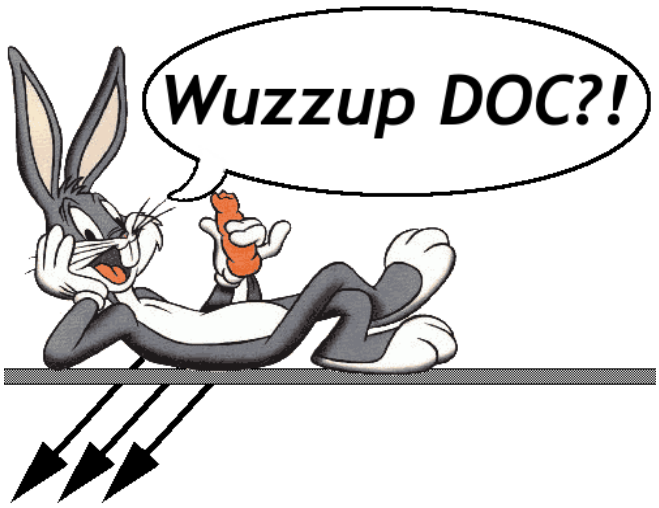


Requirement Specification

Author: Daniel Hilding

Version: 2.1

Date: 2005-12-05

Abstract

This document describes the requirements of the Verilog documentation tool that PUM group 12 will develop for the department of electrical engineering at Linköping University. The document gives an overview of the system and will be used as a reference during the development. It also includes delivery terms and acceptance tests that will be important when evaluating the system.

Project identity

Project group

Wuzzup DOC
PUM 12 2005
Linköping tekniska högskola
Institutionen för datavetenskap (IDA)

Project members

Name	Area of responsibility	Telephone	E-mail
Martin Jormedal	Project leader (PL)	073-3121319	ook4mi@gmail.com
Daniel Hilding	Customer relations manager (CRM)	070-7440440	danhi139@student.liu.se
Joakim Svartengren	Documentation manager (DOC)	070-4040005	joasv190@student.liu.se
Jonas Norling	Design manager (DES)	070-3904809	norling@lysator.liu.se
Johan Lissing	Implementation manager (IM)	073-9036256	johli650@student.liu.se
Thobias Bergqvist	Quality manager (QM)	073-6223040	thobe651@student.liu.se
Eric Åberg	Test manager (TM)	070-4058130	eriab522@student.liu.se

Mailinglist for group

pum12@und.ida.liu.se

Web page

http://www-und.ida.liu.se/~pum12/

Customer

Per Karlström, Computer Engineering, ISY LiU

Customer contact person

Per Karlström, 013-28 29 03, perk@isy.liu.se

Project supervisor

David Broman, 013-28 57 24, davbr@ida.liu.se

Examiner

Robert Kaminski, 013-28 24 57, robka@ida.liu.se

Document History

Date	Version	Changes	Name
2005-09-15	0.1	Document created	Daniel Hilding
2005-09-16	0.2	Document revised after review, spelling changed.	Daniel Hilding
2005-09-27	1.0	Changed in project description to make it clearer how Doxygen will be a part of our product.	Daniel Hilding
2005-10-03	1.1	Changed all references to non-functional requirements from "I" to "N".	Daniel Hilding
2005-10-06	1.2	Moved F-6 to extra requirements. Added execution to test id 8.	Daniel Hilding
2005-10-06	2.0	Changed background. Added glossary. Changed use case and users. Added source of the requirement in the requirements motivation. Extended specification for functional requirements. Moved protocol for acceptance of product components to a new appendix. Specified the acceptance tests.	Daniel Hilding
2005-12-05	2.1	Changed F-1. Added F-12 and F-13. Added test case 17 and 18	Daniel Hilding

1 Introduction

This chapter helps the reader to get a quick overview of the content of this document and its structure.

1.1 Purpose

The purpose of this document is to specify the requirements for the system that is to be developed by PUM-group 12. It describes all requirements, functional and non-functional. It also describes how they are tested. The document will also define the terms of delivery.

1.2 Chapter description

Below is an overview and a brief description of the chapters in this document.

1. Introduction

   Gives an overview of the content and the chapters in the document. This chapter also contains reading instructions and a glossary for the requirement specification.
2. Project description

   This chapter describes the system, its users, and the environment it is developed for. The project description also contains the involved parties, the background and a system description describing the overall goals of the project.
3. Use cases

   An overview of the use cases that describes the basic functionality of the system.
4. Design of requirements

   This chapter describes the notation and design of the requirements.
5. Functional requirements

   This chapter describes the functional requirements of the system.
6. Non-functional requirements

   A description of the non-functional requirements of the system.
7. Product components

   An overview of the components that are included in the product.
8. Delivery terms

   This chapter states the terms for delivery of the product.
9. Acceptance test specification

   This chapter states the terms for the customer to accept the product.
10. References

    An overview of the references used in this document.
11. Appendix A. Protocol, delivered product components

    Protocol over delivered product components.
12. Appendix B. Test cases

    Protocol for testing the requirements.
13. Appendix C. Contract

Contract where the customer and the developer accepts the requirement specification.

1.3 Reading instructions

To quickly get an overview of the project it is recommended to read chapter 2, the project description. This gives a brief background and an explanation of the project objectives. It is also recommended to take a look at the use cases in chapter 3 as they give an overview of the basic functionality of the system.

For more detailed information about the system the reader is directed to chapters 5 and 6, the functional and non-functional requirements of the system. These are the most important chapters in this document since they define the actual system that will be developed. To fully be able to appreciate these chapters it is recommended to first read the glossary in section 1.5 and the design of requirements in chapter 4.

Product components and delivery terms are found, respectively, in chapters 7 and 8. This is important reading for the customer.

1.4 Document dependencies

Changes in this requirement specification might result in changes in the following documents:

- Project plan [Jormedal, 2005]
- Architecture specification [Norling, 2005]
- Design specification [Lissing, 2005]
- Technical documentation [Lissing, 2005]
- Test plan [Åberg, 2005]

1.5 Distribution

This document should be distributed to:

- David Broman and Angela Johansson, examiners of the requirements specification.
- Per Karlström, the customer.
- The project folder.

1.6 Glossary

Parser -- A piece of software that determines the syntactic structure of a language

Flex -- The Gnu lexical analyzer generator (a variant of lex). See [Flex].

Doxygen -- Documentation tool that generates documentation from source code files from different programming languages such as C++, C, Java and IDL. The documentation is presented several output formats such as HTML and LaTeX. See [Doxygen].

Verilog -- Verilog is a common hardware design language (HDL) used by electrical engineers worldwide and specifically by the customer. Structu-

rally, it resembles normal computer programming languages but naturally it has some special features and data structures, such as modules, wires and so on. See [Verilog].

2 Project description

This chapter describes the system, its users, and the environment it is developed for. The project description also contains a general description, the involved parties, the background and the goals of the project.

2.1 Parties

The parties involved in the project are the customer and the developer of the software. The customer is the subdepartment Computer Engineering of the department of Electrical Engineering (ISY) at Linköping university. Per Karlström is our contact person at the department. Developer of the system are PUM-group 12, Wuzzup DOC, with contact person Daniel Hilding, customer relationships.

2.2 Background

The customer have used documentation tools when programming in C++ and realized the benefits of such a tool. A software tool of this kind helps keeping the documentation in synchronization with the actual implementation. This is of great value when several persons are working on the same project and must be informed about each other's work. It also saves a lot of time when the documentation document is created by a tool.

While there are several documentation tools for the most popular computer programming languages, such as C++ and Java, there is no tool capable of documenting Verilog [Verilog] code.

Today the customer uses Verilog and since it does not exist any documentation tool for Verilog this is what the customer wants.

2.3 System description

Our product is not meant to be a complete and usable tool, but merely a foundation for such. The final product, should it ever be developed, should have the same base functionality as Doxygen [Doxygen]. The program will be developed as an extension to Doxygen.

2.4 Goal of project

This project will probably not have sufficient resources to produce a complete Verilog documentation tool. The goal with this project is therefore to make a foundation for a documentation tool and to investigate whether this could be done using Doxygen.

All findings, research and/or source code, will be extensively documented for future projects on the same topic.

2.5 Users and environment

The final product is targeted at employees at the department of electrical engineering, Linköping University, with programming experience. The program will be run at their computer system which is mainly UNIX and GNU/Linux.

2.6 Assumptions

Since we have a good and regular contact with the customer, we have discussed any uncertainties directly with him. For that reason we haven't needed to make any assumptions.

3 Use cases

This chapter covers the basic use cases of the system. The purpose of these use cases is to capture high-level and user-oriented requirements of the system, as well as to give the reader of this document a quick overview of the basic functionality of the system.

3.1 User category

The program's only user category is Verilog programmers. The program is especially designed for employees at the department of electrical engineering, Linköping University.

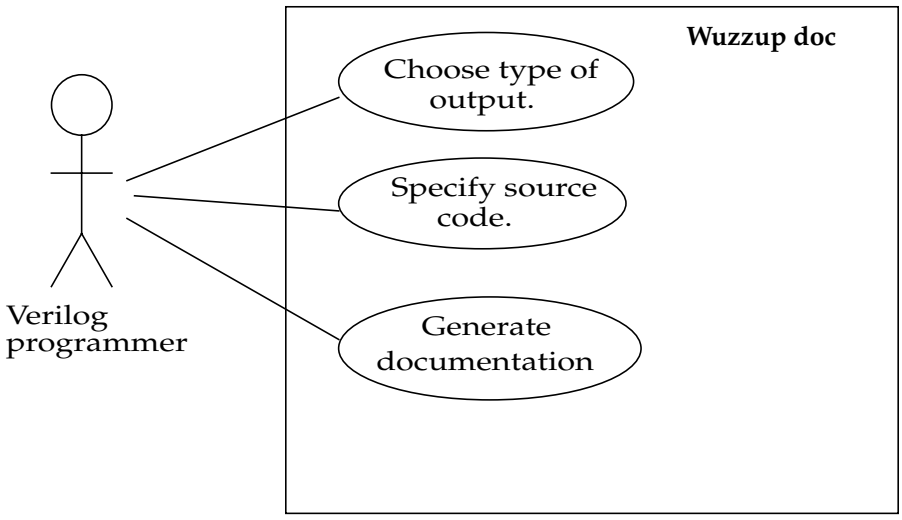
3.2 User environment

The program will be used on the computers at ISY, Linköping University.

ISY uses Linux on their computers. The software will primarily be made for Linux. An extra requirement is that it could be used on windows computers too.

3.3 Use case diagram

Figure 1 is a use case diagram that visualizes the functionality of the system.



Figur 1: Use case diagram that shows how the tool will be used.(UML 1.5 notation)

3.3.1 Details of use case

A programmer wants documentation for the Verilog code he has written. The final program should be able to output several output formats, e.g. HTML, LaTeX, for that reason the programmer needs to specify output first.

The program must know what code the tool should document, so the programmer must specify his source files.

Finally the programmer wants the tool to generate documentation for the chosen source file.

4 Design of requirements

This chapter describes how the requirements in this specification are designed. The chapter explains how to read and interpret the information and how to document possible changes.

4.1 Notation

The requirements are structured as in the following example:

Level: The level of the requirement. See section 4.3.
Status: The status of the requirement. See section 4.4.
Motivation: An explanation of why the requirement has been introduced. See section 4.5.
Influences: A list of numbers for requirements that are influenced by this particular requirement. See section 4.6.

4.2 Identification number

Every requirement has been assigned a unique identification number to make it possible to refer to a specific requirement. This enables tracing of certain requirements throughout the whole project. The identification number consists of a capital letter followed by a number, for example F-2. The letter shows what type of requirement it is; functional requirement (F), non-functional requirement (N), or product component (C).

From version 1.0 and on of this document, the requirement identification numbers are permanent and can not be changed. A new requirement will be placed in the appropriate subsection and get the next free identification number, according to the standard described above.

4.3 Level

All requirements have been divided into three levels of priority: basic, normal and extra. The levels will be fulfilled in this order, starting with the basic requirements.

- Basic

  This level has the highest priority and is fundamental for the functionality of the system. All requirements on this level must be fulfilled.
- Normal

  Requirements on this level should be fulfilled to achieve satisfying functionality in the system. This is also the level that the project group calculates to be able to reach, but in case a situation arises where the project runs out of time, it can not be guaranteed that this level is fulfilled since the project has a strict time limit.
- Extra

  This level contains extra features that enhance the functionality of the system. If the work progresses faster than expected, there might be time left to implement these requirements. Otherwise they should be seen as possible future developments that are not included in the design, but are still taken into account during development.

4.4 Status

Every requirement has a description of its status to increase traceability. The status will tell if the requirement is valid or not.

- Active

  Active means that the requirement is valid and included in the development of the system.

- Inactive

  Inactive means that the requirement is not valid, but it is still included in the specification to increase traceability. The requirement will keep its identification number but will not be included in the development of the system. This could happen if the developer together with the customer decides that a requirement should not be involved in the product anymore.

4.5 Motivation

Every requirement has a short motivation of why and by who/what it has been introduced. The motivation gives the reader a better understanding of the purpose of the requirement.

4.6 Influences

If a requirement influences other requirements, in a way that a change in the first makes it necessary to change the others, they are listed with their identification numbers. This increases traceability during development. If a requirement does not have a significant influence on any other requirement this line will be left blank.

5 Functional requirements

This section describes the functional requirements that apply to the system. Functional requirements not mentioned here can not be expected to be included in the delivered system. New requirements that occur after version 1.0 is added last.

5.1 Basic requirements

F-1 Create a parser for Verilog code

The tool should be able to parse code written in the Verilog -95 language, skipping comments (see requirement F-6). The Verilog parser should output a syntax tree as described in requirement F-2. The parser does not have to preprocess the Verilog source and may ignore compiler directives. If the parser is not compatible to some syntax that shold be documented in the Technical documentation [Lissing, 2005].

Level: Basic
Status: Active
Motivation: Verilog -95 is the customer's language of choice. The initial step towards a documentation tool is a parser.
Influences: -

F-2 Create a syntax tree

The output from the parser should be a syntax tree, using an arbitrary data type, for each parsed file. The syntax tree should contain nodes for all constructs in the Verilog source file that are important to the generation of documentation (this includes module declarations, module instantiations, wires and registers).

Level: Basic
Status: Active
Motivation: The syntax tree serves as an intermediate representation of data that is fed to the part of the program that is responsible for output generation. To ease implementation of requirement F-5, the arbitrary data type should be similar to the tree format that Doxygen uses.
Influences: F-1

F-3 Print the syntax tree

The syntax tree should be printed in a readable form.

Level: Basic
Status: Active
Motivation: This output is needed to verify the correctness of the parsing. It is required for debugging by the implementation team and for testing by the testing team.
Influences: F-2

F-4 Linux compatible

The product should be run in Linux.

Level: Basic
Status: Active
Motivation: A demand from the customer. The customer uses the Linux platform.
Influences: F-1

5.2 Normal requirements

F-5 Doxygen compatible syntax tree

The syntax tree should be compatible with Doxygen's data organizer, i.e. use Doxygen's Entry class for tree nodes.

Level: Normal
Status: Active
Motivation: This will allow the tool to be made a part of Doxygen, benefiting from its features. This a request from the customer.
Influences: F-2

F-6 Extract comments

The program should extract comment blocks with documentation regarding modules and variables in the Verilog code. The comments should be connected to the structure that follows after them, similar to how Doxygen works with comments in C++ code.

Level: Normal
Status: Active
Motivation: Makes a better documentation of the Verilog code. A request from the customer.
Influences: F-1, F-2, F-9, F-11

5.3 Extra Requirements

F-7 Extract a hierarchy

A hierarchy over the system described by the Verilog code should be extracted. The hierarchy should show module instantiations within modules and source files.

Level: Extra
Status: Active
Motivation: Gives a good overview of the code. Necessary for Doxygen output. A request from the customer.
Influences: F-9, F-11

F-8 Custom directives

The parser should be able to recognize custom directives in the Verilog code comments, for example defining the author of the code, known bugs, to-do lists, etc. The supported directives are: author, date, bug, clock (specifying a clock signal), reset (specifying a reset signal), comb (stating that an always block is purely combinatorial), state (describing a state in a state machine), class (grouping of signals).

Level: Extra
Status: Active
Motivation: Custom directives can be used for extra functionality. A request from the customer.
Influences: F-6

F-9 Generate HTML

The product should be able to output the hierarchy and the comments from the Verilog code in HTML.

Level: Extra
Status: Active
Motivation: HTML is a popular and versatile format. Both the group and the customer feels like it is the easiest and best output format.
Influences: -

F-10 Windows compatible

The tool should be Windows XP compatible.

Level: Extra
Status: Active
Motivation: Windows XP is a popular platform. A request from the customer.
Influences: -

F-11 Other outputs

The product should be able to output the hierarchy and the comments from the Verilog code in LaTeX.

Level: Extra
Status: Active
Motivation: LaTeX is common among academics. A request from the customer.
Influences: -

5.4 New Normal Requirements

F-12 Verilog preprocessing

Compiler directives in the Verilog source files should be preprocessed before the files are parsed.

Level: Normal
Status: Active
Motivation: Compiler directives are common in Verilog to provide an easy way of changing the structure or behavior of the source code. The directives must be preprocessed in order to make

the documentation properly reflect the actual source code.

Influences: F-1, F-13

5.5 New Extra Requirements

F-13 Verilog-2001 compatible parser

The tool should be able to parse code written in the Verilog -2001 language, skipping comments (see requirement F-6). The Verilog parser should output a syntax tree as described in requirement F-2. The parser does not have to preprocess the Verilog source and may ignore compiler directives.

Level: Extra
Status: Active
Motivation: Verilog -2001 is an extended version of Verilog -95. The initial step towards a documentation tool is a parser.
Influences: -

6 Non-functional requirements

This section describes the non-functional requirements that apply to the system. Non-functional requirements not mentioned here can not be expected to be included in the delivered system.

6.1 Basic requirements

N-1 Programming language

All code should be written with C++ and Flex.

Level: Basic
Status: Active
Motivation: C++ makes the program easy to expand since it's very well known. The Doxygen parser is made with Flex. C++ is a request from the customer.
Influences: F-1

N-2 Documented with Doxygen

All C++ code in the product should be documented with Doxygen.

Level: Basic
Status: Active
Motivation: Makes it easy to get an overview of our tool. A request from the customer.
Influences: N-1, F-1

N-3 Upgradeable

The program should be easy to extend and expand.

Level: Basic
Status: Active
Motivation: Future upgrade may be wanted for more functionality. A request from the customer.
Influences: N-4, F-1

N-4 English usage

Source code, user interface and documentation should be in English.

Level: Basic
Status: Active
Motivation: The tool may be used world wide. A request from the customer.
Influences: N-1, N-4, F-1

N-5 Documentation format

All documents should be delivered in PDF- and FrameMaker-format.

Level: Basic
Status: Active
Motivation: PDF is a portable format. The use of FrameMaker is

mandatory in the project course.

Influences: C-2, C-3, C-4, C-5

N-6 Revision control

All code should be handled with Subversion.

Level: Basic
Status: Active
Motivation: Helps the developer not to mix up different versions of code. A request from the customer.
Influences: -

7 Product components

This section describes all the product components that will be part of the system and delivered to the customer. Nothing but what is mentioned here can be expected to be delivered.

7.1 Software

This section describes all the software components that should be delivered to the customer.

C-1 Source-code

All produced source-code should be delivered to the customer.

Level: Basic
Status: Active
Motivation: Enables future development. A request from the customer.
Influences: -

7.2 Documentation

This section describes all documentation that should be delivered to the customer.

C-2 Requirements specification

The requirements specification document should be delivered to the customer.

Level: Basic
Status: Active
Motivation: The customer gets an opportunity to verify that the constructed software fulfills the agreed requirements. A request from the customer.
Influences: -

C-3 Architectural design

The architectural design document should be delivered to the customer.

Level: Basic
Status: Active
Motivation: Simplifies future development. A request from the customer.
Influences: -

C-4 Design specification

The detailed design document should be delivered to the customer.

Level: Basic
Status: Active
Motivation: Simplifies future development. A request from the customer.
Influences: -

C-5 Technical documentation

An extensive technical documentation over the product's code should be delivered to the customer.

Level: Basic
Status: Active
Motivation: Simplifies future development. A request from the customer.
Influences: -

8 Terms of delivery

This section describes the terms for the delivery of the finished product.

8.1 Time

The exact time of delivery is not yet decided, but the product should be delivered to the customer no later than 2005-12-16.

8.2 Place

The product should be delivered to the customer at ISY, Linköping University.

8.3 Report

At delivery, a report will be written to ensure that the product fulfills all that is promised.

8.4 Demonstration

A short demonstration of the product will be performed at delivery.

8.5 Installation

At delivery the product will be installed on the hardware that the customer provides us with.

8.6 Maintenance

No support or maintenance is included in the agreement with the customer.

9 Acceptance test specification

This chapter describes what tests to be conducted and in which order to run them. This ensures that the acceptance tests will run smoothly.

9.1 Acceptance test

Every test has its own table similar to the one below.

The test id is a number starting at 1 and increases with each test. This is also the test order.

All the test cases can be found in Appendix A Test cases.

Test id	Identification of the acceptance test.
Purpose	Describes why the test is being done.
Req.	The corresponding requirement
Req. level	The level of the corresponding requirement.
Data input	The data that has to be inserted into the system to be able to carry out the test.
Execution	Describes how the test should be carried out.
Expected result	If the test result is the same as the expected result, the test has passed.
Passed	The customer should sign here if the test passed the requirement.

Table 11: Test table overview

9.2 Acceptance terms

When all the tests of basic requirement level have been passed, the product should have met the customer's requirements and be considered accepted.

10 References

10.1 Internal documents

[Jormedal] Jormedal, Martin (2005), "Project plan". Linköping, 2005.

[Norling] Norling, Jonas (2005), "Architecture specification". Linköping, 2005.

[Lissing] Lissing, Johan (2005), "Design specification". Linköping, 2005.

[Lissing] Lissing, Johan (2005), "Technical documentation". Linköping, 2005.

[Åberg] Åberg, Eric (2005), "Test plan". Linköping, 2005.

10.2 External documents

- [Flex] Free Software Foundation,
  WWW: http://www.gnu.org/software/flex/
- [Verilog] McNamara, Michael, WWW: http://www.verilog.com/
- [Doxygen] van Heesch, Dimitri, WWW: http://www.doxygen.org/

Appendix A Protocol delivered product components

This appendix lists the components that should be delivered to the customer. The customer should go through this protocol and sign every delivered component. All components must be delivered if the delivery should be approved.

Component id	Description	Signature
C-1	Source-code	
C-2	Requierments specification	
C-3	Architectural design	
C-4	Design specification	
C-5	Technical documentation	

Table 12:

Appendix B Test cases

This document lists all requirements listed. At delivery the testes will be run and the customer will sign those test cases that are fulfilled. All tests at basic level must be accepted and signed for approved delivery. Tests of new requirements are added last.

Test id	1
Purpose	To verify that the program is written in C++ and Flex.
Req.	N-1
Req. level	Basic
Data input	-
Execution	Go through the code and examine if it is written in C++ and Flex.
Expected result	The code is written in C++, Flex and Bison.
Passed	

Table 13: Test case 1

Test id	2
Purpose	To verify that the program is documented in Doxygen.
Req.	N-2
Req. level	Basic
Data input	-
Execution	Go through the comments and examine if it is commented with Doxygen tags.
Expected result	The code is commented with Doxygen.
Passed	

Table 14: Test case 2

Test id	3
Purpose	To verify that the program is easy to upgrade in the future.
Req.	N-3

Req. level	Basic
Data input	-
Execution	If the customer does not think that the program is easy to upgrade an objective person with good knowledge in computer science will decide if the program is easy to upgrade. Both the customer and the developer should accept this person.
Expected result	The customer thinks that the program is easy to upgrade
Passed	

Table 15: Test case 3

Test id	4
Purpose	To verify that the program's code and user interface as well as the documentation are all written in English.
Req.	N-5
Req. level	Basic
Data input	-
Execution	Go through the code and documents and see if they are in English.
Expected result	The code and documentation are written in English.
Passed	

Table 16: Test case 4

Test id	5
Purpose	To verify that the program's code is handled with Subversion.
Req.	N-6
Req. level	Basic
Data input	-
Execution	Check the Subversion history for existing versions.
Expected result	The Subversion history contains several versions of code.
Passed	

Table 17: Test case 5

Test id	6
Purpose	To verify that the program has the ability to parse Verilog -95 code.
Req.	F-1
Req. level	Basic
Data input	Verilog file only containing Verilog -95 code.
Execution	Execute the program.
Expected result	The program executes without errors.
Passed	

Table 18: Test case 6

Test id	7
Purpose	To verify that the program can produce a syntax tree from a Verilog code file.
Req.	F-2
Req. level	Basic
Data input	Verilog file.
Execution	Execute the program.
Expected result	The program produces a syntax tree that contain nodes for all constructs in the Verilog source file that are important to the generation of documentation.
Passed	

Table 19: Test case 7

Test id	8
Purpose	To verify that the program prints the syntax tree in a readable form.
Req.	F-3
Req. level	Basic
Data input	Verilog file.
Execution	Execute the program.

Expected result	The program prints a readable syntax tree that contain nodes for all constructs in the Verilog source file that are important to the generation of documentation.
Passed	

Table 20: Test case 8

Test id	9
Purpose	To verify that the program can be run in Linux.
Req.	F-4
Req. level	Basic
Data input	Verilog file.
Execution	Execute the program in Linux environment.
Expected result	The program executes without errors.
Passed	

Table 21: Test case 9

Test id	10
Purpose	To verify that the syntax tree is compatible with Doxygen's data organizer.
Req.	F-5
Req. level	Normal
Data input	Verilog file
Execution	Implement Doxygen's data organizer with the projects parser and execute the program.
Expected result	The program runs without errors and produces a number of lists and tables that corresponds to the syntax tree.
Passed	

Table 22: Test case 10

Test id	11
Purpose	To verify that the program extracts comments regarding modules and variables in the Verilog code.

Req.	F-6
Req. level	Normal
Data input	Verilog file.
Execution	Execute the program.
Expected result	The program extracts comments regarding modules and variables similar to Doxygen.
Passed	

Table 23: Test case 11

Test id	12
Purpose	To verify that the program extracts a hierarchy over the Verilog code.
Req.	F-7
Req. level	Extra
Data input	Verilog file.
Execution	Execute the program.
Expected result	A hierarchy is extracted showing module instantiations within modules and source files.
Passed	

Table 24: Test case 12

Test id	13
Purpose	To verify that the program can recognize custom directives in the Verilog code comments.
Req.	F-8
Req. level	Extra
Data input	Verilog file.
Execution	Execute the program.
Expected result	The program recognizes custom directives: author, date, bug, clock, reset, comb, state, class.
Passed	

Table 25: Test case 13

Test id	14
Purpose	To verify that the program can generate output in HTML.
Req.	F-9
Req. level	Extra
Data input	Verilog file.
Execution	Execute the program.
Expected result	A HTML documentation with the Verilog code's hierarchy and comments is generated.
Passed	

Table 26: Test case 14

Test id	15
Purpose	To verify that the program is compatible with Windows XP.
Req.	F-10
Req. level	Extra
Data input	Verilog file.
Execution	Execute the program in a Windows environment.
Expected result	The program executes without errors.
Passed	

Table 27: Test case 15

Test id	16
Purpose	To verify that the program can generate LaTeX output.
Req.	F-11
Req. level	Extra
Data input	Verilog file.
Execution	Execute the program

Expected result	A LaTeX file with the Verilog code's hierarchy and comments is produced.
Passed	

Table 28: Test case 16

Test id	17
Purpose	To verify that the Verilog source files are preprocessed before the files are parsed.
Req.	F-12
Req. level	Normal
Data input	Verilog file containing ‘define, ‘if and ‘ifdef.
Execution	Execute the program.
Expected result	The program should write out chosen codeblocks.
Passed	

Table 29: Test case 17

Test id	18
Purpose	To verify that the program has the ability to parse Verilog -2001 code.
Req.	F-13
Req. level	Extra
Data input	Verilog file containing Verilog -2001 code.
Execution	Execute the program.
Expected result	The program executes without errors.
Passed	

Table 30: Test case 18

Appendix C Contract

This document defines the terms and conditions for the product that PUM group 12 will develop for the customer. The product is developed within the limits of the course TDDC02 - Software Engineering Project at Linköping University.

C.1 Parties

The parties involved in this contract are:

- Subdepartment Computer Engineering of the department of Electrical Engineering, represented by Per Karlström, from now on referenced as "the customer".
- PUM group 12, from now on referenced as "the supplier", represented by Daniel Hilding.

C.2 Contract specification

This contract specification includes decided delivery terms and the requirements from the requirements specification. Requirements on the basic level shall be met in the final product. Requirements on the normal level should be met if the project follows its planned schedule. The requirements on the extra level will be met if there is time over after the basic and normal requirements are fulfilled. If the supplier and the customer have different interpretations of a specific requirement, there will be a renegotiation of this requirement.

C.3 Delivery

Terms for the delivery are specified in chapter 8, signing this contract implies acceptance of these terms.

C.4 Acceptance

Chapter 9 includes additional terms, regarding the terms of acceptance. Signing this contract implies acceptance of these terms.

C.5 Signature

This contract is hereby approved by customer and supplier.

Location.....Date.....

Signature.....

Per Karlström, Department of Electrical Engineering, Linköping University.

Location.....Date.....

Signature.....

Daniel Hilding, customer relations manager, PUM group 12.