



FP6-IST-507219

# PROSYD

Property-Based System Design

Instrument: Specific Targeted Research Project

Thematic Priority: Information Society Technologies

## Support for Embedded PSL in Verilog Flavour - Static Checking

(Deliverable 3.2/15)

Due date of deliverable: March 31, 2006

Actual Delivery date: March 30, 2006

Start date of project: 01.01.2004

Duration: 3 years

Organisation name of lead contractor for this deliverable: IBM

Revision: 1.0

Project co-funded by the European Commission within the Sixth Framework Programme (2000-2006)		
Dissemination Level		
PU	Public	<input checked="" type="checkbox"/>
PP	Restricted to other programme participants (including the Commission Services)	<input type="checkbox"/>
RE	Restricted to a group specified by the consortium (including the Commission Services)	<input type="checkbox"/>
CO	Confidential, only for members of the consortium (including the Commission Services)	<input type="checkbox"/>

## Notices

For information, contact [pidan1@il.ibm.com](mailto:pidan1@il.ibm.com)

This document is intended to fulfil the obligations of the PROSYD project concerning deliverable 3.2/15, described in contract number 507219.

The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

© Copyright PROSYD 2006. All rights reserved.

## Table of Revisions

Version	Date	Description and Reason	By	Affected Sections
0.1	15 February 2006	First draft by authors	Ziv Nevo, Michael Shamis, Dmitry Pidan	all
0.2	28 February 2006	Revised version	Dmitry Pidan	all
0.3	14 March 2006	Revised after project manager review	Dmitry Pidan	3,4
1.0	30 March 2006	Format features list	Cindy Eisner	3

## Authors

Ziv Nevo

Michael Shamis

Dmitry Pidan

## Executive Summary

One of the challenges verification engineers must address while doing assertion-based verification is attaching behaviour-checking properties to their correct context in the design.

The RuleBase PE static verification tool already supports one method of attaching properties to their context by binding the verification units. The purpose of this deliverable is to provide another method of attaching properties to their context in the design by writing them down where they should be attached—in other words, writing *embedded assertions*.

## Purpose

The purpose of this document is to describe the implementation of an extension made to RuleBase PE in the framework of Deliverable 3.2/15.

## Intended Audience

This document is intended for users of IBM's RuleBase PE tool

## Background

RuleBase PE is a static verification tool that receives PSL assertions as input. Assertions can be written in two flavours: GDL and Verilog. In general, design developers prefer using the PSL flavour that matches the original language of the design under verification.

Assertions are provided to RuleBase PE in a separate file. Various PSL language constructs allow mapping the assertions into their appropriate context. However, as can be seen from a variety of EDA tools that support PSL, many verification and (especially) design engineers prefer embedding their assertions directly in their RTL code. Deliverable 3.2/15 extends RuleBase PE with the ability to gather PSL Verilog flavour assertions embedded in the RTL code and perform a static analysis using them.



# Contents

Table of Revisions .....	iii
Authors .....	iii
Executive Summary .....	iii
Purpose .....	iii
Intended Audience .....	iii
Background.....	iii
List of Tables .....	v
Glossary.....	vi
1 Introduction.....	1
2 Embedded PSL Assertions .....	2
Syntax.....	2
Implementation .....	3
User Interface.....	3
Running a Vunit Verification Session.....	4
Using the Rule List.....	4
Refreshing the Rule List .....	5
Displaying Rule Window for Vunits.....	5
Running a Basic Vunit Verification Session .....	5
Stopping a Vunit Verification Session .....	5
Bound Vunits and <i>Embedded Assertions</i> (Verilog Flavour) .....	6
3 Tool Properties and Development.....	8
Features List.....	8
Work summary.....	9
Starting point.....	9
Work performed .....	9
Results .....	9
Supported PSL subset.....	9
4 References.....	10

## List of Tables

Table 1 List of features.....	8
-------------------------------	---

# Glossary

## **Assertion**

A property that is expected to hold on a specific design.

## **Embedded assertion**

Assertion that appear in the body of the design implementation code.

## **HDL (Hardware Description Language)**

One of several specialized high-level languages used by semiconductor designers to describe the features and functionality of chips and systems prior to handoff to the IC layout process. HDL descriptions are used in both the design implementation and verification flows. Currently, the two standard HDLs in use worldwide are Verilog HDL and VHDL. Several proprietary HDLs also exist, mainly for describing logic targeted for vendor-specific programmable logic devices.

## **HDL concurrent statement (assignment)**

An HDL statement that is executed concurrently with other statements in the block. For the assignments, the value assigned to the left hand side of the assignment is considered to be valid at the next time point. See also [2].

## **Static verification**

The automatic or almost automatic verification of a property of a model of a hardware component or software.

## **RuleBase PE**

An IBM tool for performing static verification.

## **Vunit**

Verification unit – a PSL construct that can contain a number of assertions and additional modelling code.

# 1 Introduction

Static verification is an important part of the full assertion-based verification flow. As with many other assertion-based methods, one of the most powerful techniques of static verification is white-box testing—examining the internals of the blocks rather than their interface. This technique is designed to perform better testing of the block's internal behaviour than coast-to-coast (black-box) testing.

Writing white-box assertions is not a simple task. It requires a deep knowledge and understanding of the design under verification. In addition, using these white-box assertions requires the language and/or the tool to tie the assertion to the correct place in the block of the design under verification. This is because the assertion must follow the internal changes of the block, and not just the changes that are visible from the outside (interface changes). With PSL [1] using assertions embedded in the design code is one possible method to solving this issue.

Embedded PSL assertions also provide a convenient way for writing static verification assertions during the design development stage. When carrying out white-box checking, the block's designer knows more details about the behaviour of the block than anyone else. Those details are important verification items. PSL embedded assertions allow the designer to code these verification items during the development of the block, in the RTL code itself, so that when the block undergoes static verification, these assertions will already be in place. Thus, using PSL embedded assertions can help ensure the quality of the verification process.

The rest of this document is organized as follows: Chapter 2 describes syntax, implementation details, and RuleBase PE user interface support for PSL assertions embedded in Verilog code. Chapter 3 summarizes the features of the tool provided in the framework of Deliverable 3.2/15.

## 2 Embedded PSL Assertions

In the framework of this deliverable, the IBM Static Property Checking tool, RuleBase PE, was extended to support the PSL assertions embedded in the RTL code (Verilog [2]), in addition to the PSL assertions written in the verification units, provided in the separate file. This chapter describes how to write the assertions in the code and how to use them for the static checking.

---

### Syntax

PSL assertions are embedded in the user-written Verilog code, using smart comments—the standard Verilog comments, marked by the reserved word "psl". Each assertion can be written in a single line or expanded over consecutive lines. Smart comments containing embedded assertions can also contain modelling layer code.

#### Single-line assertion:

- Verilog single-line comment [2] that contains the reserved word "psl":

```
// psl <assertion text> ;
```

#### Multi-line assertion:

- A sequence of Verilog single-line comments. The first line starts with the reserved word "psl", and the last line is terminated by ";". This is essentially one assertion segmented to a number of lines:

```
// psl <start assertion text  
// <continue assertion text>  
// <end assertion test> ;
```

- Verilog multi-line comment [2], starting with the reserved word "psl":

```
/* psl  
   <modelling code>  
   <assertion text>  
*/
```

The <> brackets appear in the syntax for readability purpose only and are not part of the syntax.

As can be seen from the syntax, embedded PSL assertions are written in smart comments. As a result, both tools that support these embedded assertions and tools that do not can use the design with embedded assertions.

Embedded PSL assertions can appear only inside Verilog concurrent code. PSL assertions that appear in sequential code are not supported by the tool. Only signals that are known in the module in which the PSL assertion appears can be used inside the assertion. This is the same for regular Verilog concurrent statements appearing in the module.

---

## Implementation

In the first stage, RuleBase PE reads the Verilog code with the embedded assertions and builds a map of the full hierarchy of the design, while the modules that contain embedded PSL assertions are marked. Then, for every instance of a module that contains embedded PSL assertions, a verification unit bound to this instance is produced and the assertions are copied to this verification unit. In this way, the design with the embedded PSL assertions is converted into a pure Verilog design and a set of standalone verification units, each bound to the specific instance in the design. From this point on, RuleBase PE's standard workflow continues, using static model checking algorithms.

### Example of converting the embedded assertions into bound units:

Source code:

```
module top( ... )
    . . .
    with_assert inst_1 ( ... )
    with_assert inst_2 ( ... )
endmodule

module with_assert ( ... )
    . . .
    //psl assert always p -> next q;
endmodule
```

Resulting verification units:

```
vunit embedded_in_with_assert(top.inst_1) {
    assert always p -> next q;
}

vunit embedded_in_with_assert(top.inst_2) {
    assert always p -> next q;
}
```

---

## User Interface

In this section, we describe the part of the RuleBase PE user interface that is related to the displaying and running of the verification units. In the framework of

deliverable 3.2/15, this interface was extended to support Verilog embedded assertions. Text in *italics* describes the new user interface features developed in the framework of deliverable 3.2/15.

**Note:** This section is based on the relevant sections of the RuleBase PE User Guide.

## Running a Vunit Verification Session

Depending on the type of design the user is verifying, the user may have to first compile the design before verifying vunits is possible (see "Compiling the Design" [3]).

The user starts a vunit verification process by following these four simple steps:

1. Choose what to verify by selecting a vunit (rule) in the **Rule List**.
2. Choose how to verify the selected vunit by selecting a configuration in the **Configuration** pane (see "Handling Configurations" [3])
3. Choose where to run the verification process using the two selection boxes in the toolbar.
4. Click **Run Rule** on the toolbar.

**Note:** The user may choose multiple rules in step 1 to allow simultaneous verification of the selected vunits.

After clicking the **Run Rule** button, the RuleBase PE GUI automatically opens a Rule window in the workspace for each vunit the user selected for running. The user uses the Rule window to monitor the verification progress and inspect the verification results (see "Using the Rule Window" [3]).

A vunit verification session ends when one of the following occurs:

- Verification engines finish checking the vunit assertions
- The user clicks the Kill Rule button
- An error occurs

## Using the Rule List

The Rule List is the area where the vunits in the current verification project are displayed.

The Rule List is used to perform various operations, including:

- View vunits
- Sort vunits
- Filter vunits
- Check vunits' status online
- Run vunits verification sessions
- Kill vunits verification sessions
- Build batches

The Rule List pane is divided into lower and upper panes. The lower pane shows the list of vunits, each with a summary of its last verification session. The upper pane is used to filter-out vunits from the list in the lower pane.

## Refreshing the Rule List

The list of vunits reflects the vunits in the user's verification-project source files *and the vunits that represent the embedded assertions in the design under verification.*

To refresh the Rule List, click **Refresh Rule List** on the tool bar, or select **Refresh Rule List** from the Rule menu.

*Note: Using Verilog flavour, RuleBase/PE reads changes in embedded assertions only when the user recompiles.*

## Displaying Rule Window for Vunits

The Rule window shows extensive information about a specific vunit. The user also uses the Rule window to monitor active verification sessions and control their flow, to some extent. For more information, see "Using the Rule List" [3].

### To display a vunit in the Rule window:

Double-click a vunit from the lower pane of the **Rule List** to display the relevant **Rule** window in the Workspace.

## Running a Basic Vunit Verification Session

The user can run a vunit verification session directly from the Rule List.

### To run a vunit verification session:

1. Select a vunit in the lower pane of the Rule List by right-clicking it. A pop-up context menu is displayed.
2. Choose **Run**.

**A Rule window opens for the selected vunit, and the vunit verification session starts.**

## Stopping a Vunit Verification Session

From time to time, the user may wish to stop a specific vunit verification session before it concludes. Reasons may be a very long runtime (in which case the user may want to define a different configuration and rerun) or a wrongly expressed assertion (in which case the user may want to fix his/her assertion and rerun).

### To stop a vunit verification session:

1. Select a vunit with a currently running verification session in the lower pane of the Rule List by right-clicking it. A pop-up context menu is displayed.
2. Choose **Kill**. A pop-up window is displayed, asking the user to confirm the kill action.
3. Click **Yes**.

## Bound Vunits and *Embedded Assertions* (Verilog Flavour)

PSL Verilog flavour allows binding vunits (as well as vmodes and vprops) to specific Verilog modules in the design under test. The user would usually use bound vunits for white-box verifications (verifying design internals rather than its inputs and outputs). For example:

```
vunit test1 (someDesignModule)
{
  ...
}
```

In this example, vunit test1 is bound to the design module someDesignModule.

RuleBase/PE automatically duplicates bound vunits for every instance of the module to which they are bound. Each copy is then verified using its own instance context.

*Embedded assertions appear within the design Verilog files, and are usually written by the hardware designer for sanity-check purposes. Smart comments are used to embed the assertion within the Verilog code. For example:*

```
// psl assert always ( outputA == inputA );
```

*RuleBase/PE automatically generates a vunit for every instance of a module containing embedded assertions. Each such generated vunit is then verified using its own instance context.*

*Unlike regular vunits and bound vunits, vunits containing embedded assertions are verified using vmode default only. Inheriting other vmodes is impossible. When using embedded assertions, the user should plan his/her vmode default to model all legal behaviours of the design inputs.*

### Verifying Bound Vunits and *Embedded Assertions*

RuleBase/PE automatically assigns structured names for bound vunits and for vunits generated from embedded assertions. These names reflect both their associated module and their associated instance.

Bound vunits are assigned a name in the following format:

<vunit name>(<module> = <instance hierarchy>)

*Vunits generated from embedded assertions are assigned a name in the following format:*

*embedded\_in\_<module>(<module> = <instance hierarchy>)*

*Once these names are assigned, bound vunits and vunits generated from embedded assertions can be verified like any other vunit.*

### Filtering Bound Vunits and *Embedded Assertions*

RuleBase/PE lets the user concentrate on just the vunits related to a specific module or a specific module instance. These include bound vunits, vunits generated from a specific module instance, *and vunits generated from embedded assertions within the selected module.*

**To show only vunits bound to a specific module**

1. Click the plus sign next to the **Bound by module** item in the upper pane of the Rule List pane. The list of available modules is displayed.
2. Click the requested module name.

**To show only vunits that relate to a specific module instance**

1. Click the plus sign next to the **Bound by instance** item in the upper pane of the Rule List pane. A tree showing the design's instance hierarchy is displayed.
2. Click the requested instance in the hierarchy.

**Note:** The above filtering also shows vunits related to instances instantiated from the selected instance or from one of its descendants.

*Note:* The above filtering also shows vunits generated from embedded assertions.

# 3 Tool Properties and Development

---

## Features List

Table 1 List of features

	Present	Ref.
<b>Mandatory features</b>		
Pointers to algorithms used	yes	Chapter 2
List of target operating systems	yes	See below
Explanation of coding standards	yes	See below
Discussion of license issues	yes	See below
User documentation, including documentation of user interface (command line switches) and imported/exported file formats	yes	See below
Test suite	yes	See below
Standard input language – PSL	yes	See [1]
Support for Verilog flavour	yes	Chapter 2
Support for embedding of PSL properties in hardware design	yes	Chapter 2
Improved GUI to support new property style	yes	Chapter 2
<b>Nice-to-have features</b>		
Support for other flavours	no	

**Target operating systems:** Linux, Solaris, AIX

**Coding standards:** The tools are implemented in C and C++, according to the coding standards of the IBM Haifa Research Laboratory.

**License issues:** A FlexLM license mechanism is included for the tool. The license can be purchased from IBM under a license agreement.

**User documentation:** The user guide of RuleBase PE is available to licensed users of the tools. This deliverable includes relevant excerpts from the user guide.

**Test suite:** Information about the test suite is delivered, but not the test suite itself.

**Number of new tests written for PROSYD:** 10 designs with their test suites were converted from the designs with bounded vunits to be the designs with embedded assertions.

---

## Work summary

### Starting point

At the starting point of this deliverable, RuleBase PE tool supported PSL Verilog flavour for the standalone verification units only. Verification units had to be provided in the separate file.

The module that supports PSL Verilog flavour is the VFE (Verilog Front-End) module, developed in the framework of PROSYD deliverable 3.3/2 [4].

### Work performed

#### VFE module

VFE (Verilog Front-End) module was extended to have the following capabilities:

- Parse and statically elaborate PSL assertions embedded in the Verilog code using the syntax described in Chapter 2
- Bind embedded PSL assertions to the correct context
- Produce standalone verification units from embedded PSL assertions
- Produce a list of new verification units for the interface

#### GUI module

GUI (Graphic User Interface) was extended to have the following capabilities:

- Display list of verification units combined of standalone units and the units produced from embedded assertions
- Filter the verification units according their origin – standalone or produced from embedded assertions
- Invoke VFE module for the processing of embedded assertions

### Results

RuleBase PE now supports PSL assertions embedded in Verilog code.

---

## Supported PSL subset

The subset of PSL supported by RuleBase PE is described in section 5 in [4]. For the embedded PSL assertions, RuleBase PE supports the same subsets of boolean, temporal and modeling layers of PSL, with the following restrictions:

- Boolean and modeling layers are written in Verilog
- All the PSL constructs that are marked "FoCs only" in [4] are unsupported

From the verification layer, the constructs "assert", "cover", "restrict", "assume" and "default clock" are supported in embedded PSL assertions.

## 4 References

- [1] Accellera. Accellera Property Language Reference Manual. In <http://www.eda.org/vfv/docs/PSL-v1.1.pdf>, June 2004
- [2] The Institute of Electrical and Electronics Engineers, Inc. IEEE Standard Verilog Hardware Description Language, 2001
- [3] RuleBase PE User Guide. Available as a part of the RuleBase PE product
- [4] Porting of IBM tools to support Accelera-Standard version of PSL. PROSYD deliverable 3.3/2