floating point values. Specifically, using pseudo-random test patterns is not portable if the pseudorandom generator is using the *Real* type;
- The *Std.TextIO* portability limitations shall be avoided, see 2.5.

Refer to appendix C of AD1 for more information.

CAD tool specific types shall not be used. Features specific to an operating system, such as links and the */dev/null* file on Unix systems, should be avoided. Absolute paths shall not be used for filenames.

Objects with an implicitly declared index, for example a line returned from the *Std.TextIO.Read* procedure for a string, shall never be used with absolute indexing. Instead, the predefined attributes for indexing, such as *'Left,* shall be used. As a consequence, absolute indexing shall be used when declaring an object to be referenced using an absolute index.

The dependence on implementation defined limitations, for example 32-bit limitations on the *Integer* and *Time* types, shall be minimised. In particular, a model should not encounter implementation defined limitations on *Time* as long as the simulated time does not exceed the limitation.

Subprograms and components should not be renamed by encapsulating them with subprograms or components with other names unless where this clearly increases the readability.

Signals, variables, constants, subprograms or components shall not be hidden by declaring another object with the same name. Overloading is not considered as hiding and is encouraged where beneficial.

The predefined operators, subprograms, attributes etc. shall never be redefined. This shall also apply to the packages in the *IEEE* design library. Neither shall similar declarations using the same names be created.

Since the model shall be placed in another design library than *Work,* there shall be no references to *Work* within the code for the model and its packages.

The constructs below are considered as obsolescent. Being not strictly necessary to use for modelling, they should therefore not be used:

- guarded expressions, signals and assignments, including the reserved words **bus, disconnect, guarded, register**;
- the linkage mode for interface declarations;
- the *Allowed Replacement Characters* defined in section 13.10 of AD1;
- the *Std.TextIO.EndLine* function, *L'Length = 0* could be used instead *(EndLine* has been excluded from VHDL-87 being illegal VHDL);
- file types other than *Std.TextIO.Text.*

2.13 Verification

The purpose of the verification shall be to verify that the developed model is correct, with few or no errors being found. It shall not be a means to locate errors in the VHDL code in order to patch them.

A model or package shall first be verified by its developer, being the person with the best knowledge of its function. The final verification shall be performed by somebody not involved in the creation of that model or package to avoid that a misunderstanding of the functionality is masked by the same misunderstanding in the verification.

In case another simulation model is available, the VHDL model should also be verified versus this other model, regardless whether or not the other model is a VHDL model.

The verification shall solely be performed using VHDL testbenches as specified in 3.4, no simulator specific features or commands shall be used.