

IEEE Std 1076.6™-2004

(Revision of
IEEE Std 1076.6-1999)

IEEE Standards

1076.6™

IEEE Standard for VHDL Register Transfer Level (RTL) Synthesis

IEEE Computer Society

Sponsored by the
Design Automation Standards Committee



3 Park Avenue, New York, NY 10016-5997, USA

11 October 2004

Print: SH95242

PDF: SH95242



Documents

Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on March 06, 2026 at 05:10:02 UTC from IEEE Xplore. Restrictions apply.

Full document details: [IEEE 1076.6-2004](#)

Recognized as an
American National Standard (ANSI)

IEEE Std 1076.6™-2004
(Revision of
IEEE Std 1076.6-1999)

IEEE Standard for VHDL Register Transfer Level (RTL) Synthesis

Sponsor

Design Automation Standards Committee
of the
IEEE Computer Society

Approved 25 August 2004

American National Standard Institute

Approved 12 May 2004

IEEE-SA Standards Board

Abstract: This document specifies a standard for use of very high-speed integrated circuit hardware description language (VHDL) to model synthesizable register-transfer level digital logic. A standard syntax and semantics for VHDL register-transfer level synthesis is defined. The subset of the VHDL language, which is synthesizable, is described, and nonsynthesizable VHDL constructs are identified that should be ignored or flagged as errors.

Keywords: hardware description language, logic synthesis, register transfer level (RTL), very high-speed integrated circuit hardware description language (VHDL)

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

Copyright © 2004 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 11 October 2004. Printed in the United States of America.

IEEE is a registered trademark in the U.S. Patent & Trademark Office, owned by the Institute of Electrical and Electronics Engineers, Incorporated.

Print: ISBN 0-7381-4064-3 SH95242
PDF: ISBN 0-7381-4065-1 SS95242

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

Use of an IEEE Standard is wholly voluntary. The IEEE disclaims liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEEE Standards documents are supplied “**AS IS**.”

The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

In publishing and making this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEEE Standards document, should rely upon the advice of a competent professional in determining the exercise of reasonable care in any given circumstances.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position, explanation, or interpretation of the IEEE.

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

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

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

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Introduction

(This introduction is not part of IEEE Std 1076.6-2004, IEEE Standard for VHDL Register Transfer Level (RTL) Synthesis.)

This standard describes a standard syntax and semantics for VHDL RTL synthesis. It defines the subset of IEEE Std 1076™-2002 (VHDL) that is suitable for RTL synthesis and defines the semantics of that subset for the synthesis domain. This standard is based on IEEE Std 1076-2002, IEEE Std 1164™-1993, and IEEE Std 1076.3™-1997.

The purpose of this standard is to define a syntax and semantics that can be used in common by all compliant RTL synthesis tools to achieve uniformity of results in a similar manner to which simulation tools use IEEE Std 1076-2002. This will allow users of synthesis tools to produce well-defined designs whose functional characteristics are independent of a particular synthesis implementation by making their designs compliant with this standard.

The standard is intended for use by logic designers and electronic engineers.

This document specifies IEEE Std 1076.6-2004, which is a revision of IEEE Std 1076.6-1999. The VHDL Synthesis Interoperability Working Group (SIWG) of the IEEE Computer Society started the development of IEEE Std 1076.6-2004 in January 1998. The work initially started as a Level 2 effort (Level 1 being IEEE Std 1076.6-1999). In fact the work on Level 2 continued right after Level 1 was completed by the working group. The working group realized that a Level 2 was required and that it would take some time to develop and continued working on it at regular face-to-face meetings and teleconferences. As the Level 2 draft continued to mature, the working group decided that rather than having two different levels of synthesis subsets, it was better to just have one standard, with IEEE Std 1076.6-2004 becoming Level 2.

The intent of this version was to include a maximum subset of VHDL that could be used to describe synthesizable RTL logic. This included considering new features introduced by IEEE Std 1076-2002, new semantics based on algorithmic styles rather than template-driven, and a set of synthesis attributes that could be used to annotate an RTL description. The following team leaders drove this effort:

Syntax: Lance Thompson

Semantics: Vinaya Singh

Attributes: Sanjiv Narayan

In addition, the following provided much-needed additional support:

Web and reflector admin: David Bishop

Documentation: John Michael Williams

A majority of the work conducted by the working group was done via teleconferencing, which was held regularly and open to all. Also, the working group used an e-mail reflector and its web page effectively to distribute and share information.

The following volunteers contributed to the development of this standard:

J. Bhasker, Chair

Jim Lewis, Vice-Chair

Rob Anderson
Bill Anker
Victor Berman
David Bishop
Dominique Borrione
Dennis Brophy
Andrew Brown
Patrick Bryant
Ben Cohen
Tim Davis
Colin Dente
Wolfgang Ecker
Bob Flatt
Christopher Grimm
Steve Grout

Rich Hatcher
Mohammad Kakoei
Masamichi Kawarabayashi
Apurva Kalia
Satish Kumar
Evan Lavelle
Vijay Madisetti
Erich Marschner
Paul Menchini
Amitabh Menon
Egbert Molenkamp
Bob Myers
Sanjana Nair
Sanjiv Narayan
Zain Navabi

Jonas Nilsson
Alain Raynaud
Mehrdad Reshadi
Fredj Rouatbi
Steve Schultz
Manish Shrivastava
Vinaya Singh
Douglas Smith
Lance Thompson
Alessandro Uber
Jim Vellenga
Eugenio Villar
John Michael Williams
Francisco De Ycaza
Alex Zamfirescu

Development of IEEE Std 1076.6-1999

Initial work on this standard started as a synthesis interoperability working group under VHDL International. The working group was also chartered by the EDA Industry Council Project Technical Advisory Board (PTAB) to develop a draft based on the donated subsets by the following companies/groups:

- Cadence
- European Synthesis Working Group
- IBM
- Mentor Graphics
- Synopsys

After the PTAB approved of the draft 1.5 with an overwhelming affirmative response, an IEEE PAR was obtained to clear its way for IEEE standardization. Most of the members of the original group continued to be part of the Pilot Group under P1076.6 to lead the technical work.

At the time the 1999 standard was completed, the P1076.6 Pilot Team had the following membership:

Rob Anderson
Victor Berman
J. Bhasker
David Bishop
Dominique Borrione
Dennis Brophy
Ben Cohen
Colin Dente

Wolfgang Ecker
Bob Flatt
Christopher Grimm
Rich Hatcher
Apurva Kalia
Masamichi Kawarabayashi
Jim Lewis
Sanjiv Narayan

Doug Perry
Steve Schultz
Doug Smith
Lance Thompson
Fur-Shing Tsai
Jim Vellenga
Eugenio Villar
Nels Vander Zanden

Many individuals from different organizations contributed to the development of this standard. In particular, in addition to the Pilot Team, the following individuals contributed to the development of the standard by being part of the working group:

Bill Anker
LaNae Avra

Robert Blackburn

John Hillawi
Pradip Jha

In addition, 95 individuals on the working group e-mail reflector also contributed to this development.

Notice to users

Errata

Errata, if any, for this and all other standards can be accessed at the following URL: <http://standards.ieee.org/reading/ieee/updates/errata/index.html>. Users are encouraged to check this URL for errata periodically.

Interpretations

Current interpretations can be accessed at the following URL: <http://standards.ieee.org/reading/ieee/interp/index.html>.

Patents

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

Participants

The following members of the individual balloting committee voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

Bill Anker	Guru Dutt Dhingra	D. C. Mohla
Peter Ashenden	Colin Dente	E. Molenkamp
John Aynsley	George Economakos	Serafin A. Perez Lopez
Stephen Bailey	Peter Flake	John Shields
Jayaram Bhasker	Ian Andrew Guyler	Mark Tillinghast
Stefen Boyd	William A. Hanna	John Michael Williams
Kai Moon Chow	Jim Lewis	Mark Zwolinski
Keith Chow	Michael McNamara	

When the IEEE-SA Standards Board approved this standard on 12 May 2004, it had the following membership:

Don Wright, Chair
Steve M. Mills, Vice Chair
Judith Gorman, Secretary

Chuck Adams	Raymond Hapeman	Paul Nikolich
H. Stephen Berger	Richard J. Holleman	T. W. Olsen
Mark D. Bowman	Richard H. Hulett	Ronald C. Petersen
Joseph A. Bruder	Lowell G. Johnson	Gary S. Robinson
Bob Davis	Joseph L. Koepfinger*	Frank Stone
Roberto de Boisson	Hermann Koch	Malcolm V. Thaden
Julian Forster*	Thomas J. McGean	Doug Topping
Arnold M. Greenspan	Daleep C. Mohla	Joe D. Watson
Mark S. Halpin		
*Member Emeritus		

Also included are the following nonvoting IEEE-SA Standards Board liaisons:

Satish K. Aggarwal, *NRC Representative*
Richard DeBlasio, *DOE Representative*
Alan Cookson, *NIST Representative*

Don Messina
IEEE Standards Project Editor

Contents

1.	Overview.....	1
1.1	Scope.....	1
1.2	Compliance to this standard.....	1
1.3	Terminology.....	2
1.4	Conventions	2
2.	References.....	3
3.	Definitions and acronyms	3
3.1	Definitions	3
3.2	Acronyms.....	4
4.	Predefined types.....	5
5.	Verification methodology	5
5.1	Combinational verification	6
5.2	Sequential verification	6
6.	Modeling hardware elements.....	7
6.1	Edge-sensitive sequential logic.....	7
6.2	Level-sensitive sequential logic.....	19
6.3	Three-state logic and busses	23
6.4	Combinational logic.....	23
6.5	ROM and RAM memories.....	24
7.	Pragmas.....	29
7.1	Attributes	29
7.2	Metacomments.....	46
8.	Syntax	47
8.1	Design entities and configurations.....	47
8.2	Subprograms and packages.....	52
8.3	Types.....	56
8.4	Declarations	61
8.5	Specifications.....	67
8.6	Names	69
8.7	Expressions	71
8.8	Sequential statements.....	75
8.9	Concurrent statements.....	81
8.10	Scope and visibility.....	86
8.11	Design units and their analysis	87
8.12	Elaboration.....	88
8.13	Lexical elements	88
8.14	Predefined language environment	88
	Annex A (informative) Syntax summary.....	91
	Annex B (normative) Synthesis package RTL_ATTRIBUTES.....	110
	Index	111

IEEE Std 1076.6™-2004

(Revision of
IEEE Std 1076.6-1999)

IEEE Standards

1076.6™

IEEE Standard for VHDL Register Transfer Level (RTL) Synthesis

IEEE Computer Society

Sponsored by the
Design Automation Standards Committee



3 Park Avenue, New York, NY 10016-5997, USA

11 October 2004

Print: SH95242

PDF: 1076.6-2004



Documents

Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on March 06, 2026 at 05:10:02 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 1076.6-2004

Recognized as an
American National Standard (ANSI)

IEEE Std 1076.6™-2004
(Revision of
IEEE Std 1076.6-1999)

IEEE Standard for VHDL Register Transfer Level (RTL) Synthesis

Sponsor

Design Automation Standards Committee
of the
IEEE Computer Society

Approved 25 August 2004

American National Standard Institute

Approved 12 May 2004

IEEE-SA Standards Board

Abstract: This document specifies a standard for use of very high-speed integrated circuit hardware description language (VHDL) to model synthesizable register-transfer level digital logic. A standard syntax and semantics for VHDL register-transfer level synthesis is defined. The subset of the VHDL language, which is synthesizable, is described, and nonsynthesizable VHDL constructs are identified that should be ignored or flagged as errors.

Keywords: hardware description language, logic synthesis, register transfer level (RTL), very high-speed integrated circuit hardware description language (VHDL)

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

Copyright © 2004 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 11 October 2004. Printed in the United States of America.

IEEE is a registered trademark in the U.S. Patent & Trademark Office, owned by the Institute of Electrical and Electronics Engineers, Incorporated.

Print: ISBN 0-7381-4064-3 SH95242
PDF: ISBN 0-7381-4065-1 SS95242

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

Use of an IEEE Standard is wholly voluntary. The IEEE disclaims liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEEE Standards documents are supplied “**AS IS.**”

The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

In publishing and making this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEEE Standards document, should rely upon the advice of a competent professional in determining the exercise of reasonable care in any given circumstances.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position, explanation, or interpretation of the IEEE.

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

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

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

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Introduction

(This introduction is not part of IEEE Std 1076.6-2004, IEEE Standard for VHDL Register Transfer Level (RTL) Synthesis.)

This standard describes a standard syntax and semantics for VHDL RTL synthesis. It defines the subset of IEEE Std 1076™-2002 (VHDL) that is suitable for RTL synthesis and defines the semantics of that subset for the synthesis domain. This standard is based on IEEE Std 1076-2002, IEEE Std 1164™-1993, and IEEE Std 1076.3™-1997.

The purpose of this standard is to define a syntax and semantics that can be used in common by all compliant RTL synthesis tools to achieve uniformity of results in a similar manner to which simulation tools use IEEE Std 1076-2002. This will allow users of synthesis tools to produce well-defined designs whose functional characteristics are independent of a particular synthesis implementation by making their designs compliant with this standard.

The standard is intended for use by logic designers and electronic engineers.

This document specifies IEEE Std 1076.6-2004, which is a revision of IEEE Std 1076.6-1999. The VHDL Synthesis Interoperability Working Group (SIWG) of the IEEE Computer Society started the development of IEEE Std 1076.6-2004 in January 1998. The work initially started as a Level 2 effort (Level 1 being IEEE Std 1076.6-1999). In fact the work on Level 2 continued right after Level 1 was completed by the working group. The working group realized that a Level 2 was required and that it would take some time to develop and continued working on it at regular face-to-face meetings and teleconferences. As the Level 2 draft continued to mature, the working group decided that rather than having two different levels of synthesis subsets, it was better to just have one standard, with IEEE Std 1076.6-2004 becoming Level 2.

The intent of this version was to include a maximum subset of VHDL that could be used to describe synthesizable RTL logic. This included considering new features introduced by IEEE Std 1076-2002, new semantics based on algorithmic styles rather than template-driven, and a set of synthesis attributes that could be used to annotate an RTL description. The following team leaders drove this effort:

Syntax: Lance Thompson

Semantics: Vinaya Singh

Attributes: Sanjiv Narayan

In addition, the following provided much-needed additional support:

Web and reflector admin: David Bishop

Documentation: John Michael Williams

A majority of the work conducted by the working group was done via teleconferencing, which was held regularly and open to all. Also, the working group used an e-mail reflector and its web page effectively to distribute and share information.

The following volunteers contributed to the development of this standard:

J. Bhasker, *Chair*

Jim Lewis, *Vice-Chair*

Rob Anderson
Bill Anker
Victor Berman
David Bishop
Dominique Borriane
Dennis Brophy
Andrew Brown
Patrick Bryant
Ben Cohen
Tim Davis
Colin Dente
Wolfgang Ecker
Bob Flatt
Christopher Grimm
Steve Grout

Rich Hatcher
Mohammad Kakoe
Masamichi Kawarabayashi
Apurva Kalia
Satish Kumar
Evan Lavelle
Vijay Madisetti
Erich Marschner
Paul Menchini
Amitabh Menon
Egbert Molenkamp
Bob Myers
Sanjana Nair
Sanjiv Narayan
Zain Navabi

Jonas Nilsson
Alain Raynaud
Mehrdad Reshadi
Fredj Rouatbi
Steve Schultz
Manish Shrivastava
Vinaya Singh
Douglas Smith
Lance Thompson
Alessandro Uber
Jim Vellenga
Eugenio Villar
John Michael Williams
Francisco De Ycaza
Alex Zamfirescu

Development of IEEE Std 1076.6-1999

Initial work on this standard started as a synthesis interoperability working group under VHDL International. The working group was also chartered by the EDA Industry Council Project Technical Advisory Board (PTAB) to develop a draft based on the donated subsets by the following companies/groups:

- Cadence
- European Synthesis Working Group
- IBM
- Mentor Graphics
- Synopsys

After the PTAB approved of the draft 1.5 with an overwhelming affirmative response, an IEEE PAR was obtained to clear its way for IEEE standardization. Most of the members of the original group continued to be part of the Pilot Group under P1076.6 to lead the technical work.

At the time the 1999 standard was completed, the P1076.6 Pilot Team had the following membership:

Rob Anderson
Victor Berman
J. Bhasker
David Bishop
Dominique Borriane
Dennis Brophy
Ben Cohen
Colin Dente

Wolfgang Ecker
Bob Flatt
Christopher Grimm
Rich Hatcher
Apurva Kalia
Masamichi Kawarabayashi
Jim Lewis
Sanjiv Narayan

Doug Perry
Steve Schultz
Doug Smith
Lance Thompson
Fur-Shing Tsai
Jim Vellenga
Eugenio Villar
Nels Vander Zanden

Many individuals from different organizations contributed to the development of this standard. In particular, in addition to the Pilot Team, the following individuals contributed to the development of the standard by being part of the working group:

Bill Anker
LaNae Avra

Robert Blackburn

John Hillawi
Pradip Jha

In addition, 95 individuals on the working group e-mail reflector also contributed to this development.

Notice to users

Errata

Errata, if any, for this and all other standards can be accessed at the following URL: <http://standards.ieee.org/reading/ieee/updates/errata/index.html>. Users are encouraged to check this URL for errata periodically.

Interpretations

Current interpretations can be accessed at the following URL: <http://standards.ieee.org/reading/ieee/interp/index.html>.

Patents

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

Participants

The following members of the individual balloting committee voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

Bill Anker	Guru Dutt Dhingra	D. C. Mohla
Peter Ashenden	Colin Dente	E. Molenkamp
John Aynsley	George Economakos	Serafin A. Perez Lopez
Stephen Bailey	Peter Flake	John Shields
Jayaram Bhasker	Ian Andrew Guyler	Mark Tillinghast
Stefen Boyd	William A. Hanna	John Michael Williams
Kai Moon Chow	Jim Lewis	Mark Zwolinski
Keith Chow	Michael McNamara	

When the IEEE-SA Standards Board approved this standard on 12 May 2004, it had the following membership:

Don Wright, Chair
Steve M. Mills, Vice Chair
Judith Gorman, Secretary

Chuck Adams	Raymond Hapeman	Paul Nikolic
H. Stephen Berger	Richard J. Holleman	T. W. Olsen
Mark D. Bowman	Richard H. Hulett	Ronald C. Petersen
Joseph A. Bruder	Lowell G. Johnson	Gary S. Robinson
Bob Davis	Joseph L. Koepfinger*	Frank Stone
Roberto de Boisson	Hermann Koch	Malcolm V. Thaden
Julian Forster*	Thomas J. McGean	Doug Topping
Arnold M. Greenspan	Daleep C. Mohla	Joe D. Watson
Mark S. Halpin		
*Member Emeritus		

Also included are the following nonvoting IEEE-SA Standards Board liaisons:

Satish K. Aggarwal, *NRC Representative*
Richard DeBlasio, *DOE Representative*
Alan Cookson, *NIST Representative*

Don Messina
IEEE Standards Project Editor

Contents

1.	Overview.....	1
1.1	Scope.....	1
1.2	Compliance to this standard.....	1
1.3	Terminology.....	2
1.4	Conventions	2
2.	References.....	3
3.	Definitions and acronyms	3
3.1	Definitions	3
3.2	Acronyms.....	4
4.	Predefined types.....	5
5.	Verification methodology	5
5.1	Combinational verification	6
5.2	Sequential verification.....	6
6.	Modeling hardware elements.....	7
6.1	Edge-sensitive sequential logic.....	7
6.2	Level-sensitive sequential logic.....	19
6.3	Three-state logic and busses	23
6.4	Combinational logic.....	23
6.5	ROM and RAM memories.....	24
7.	Pragmas.....	29
7.1	Attributes	29
7.2	Metacomments.....	46
8.	Syntax	47
8.1	Design entities and configurations.....	47
8.2	Subprograms and packages.....	52
8.3	Types.....	56
8.4	Declarations	61
8.5	Specifications.....	67
8.6	Names	69
8.7	Expressions	71
8.8	Sequential statements.....	75
8.9	Concurrent statements.....	81
8.10	Scope and visibility.....	86
8.11	Design units and their analysis	87
8.12	Elaboration.....	88
8.13	Lexical elements	88
8.14	Predefined language environment	88
	Annex A (informative) Syntax summary.....	91
	Annex B (normative) Synthesis package RTL_ATTRIBUTES.....	110
	Index	111

IEEE Standard for VHDL Register Transfer Level (RTL) Synthesis

1. Overview

1.1 Scope

This standard defines a subset of very high-speed integrated circuit hardware description language (VHDL) that ensures portability of VHDL descriptions between register transfer level synthesis tools. Synthesis tools may be compliant and yet have features beyond those required by this standard. This standard defines how the semantics of VHDL shall be used, for example, to model level-sensitive and edge-sensitive logic. It also describes the syntax of the language with reference to what shall be supported and what shall not be supported for interoperability.

Use of this standard should minimize the potential for functional simulation mismatches between models before they are synthesized and after they are synthesized.

1.2 Compliance to this standard

1.2.1 Model compliance

A VHDL model shall be defined as being compliant to this standard if the model

- a) Uses only constructs described as supported or ignored in this standard
- b) Adheres to the semantics defined in this standard

1.2.2 Tool compliance

A synthesis tool shall be defined as being compliant to this standard if it

- a) Accepts all models that adhere to the model compliance definition defined in 1.2.1
- b) Supports language related pragmas defined by this standard
- c) Produces a circuit model that has the same functionality as the input model based on the verification process as outlined in Clause 5.

1.3 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.

The constructs in the standard shall be categorized as follows:

Supported: RTL synthesis shall interpret a construct, that is, map the construct to an equivalent hardware representation.

Ignored: RTL synthesis shall ignore the construct and produce a warning. Encountering the construct shall not cause synthesis to fail, but synthesis results may not match simulation results. The mechanism, if any, by which RTL synthesis notifies (warns) the user of such constructs is not defined by this standard. Ignored constructs may include unsupported constructs.

Not Supported: RTL synthesis does not support the construct. RTL synthesis does not expect to encounter the construct, and the failure mode shall be undefined. RTL synthesis may fail upon encountering such a construct. Failure is not mandatory; more specifically, RTL synthesis is allowed to treat such a construct as ignored.

NOTE—A synthesis tool may interpret constructs that are identified as not supported in this standard. However a model that contains such unsupported constructs is not compliant with this standard.¹

1.4 Conventions

This standard uses the following conventions:

- The body of the text of this standard uses **boldface** to denote VHDL reserved words (such as **downto**).
- The text of the VHDL examples and code fragments is represented in a fixed-width font.
- Syntax text that is struck-through (e.g., ~~text~~) refers to syntax that shall not be supported.
- Syntax text that is underscored (e.g., text) refers to syntax that shall be ignored.
- < and > pairs are used to represent text in one of several different, but specific forms. For example, one of the forms of <clock_edge> could be “CLOCK'EVENT and CLOCK = '1”.
- Any paragraph starting with “NOTE—” is informative and not part of the standard.
- The examples that appear in this document under “*Example:*” are for the sole purpose of demonstrating the syntax and semantics of VHDL for synthesis. It is not the intent of this standard to demonstrate, recommend, or emphasize coding styles that are more (or less) efficient in generating an equivalent hardware representation. In addition, it is not the intent of this standard to present examples that represent a compliance test suite, or a performance benchmark, even though these examples are compliant to this standard (except as noted otherwise).

¹Notes in text, tables, and figures are given for information only and do not contain requirements needed to implement the standard.

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™-2002, IEEE Standard VHDL Language Reference Manual.^{2, 3}

IEEE Std 1076.3™-1997, IEEE Standard Synthesis Packages (NUMERIC_BIT and NUMERIC_STD).

IEEE Std 1164™-1993, IEEE Standard Multivalued Logic System for VHDL Model Interoperability (STD_LOGIC_1164).

3. Definitions and acronyms

3.1 Definitions

For the purposes of this standard, the following terms and definitions apply. *The Authoritative Dictionary of IEEE Standards Terms, Seventh Edition* should be referenced for terms not defined in this clause. Terms used within this standard but not defined in this clause are assumed to be from IEEE Std 1076-2002, IEEE Std 1164-1993, or IEEE Std 1076.3-1997.⁴

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

3.1.2 combinational logic: Logic that settles to a state entirely determined by the current input values and therefore that cannot store information. Any change in the input causes a new state completely defined by the new inputs.

3.1.3 don't care value: The enumeration literal ‘-’ of the type STD_ULOGIC (or subtype STD_LOGIC).

3.1.4 edge-sensitive storage element: Any storage element mapped to by a synthesis tool that

- a) Propagates the value at the data input whenever an appropriate transition in value is detected on a clock control input
- b) Preserves the last value propagated at all other times, except when any asynchronous control inputs become active (for example, a flip-flop)

3.1.5 high-impedance value: The enumeration literal ‘Z’ of the type STD_ULOGIC (or subtype STD_LOGIC).

3.1.6 level-sensitive storage element: Any storage element mapped to by a synthesis tool that

- a) Propagates the value at the data input whenever an appropriate value is detected on a clock control input
- b) Preserves the last value propagated at all other times, except when any asynchronous control inputs become active (for example, a latch)

²The IEEE standards or products referred to in this clause are trademarks of the Institute of Electrical and Electronics Engineers, Inc.

³IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA (<http://standards.ieee.org/>).

⁴Information on references can be found in Clause 2.

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

3.1.8 metacomment: A VHDL comment (--) that is used to provide synthesis-specific interpretation by a synthesis tool.

3.1.9 metalogical value: One of the enumeration literals 'U', 'X', 'W', or '-' of the type STD_ULOGIC (or subtype STD_LOGIC).

3.1.10 pragma: A generic term used to define a construct with no predefined language semantics that influences how a synthesis tool will synthesize VHDL code into an equivalent hardware representation.

3.1.11 sequential logic: Logic that settles to a state not determined solely by current inputs. The current state of such logic can be determined only by knowing the current inputs and some history of past inputs in their sequential order. Sequential logic always stores information from past input and therefore may be used to implement storage elements.

3.1.12 synchronous assignment: An assignment that takes place when a signal or variable value is updated as a direct result of a clock edge expression evaluating as true.

3.1.13 synthesis library: A library of digital design objects such as logic gates, chip pads, memory blocks, or other blocks; instances of these elements are connected together by a synthesis tool to create a synthesized netlist.

3.1.14 synthesis tool: Any system, process, or tool that interprets register transfer level VHDL source code as a description of an electronic circuit and derives a netlist description of that circuit.

3.1.15 synthesis-specific attribute: An attribute recognized by a tool compliant to this standard.

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

3.1.17 vector: A one-dimensional array.

3.1.18 well-defined: Containing no metalogical or high-impedance value.

3.2 Acronyms

LRM	The IEEE VHDL language reference manual, that is, IEEE Std 1076-2002.
RTL	The register transfer level of modeling circuits in VHDL for use with register transfer level synthesis. Register transfer level is a level of description of a digital design in which the clocked behavior of the design is expressly described in terms of data transfers between storage elements in sequential logic, which may be implied, and combinational logic, which may represent any computing or arithmetic-logic-unit logic. RTL modeling allows design hierarchy that represents a structural description of other RTL models.

4. Predefined types

A synthesis tool, compliant with this standard, shall support the following predefined types:

- a) BIT, BOOLEAN, and BIT_VECTOR as defined by IEEE Std 1076-2002
- b) CHARACTER and STRING as defined in IEEE Std 1076-2002
- c) INTEGER as defined in IEEE Std 1076-2002
- d) STD_ULOGIC, STD_ULOGIC_VECTOR, STD_LOGIC, and STD_LOGIC_VECTOR as defined by the package STD_LOGIC_1164 (IEEE Std 1164-1993)
- e) SIGNED and UNSIGNED as defined by the VHDL package NUMERIC_BIT as part of IEEE Std 1076.3-1997
- f) SIGNED and UNSIGNED as defined by the VHDL package NUMERIC_STD as part of IEEE Std 1076.3-1997

The synthesis interpretation of the values that belong to type STD_ULOGIC shall be as defined in IEEE Std 1076.3-1997.

No array type, other than those listed in e) and f), shall be used to represent signed or unsigned numbers.

The synthesis tool shall also support user-defined and other types derived from the predefined types according to the rules of 8.3.

By definition, if a type with a metalogical or high-impedance value is used in a model, then this type shall have as an ancestor a type that belongs to the package STD_LOGIC_1164 (IEEE Std 1164-1993).

5. Verification methodology

Synthesized results may be broadly classified as either combinational or sequential. Sequential logic has some form of internal storage (latch, register, memory). Combinational logic has outputs that are solely a function of the inputs with no internal loops and no internal storage. Designs may contain both sequential and combinational parts.

The process of verifying synthesis results using simulation consists of applying equivalent inputs to both the original model and synthesized model and then comparing their outputs to ensure that they are equivalent. Equivalent in this context means that a synthesis tool shall produce a circuit that is equivalent at the input, output, and bidirectional ports of the model. As synthesis in general does not recognize the same delays as simulators, the outputs cannot be compared at every simulation time. Rather, they can only be compared at specific simulation times when all transient delays have settled and all active timeout clauses have been exceeded. If the outputs do not match at all comparable times, the synthesis tool shall not be compliant. There shall be no matching requirement placed on any internal nodes.

The input stimulus shall comply with the following criteria:

- a) Input data does not contain metalogical or high-impedance values.
- b) Input data may only contain 'H' and 'L' on inputs that are converted to '1' and '0', respectively.
- c) For combinational verification, input data must change far enough in advance of sensing times to allow transient delays to have settled.
- d) Clock and/or input data must change after enough time of the asynchronous set/reset signals going from active to inactive to fulfill the setup/hold times of the sequential elements in the design.

- e) For edge-sensitive designs, primary inputs of the design must change far enough in advance for the edge-sensitive storage element input data to fulfill the setup times with respect to the active clock edge. Also, the input data must remain stable for long enough to fulfill the hold times with respect to the active clock edge.
- f) For level-sensitive designs, primary inputs of the design must change far enough in advance for the level-sensitive storage element input data to fulfill the setup times. Also, the input data must remain stable for long enough to fulfill the hold times.

NOTE—A synthesis tool may define metalogical or high-impedance values appearing on primary outputs in one model as equivalent to logical values in the other model. For this reason, the input stimulus may need to reset internal storage elements to specific logical values before the outputs of both models are compared for logical values.

5.1 Combinational verification

To verify combinational logic, the input stimulus shall be applied first. Sufficient time shall be provided for the design to settle, and then the outputs examined. To verify the combinational logic portion of a model, the following sequence of events shall be done repeatedly for each input stimulus application:

- a) Apply input stimulus
- b) Wait for data to settle
- c) Check outputs

Each application of inputs shall include enough delay so that the transient delays and timeout clause delays have been exceeded. A model is not in compliance with this standard if it is possible for outputs or internal nodes of the combinational model never to reach a steady state (i.e., oscillatory behavior).

Example:

```
A <= not A after 5 ns; -- oscillatory behavior, noncompliant
```

5.2 Sequential verification

The general scheme consists of applying inputs periodically and then comparing the outputs just before the next set of inputs is applied. Sequential models contain edge-sensitive and/or level-sensitive storage elements. The sequential design must be reset, if required, before verification can begin.

The verification of designs containing edge-sensitive or level-sensitive storage elements is as follows:

- a) Edge-sensitive models: The same sequence of tasks as used for combinatorial verification shall be performed during verification: Change the inputs, compute the results, and compare the outputs. However, for sequential verification, these tasks shall be synchronized with one of the inputs, which is a clock. The inputs must change in an appropriate order with respect to the input that is treated as a clock, and their consequences must be allowed to settle prior to comparison. Comparison might best be done just before the active clock edge, and the non-clock inputs can change relatively soon after the edge. The circuit then has the rest of the clock period to compute the new results before they are stored at the next clock edge. The period of the clock generated by the stimulus shall be sufficient to allow the input and output signals to settle.
- b) Level-sensitive models: These designs are generally less predictable than edge-sensitive models due to the asynchronous nature of the signal interactions. Verification of synthesized results depends on the application. With level-sensitive storage elements, a general rule is that data inputs should be stable before enables go inactive (i.e., latch) and comparing of outputs is best done after enables are inactive (i.e., latched) and combinational delays have settled. A level-sensitive model in which it is possible, in the absence of further changes to the inputs of the model, for one or more internal values or outputs of the model never to reach a steady state (oscillatory behavior) is not in compliance with this standard.

6. Modeling hardware elements

This clause specifies styles for modeling hardware elements such as edge-sensitive storage elements, level-sensitive storage elements, three-state elements, and combinational elements.

This clause does not limit the optimizations that can be performed on a VHDL model. The scope of optimizations that may be performed by a synthesis tool depends on the tool itself. The hardware modeling styles specified in this clause do not take into account any optimizations or transformations. A specific tool may perform optimizations; this may result in removal of redundant or unused logic from the final netlist. This shall NOT be taken as a violation of this standard provided the synthesized netlist has the same functionality as the input model, as characterized in Clause 5.

6.1 Edge-sensitive sequential logic

6.1.1 Clock signal type

The allowed types for clock signals shall be BIT, STD_ULOGIC and their subtypes (e.g., STD_LOGIC). Only the values '0' and '1' from these types shall be used in expressions representing clock levels and clock edges (see 6.1.2).

Scalar elements of arrays of the above types shall be supported as clock signals.

Example:

```
signal BUS8: std_logic_vector(7 downto 0);
...
process (BUS8(0))
begin
    if BUS8(0) = '1' and BUS8(0)'EVENT then
        ...
    ...
    -- BUS8(0) is a scalar element used as a clock signal.
```

6.1.2 Clock edge specification

The general syntax for specifying an edge of a clock shall be the following:

```
clock_edge ::=
    RISING_EDGE(clk_signal_name)
  | FALLING_EDGE(clk_signal_name)
  | clock_level and event_expr
  | event_expr and clock_level
clock_level ::= clk_signal_name = '0' | clk_signal_name = '1'
event_expr  ::= clk_signal_name'EVENT | not clk_signal_name'STABLE
```

The RISING_EDGE and FALLING_EDGE functions are as declared by the package STD_LOGIC_1164 of IEEE Std 1164-1993.

6.1.2.1 Rising (positive) edge clock

The following expressions shall represent a rising edge clock:

- a) `RISING_EDGE(clk_signal_name)`
- b) `clk_signal_name = '1' and clk_signal_name'EVENT`
- c) `clk_signal_name'EVENT and clk_signal_name = '1'`
- d) `clk_signal_name = '1' and not clk_signal_name'STABLE`
- e) `not clk_signal_name'STABLE and clk_signal_name = '1'`

6.1.2.2 Falling (negative) edge clock

The following expressions shall represent a falling edge clock:

- a) `FALLING_EDGE(clk_signal_name)`
- b) `clk_signal_name = '0' and clk_signal_name'EVENT`
- c) `clk_signal_name'EVENT and clk_signal_name = '0'`
- d) `clk_signal_name = '0' and not clk_signal_name'STABLE`
- e) `not clk_signal_name'STABLE and clk_signal_name = '0'`

6.1.3 Modeling edge-sensitive storage elements

An edge-sensitive storage element may be modeled either by a signal or variable that is updated at a clock edge.

Definitions:

<sync_condition>. A <boolean_expression> with a <clock_edge> expression that only is TRUE when <clock_edge> is TRUE.

<async_condition>. A <boolean_expression> without a <clock_edge> expression.

<sync_assignment>. An assignment to a signal or variable that is controlled explicitly by <clock_edge> in all execution paths.

<async_assignment>. An assignment to a signal or variable that is not controlled by <clock_edge> in any execution path.

To illustrate these definitions, here are two examples:

Example of <async_assignment>:

```
SimpleEdgeModel: process (clk, reset)
begin
  if( rising_edge(clk) and reset = '0' ) then
    Q <= D;                                -- sync assignment
  elsif( reset = '1' ) then
    Q <= '0';                              -- async assignment
  end if;
end process;
```

In this example, the assignment `Q <= '0'` is controlled by `reset = '1'` but not by the <clock_edge> as represented by `rising_edge(clk)`. Notice that when `reset` is '1', `rising_edge(clk)` may be TRUE or FALSE; therefore, the assignment in the **elsif** is asynchronous.

Example of *<async_condition>*:

```
ComplexEdgeModel:
    process( clk, en, reset )
    begin
        if (en = '1' and rising_edge(clk))
            or (en = '1' and reset = '1') then
            if (reset = '1') then
                Q <= '0';                                -- async assignment
            elsif (en = '1' and rising_edge(clk)) then -- sync condition
                Q <= D;                                    -- sync assignment
            end if;
        end if ;
    end process ;
```

In this example, the *<sync_condition>* is the boolean expression “en = '1' and rising_edge(clk)”, because it can be true only when the clock edge also is true. The *<async_condition>* is the boolean expression “en = '1' and reset = '1'” and with reset = '1'. With these controlling the execution flow, the assignment, “Q <= '0'” is an *<async_assignment>* because it is executed when the *<async_condition>* is true, and the assignment “Q <= D” is a *<sync_assignment>* because it is executed when the *<sync_condition>* is true.

NOTE—An edge-sensitive storage element inferred for a variable may be eliminated during optimization if there exists another edge-sensitive storage element with its same functionality.

6.1.3.1 Edge-sensitive storage from a process with sensitivity list and one clock

Edge-sensitive storage shall be modeled for a signal or variable assigned inside a process with sensitivity list when all of the following apply:

- The signal or variable has a *<sync_assignment>*.
- There is no execution path in which the value update from a *<sync_assignment>* overrides the value update from an *<async_assignment>* unless the *<async_assignment>* is an assignment to itself.
- It is possible to statically enumerate all execution paths to the signal or variable assignments.
- The process sensitivity list includes the clock and any signal controlling an *<async_assignment>*.
- The *<clock_edge>* is present in the conditions only, and the *<clock_edge>* always expresses the same edge of the same clock signal.
- For a variable, the value written by a given clock edge is read during a subsequent clock edge.

NOTES

1—Except for a clock signal, signals read in a *<sync_assignment>* or signals controlling a *<sync_assignment>* are not required to be on the process sensitivity list.

2—In rule b) above, an *<async_assignment>* of a signal to itself is an exception because self-assignment retains the previous value, allowing a future, newly clocked *<sync_assignment>* value to replace a definite previous value. This specific kind of *<async_assignment>* thus merely continues the storage state previously established; it has no effect on any stored value, so overriding it makes no difference.

3—The *<clock_edge>* may be in a sequential procedure.

Example 1: Storage may be assigned in multiple statements in a process.

```
TwoReg : process(clk)
begin
  if rising_edge(clk) then
    Q1 <= D1;
    Q2 <= D2;
  end if;
end process;
```

Example 2: Multiple statements in a process, with a reset.

```
TwoRegReset : process(clk, reset)
begin
  if rising_edge(clk) then
    Q1 <= D1;
    Q2 <= D2;
  end if;

  if reset = '1' then
    Q1 <= '0';
  end if;
end process;
```

Example 3: A signal (or variable) may be updated with multiple <clock_edge> conditions on the same edge of the clock.

```
EnableEdgeProc : process(clk, reset)
begin
  if reset = '1'
  then Q <= '0';
  else
    case sel is
      when '0' => if rising_edge(clk) then Q <= D0; end if;
      when '1' => if rising_edge(clk) then Q <= D1; end if;
      when others => Q <= '0';
    end case;
  end if;
end process;
```

Example 4: More complicated multiple <clock_edge> conditions.

```
-- clk  reset  e1    e2  ||    Q
--
-- *      1      *    *  ||    0
-- rise  0      1    *  ||   D11
-- !rise 0      1    *  ||   hold  ## <clock_edge> OK as per rule b.
-- rise  0      0    1  ||   D12
-- !rise 0      0    1  ||   hold  ## <clock_edge> OK as per rule b.
```

```
multiEnableEdgeProc : process(clk, reset)
begin
  if reset = '1' then
    Q <= '0';
  elsif e1 = '1' and rising_edge(clk) then
```

```

        Q <= D11;
    elsif e2 = '1' and rising_edge(clk) then
        Q <= D12;
    end if;
end process;

```

Example 5: Async and sync assignments controlled by complicated boolean expressions.

```

RegProc5 : process ( clk, reset )
begin
    if ( (en = '1' and rising_edge(clk)) or reset = '1') then
        if ( reset = '1' ) then
            Q <= '0'; -- async assignment.
        elsif (en = '1' and rising_edge(clk)) then -- sync condition
            Q <= D; -- sync assignment
        end if;
    end if ;
end process ;

```

Incorrect Example 6: Violates rule a). Is not a <sync_assignment> because it is not controlled by a <clock_edge> in all execution paths.

```

IllegalRegProc6 : process ( clk, reset )
begin
    if ( rising_edge(clk) or reset = '1') then
        if ( reset = '1' ) then
            Q <= '0';
        else
            Q <= D;
        end if;
    end if ;
end process ;

```

Example 7: Sequential statements are allowed in a process outside the statement defining the edge-sensitive storage element(s).

```

ComboResetDFF:
    process (clock, reset1, reset2, set, async_preload, A, Q)
        variable RESET : std_logic;
    begin
        RESET := reset1 or reset2;           -- Outside the edge statement
        if RESET = '1' then
            Q <= '0';
        elsif set = '1' then
            Q <= '1';
        elsif async_preload = '1' then
            Q <= A;
        elsif rising_edge(clock) then
            Q <= D;
        end if;
        QBAR <= not Q;                       -- Outside the edge statement
    end process;

```

6.1.3.2 Edge-sensitive storage using a single wait statement

Assume the wait statement to be one of the following:

a) Wait statement with explicit clock edge:

- 1) **wait until** [<async_condition> **or**] <sync_condition>;
- 2) **wait on** <async_signals>, <clock_signal> [, <sync_signals>
 until <async_condition> **or** <sync_condition>;
- 3) **wait on** <clock_signal> [, <sync_signals>
 until <sync_condition>;

where

<async_condition> is defined in 6.1.3.

<sync_condition> is defined in 6.1.3.

<async_signals> is a sensitivity list with signals present
in the <async_condition>.

<clock_signal> is a sensitivity list with clock signal present
in the <clock_edge>.

<sync_signals> is a sensitivity list with signals present
in the <sync_condition> excluding the <clock_signal>.

<clock_edge> is defined in 6.1.2.

b) Wait statement with implicit clock edge:

- 1) **wait until** <clock_level> ;
- 2) **wait on** <clock_level_signal> **until** <clock_level_expr>;

where

<clock_level_signal> ::= sensitivity list with clock signal present
in the <clock_level>.

<clock_level> ::= *signal_name* = '0' | *signal_name* = '1'.

<clock_level_expr> ::= <boolean_expression> with <clock_level>, which
shall be TRUE only when <clock_level> is TRUE.

c) Wait statement without clock edge: This includes forms of wait statement from either item a) or item b) above in 6.1.3.2, in which the clock edge is not specified either explicitly or implicitly.

- 1) **wait on** <sensitivity_list> ;
- 2) **wait until** <condition> ;
- 3) **wait on** <sensitivity_list> **until** <condition> ;

An if statement following one of these wait statements must have <clock_edge> in the condition.

Edge-sensitive storage shall be modeled for a signal or variable, assigned inside a process with wait statement, when an assumption above is fulfilled; and, in addition:

- a) The wait statement is the first or last statement of the process
- b) The process with wait statement can be transformed to a process with “wait on <sensitivity_list>”. The resulting process with “wait on <sensitivity_list>” must adhere to the rules in 6.1.3.1.

The transformation is described as follows:

T1. A wait statement describing an implicit clock edge model [b) above] is represented as an explicit clock edge model (A above). This can be achieved by replacing “*clk_signal_name* = '0'” with *falling_edge*(*clk_signal_name*), or replacing “*clk_signal_name* = '1'” with *rising_edge*(*clk_signal_name*).

T2. A wait statement of the form, “**wait until** <condition>”, is transformed to an equivalent wait statement of the form, “**wait on** <sensitivity_list> **until** <condition>”.

T3. After these alterations, the “**wait on** <sensitivity_list> **until** <condition>” statement is transformed to an equivalent “**wait on** <sensitivity_list>” as follows:

```
process
    <process_declarative_part>
begin
    wait on <sensitivity_list> until <condition>;
    <sequence_of_statements>;
end process;
```

transforms to this equivalent process with “**wait on** <sensitivity_list>”:

```
process
    <process_declarative_part >
begin
    wait on <sensitivity_list>;
    if <condition> then
        <sequence_of_statements>;
    end if;
end process;
```

Example: Showing the transformation to wait on

```
process
begin
    wait on SET, reset, clock
        until SET = '1' or reset = '1' or rising_edge(clock);
    if reset = '1' then
        Q <= '0';
    elsif SET = '1' then
        Q <= '1';
    elsif rising_edge(clock) then
        Q <= D;
    end if;
end process;
```

Using the transformations described above, the goal is the following equivalent process with only a “**wait on** <condition>” statement:

```
process
begin
    wait on <sensitivity_list> ;
    <statement_list>
end process ;
```

This is accomplished as follows:

```
process
begin
    wait on SET, reset, clock ;
    if SET = '1' or reset = '1' or rising_edge(clock) then
        if reset = '1' then
            Q <= '0';
        elsif SET = '1' then
            Q <= '1';
        elsif rising_edge(clock) then
```

```
        Q <= D;  
    end if;  
end if;  
end process;
```

6.1.3.3 Edge-sensitive storage with one or more clocks

Multiple if statements with different clock edge conditions may be used to update a signal or variable inside a process. The process may have a sensitivity list, or it may have an equivalent **wait on** as its first or last statement.

The clock edge conditions shall be mutually exclusive.

For each clock edge expression, when the remaining clock edge expressions are replaced by FALSE in all statements of the process, the transformed process must fulfill the conditions of 6.1.3.1 or 6.1.3.2.

The signal in the first clock edge expression (textually) shall be taken as the functional clock.

NOTES

1—It is recommended to have simulation specific code enclosed within RTL_SYNTHESIS OFF/ON pragmas to check the mutual exclusivity.

2—The determination of the functional clock is made on a process-by-process basis; the intended functional clock has to be coded first in each process.

Example 1: Two different clock signals

```
library IEEE;  
use IEEE.Std_Logic_1164.all;  
  
entity multi_clock_ff_example is  
    port(reset, clk1, clk2,  
          data1, data2 : in Std_Logic;  
          Q : out Std_Logic );  
end;  
  
architecture RTL of multi_clock_ff_example is  
    begin  
        -- Process sensitive to controlling signals reset, clk1 and clk2  
        process( reset, clk1, clk2 )  
            begin  
                if reset = '1' then  
                    Q <= '0' ;  
                elsif rising_edge(clk1) then  
                    Q <= data1 ;  
                elsif rising_edge(clk2) then  
                    Q <= data2 ;  
                end if ;  
  
                -- RTL_SYNTHESIS OFF  
                if rising_edge(clk1) and rising_edge(clk2) then  
                    assert (TRUE) report  
                    "Warning: Scan and functional clock are active together"  
                    severity Warning ;  
                    Q <= 'X' ;  
                end if ;  
            end  
        end process  
    end  
end architecture
```

```

        end if ;
        -- RTL_SYNTHESIS ON

    end process;
end RTL;

```

Example 2: Two different edges of one clock signal

```

DualEdge_Proc: process (Clk, Reset) is
begin
    if Reset = '1' then
        Q <= (others => '0');
    elsif rising_edge(Clk) then
        Q <= D4Rise;
    elsif falling_edge(Clk) then
        Q <= D4Fall;
    end if;
end process DualEdge_Proc;

```

6.1.3.4 Edge-sensitive storage with multiple waits

When modeling edge-sensitive storage elements using multiple wait statements, the following rules shall apply:

- a) The wait statement shall be modeled according to 6.1.3.2, item a) or item b).

NOTE—The **wait** may reside in a sequential procedure.

- b) If one wait statement uses an <async_condition>, all wait statements shall use the same, identical <async_condition>.
- c) Each wait statement shall specify the same clock edge of a single clock.
- d) Statements under each wait statement to handle asynchronous condition (i.e., signals from <async_condition>) shall be the same.

NOTE—An **exit** or **next** following each **wait** may be used to implement a full reset of a state machine.

- e) If simulation semantics require that the value of a variable being read is written on the previous <clock_edge>, edge-sensitive storage shall be modeled for it.

Example 1: A multicycle data path element:

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity Mult is
    port (
        clk : in  std_logic;
        start : in  std_logic;
        done : out std_logic;
        A, B : in  unsigned (3 downto 0);
        Y : out unsigned (7 downto 0)
    );
end Mult;
--

```

```
Architecture ImplicitFSM of Mult is
  signal intY : unsigned(7 downto 0);
begin
  MultProc : process
  begin
    wait until clk = '1';
    if start = '1' then
      done <= '0';
      intY <= (others => '0');
      for i in A'range
        loop
          wait until clk = '1';
          if A(i) = '1' then
            intY <= (intY(6 downto 0) & '0') + B ;
          else
            intY <= (intY(6 downto 0) & '0') ;
          end if;
        end loop;
      done <= '1';
    end if;
  end process;
  Y <= intY ;      -- final state Y = A * B
end ;
```

Example 2: Asynchronous reset modeling.

```
genericStateMachineProc: process
begin
  RESET_LOOP: loop
    if reset = '1' then -- reset/init state
      Y <= '0';
      X <= '0';
    end if;

    wait until reset = '1' or rising_edge(clk);
    next RESET_LOOP when ( reset = '1' );
    X <= A; -- state one

    wait until reset = '1' or rising_edge(clk);
    next RESET_LOOP when ( reset = '1' );

    Y <= B; -- state two
    wait until reset = '1' or rising_edge(clk);
  end loop RESET_LOOP;
end process;
```

Example 3: Serial transmission modeling.

```

-----
UartTxFunction : Process
-----

begin
  TopLoop : loop
    if (nReset = '0') then
      SerialDataOut <= '1' ;
      TxRdyReg      <= '1' ;
    end if ;

    wait until nReset = '0' or
              (rising_edge(UartTxClk) and DataRdy = '1') ;
    next TopLoop when nReset = '0' ;
    SerialDataOut <= '0';
    TxRdyReg      <= '0';

    -- Send 8 Data Bits
    for i in 0 to 7 loop
      wait until nReset = '0' or rising_edge(UartTxClk) ;
      next TopLoop when nReset = '0';
      SerialDataOut <= DataReg(i) ;
      TxRdyReg      <= '0' ;
    end loop ;

    -- Send Parity Bit
    wait until nReset = '0' or rising_edge(UartTxClk) ;
    next TopLoop when nReset = '0' ;
    SerialDataOut <=
      DataReg(0) xor DataReg(1) xor DataReg(2) xor
      DataReg(3) xor DataReg(4) xor DataReg(5) xor
      DataReg(6) xor DataReg(7) ;
    TxRdyReg      <= '0';

    -- Send Stop Bit
    wait until nReset = '0' or rising_edge(UartTxClk) ;
    next TopLoop when nReset = '0';
    SerialDataOut <= '1' ;
    TxRdyReg      <= '1' ;

  end loop ;
end process ;

```

6.1.3.5 Edge-sensitive storage using concurrent signal assignment statements

A concurrent conditional signal assignment statement may be used to model an edge-sensitive storage element provided that the assignment can be mapped to a process that adheres to the rules in 6.1.3.1.

Example:

```
COND_SIG_ASSGN: Q <= '0' when RESET = '1'      else
                    '1' when SET = '1'          else
                    A  when ASYNC_LOAD = '1'    else
                    D  when CLOCK'EVENT and CLOCK = '1';
```

6.1.3.6 Edge-sensitive storage using a guarded block

A signal assigned in a guarded block shall model edge-sensitive storage if the equivalent process in the block fulfills the rules in 6.1.3.1 and the target signal is declared of kind **register**. The guard expression must be in the following form:

```
<guard_exp> ::= [<async_condition> or] <guard_sync_condition>
<guard_clk_edge> ::= not <clock_signal>'stable and <clock_signal> = '0'
                  | not <clock_signal>'stable and <clock_signal> = '1'

<guard_sync_condition> ::= A <boolean_expression> which includes
                        <guard_clk_edge> expression and which is TRUE
                        only when <guard_clk_edge> is TRUE.

<async_condition> ::= as defined in 6.1.3.
```

Example:

```
architecture GUARD1 of top is
    signal Q : std_logic register;
begin
    guardedRegBlock:
        block( set = '1' or reset = '1' or not clk'stable and clk = '1' )
        begin
            Q <= guarded '1' when set = '1' else
                '0' when reset = '1' else
                D ;
        end
        block;
    end;
end;
```

6.1.3.7 Edge-sensitive storage from a concurrent subprogram

Edge-sensitive storage shall be modeled for a signal assigned in a concurrent procedure call that can be mapped to a process adhering to the rules in 6.1.3.1.

NOTES

1—A wait in a concurrent subprogram should be used with care: Both the concurrent statement and the wait statement have sensitivity lists.

2—Recursive subprograms are supported if and only if the subprogram can be statically inlined, as required in 8.2.2.

Example:

```

architecture CONCUR_SUB of flipflop is
procedure FF
    (signal    clk,
         reset,
         D : in  std_logic;
    signal     Q : out std_logic
    ) is
begin
    if reset = '1' then
        Q <= '0';
    elsif rising_edge(clk) then
        Q <= D;
    end if;
end FF;

signal reg1, reg2, reg3 : std_logic;

begin

    FF( clk, reset,  D,   reg1 );
    FF( clk, reset, reg1, reg2 );
    FF( clk, reset, reg2, reg3 );
    FF( clk, reset, reg3,  Q  );

end;

```

6.2 Level-sensitive sequential logic

6.2.1 Modeling level-sensitive storage elements

6.2.1.1 Level-sensitive storage from process with sensitivity list

A level-sensitive storage element shall be modeled for a signal (or variable) when all the following apply:

- The signal (or variable) has an explicit assignment.
- The signal (or variable) does not have an execution path with <clock_edge> as a condition.
- There are executions of the process that do not execute an explicit assignment (via an assignment statement) to the signal (or variable).

By default, the effect of an identity assignment of the signal (or variable) shall be as though the assignment was not present.

If the combinational attribute decorates the signal (or variable), combinational logic with feedback shall be synthesized.

The process sensitivity list shall contain all signals read within the process statement.

NOTES

- 1—Variables declared in subprograms never model level-sensitive storage elements because variables declared in subprograms are always initialized in every call.
- 2—When a signal is assigned from within a procedure it shall have the same inference semantics as a signal assignment from within a process.
- 3—Recursive procedure calls are allowed if and only if the procedure can be statically inlined, as required in 8.2.2.
- 4—It is recommended to avoid a modeling style in which the value of a signal or variable is read before its assignment. This recommendation is meant to avoid the generation of unwanted storage elements.

Example 1:

```
LEV_SENS_1: process (RESET, ENABLE, D)
begin
    if RESET = '1' then
        Q <= '0';
    elsif ENABLE = '1' then
        Q <= D; -- Q is an incomplete asynchronous
    end if;    -- assignment, so it models a level-sensitive
              -- storage element.
end process;
```

Example 2:

```
-- If attribute 'combinational' is FALSE on a process,
-- identity assignment Q <= Q; causes synthesis of a latch.
-- In this example, the value TRUE causes synthesis of combinational
-- logic with feedback:

use ieee.rtl_attributes.all;           -- declaration of combinational
attribute combinational of LEV_SENS_2:label is TRUE;

LEV_SENS_2 : process ( ENABLE, D)
begin
    if ENABLE = '1' then
        Q <= D;
    else
        Q <= Q ;-- identity assignment. Same as Q <= unaffected;
    end if;
end process;
```

Example 3:

```
-- A process modeling both latch and flip-flop is supported.
RegPlusLatProc: process(clk, reset, gEnable)
    variable gLatch : std_logic;
begin
    if clk = '0' then
        gLatch := gEnable;
    end if;
    if reset = '1' then
        Q <= '0';
    elsif gLatch = '1' and rising_edge(clk) then
        Q <= D;
    end if;
end process;
```

Example 4:

-- Again, if attribute 'combinational' is FALSE on a process,
 -- identity assignment 'Q_TEMP := Q_TEMP;' is replaced by null statement.

```

use ieee.rtl_attributes.all;
attribute combinational of LEV_SENS_4:label is FALSE ;

LEV_SENS_4 : process ( ENABLE , D)
  variable Q_TEMP : BIT ;
  begin
    if ENABLE = '1' then
      Q_TEMP := D;
    else
      Q_TEMP := Q_TEMP ; -- identity assignment
    end if;
    Q <= Q_TEMP ;
  end process;

```

Example 5: Inferred latch, perhaps as part of a scan chain

-- Signal Q has an initial value in its declaration
 LEV_SENS_5 : **process** (enable, Q)
begin
 if enable = '1' **then**
 Q <= Q ;
 end if;
end process;

Example 6:

-- Ram element as level-sensitive storage is supported.

```

RAM_WRITE : process (WDE,D,ADDR)
  begin
    if WDE = '1' then
      myMem( ADDR ) <= D;
    end if;
  end process;
Q <= myMem(ADDR);

```

6.2.1.2 Level-sensitive storage from concurrent signal assignment

A level-sensitive storage element shall be modeled for a signal that is assigned in a concurrent signal assignment statement that can be mapped to a process that adheres to the rules in 6.2.1.1.

Example 1:

```

LEV_SENS_7: Q <= '0' when RESET='1' else -- This is identical
           D when ENABLE;           -- to LEV_SENS_1 in 6.2.1.1,
                                     above.

```

Example 2:

```

LEV_SENS_8: With ENABLE select
  Q <= D when '1',
  Q when others; --Identical to LEV_SENS_2 in 6.2.1.1,
                -- and models combinational logic.

```

Example 3:

```
LEV_SENS_9: with ENABLE select
    Q <= D      when '1',
    unaffected  when others;
```

6.2.1.3 Level-sensitive storage from concurrent procedure call

A level-sensitive storage element shall be modeled for a signal assigned in a concurrent procedure call that can be mapped to a process that adheres to the rules in 6.2.1.1.

Example:

```
architecture CONCUR_SUB of level_sensitive
is
  procedure latch(signal ENABLE, D :in bit;
    signal Q :out bit ) is
    begin
      if ENABLE = '1' then Q <= D;
      end if;
    end;
  signal Q : bit;
begin
  latch(a,sel,Q); --Concurrent procedure call which models a latch.
end CONCUR_SUB;
```

6.2.1.4 Level-sensitive storage from guarded block

A level-sensitive storage element shall be modeled for a signal assigned in a guarded block that can be mapped to a process fulfilling the rules in 6.2.1.1. The signal must be of kind **register** and a <clock_edge> expression must not be used in the guard expression.

Each concurrent guarded signal assignment statement within such a guarded block must be equivalent to a process statement fulfilling the rules in 6.2.1.1.

Example:

```
entity latchEntity is
  port ( clk1, dat1,
    dat2 : in Std_Logic;
    q1, q2 : out Std_Logic );
end
entity latchEntity;

architecture latchArch of latchEntity is
  signal gq1, gq2 : Std_Logic register;
begin
  dlLatch:
    block(clk1 = '1')
    begin
      gq1 <= guarded dat1;
      gq2 <= guarded dat1 and not dat2;
    end block
end latchArch;
```