

Accellera FVTC Alignment Sub-Committee Final Report

Surrendra Dudani¹ Cindy Eisner² Dana Fisman^{2,3}
Harry Foster⁴ John Havlicek⁵ Joseph Lu⁶ Erich Marschner⁷
Johan Mårtensson⁸ Ambar Sarkar⁹ Bassam Tabbara¹⁰

¹ Synopsys ² IBM ³ Weizmann Institute of Science
⁴ Jasper Design Automation ⁵ Motorola ⁶ Sun ⁷ Cadence
⁸ Safelogic ⁹ Paradigm Works ¹⁰ Novas Software

February 11, 2004

1 Introduction

On May 29, 2003, Accellera announced the official approval of Accellera PSL (Property Specification Language) 1.01, developed by the Accellera Formal Verification Technical Committee (FVTC), and of SystemVerilog 3.1, developed by the Accellera SystemVerilog Technical Committee. SystemVerilog contains an assertion capability known as SVA, developed by the SystemVerilog Assertions Committee (SV-AC).

During the development of these two standards, the pending co-existence of two different Accellera-approved assertion languages raised concern about the potential for syntactic and semantic conflict between them, which might cause confusion among users. This led to interest in aligning the two languages, to create a “unified kernel” assertion capability that would be common to both. The word ‘kernel’ in this phrase reflects the recognition that PSL and SVA each contained, and would continue to contain, their own specific features that were not and would not be present in the other language, but that where the two languages overlapped, they should be aligned.

While some effort was made during the development of SVA to align with PSL (for instance, the Semantics Sub-committee of the SV-AC included the developers of the PSL formal semantics), only partial alignment was achieved, in part because the PSL definition was in its final stages of documentation by the time SVA was developed. As PSL 1.01 and System Verilog 3.1 were going through final review in early 2003, the “unified kernel”, or common core of syntax and semantics shared by PSL and SVA, was still spotty and poorly defined. In recognition of this, the FVTC proposed a roadmap for further alignment of the two languages. This roadmap was presented to, and approved by, the Accellera Board of Directors in February 2003.

In May 2003, the Alignment Sub-committee of the FVTC was formed to continue alignment of the two languages, following the roadmap approved by the Accellera Board. The Alignment Sub-committee met weekly from May to October 2003, and sporadically from October 2003 to January 2004. Despite

formally being a sub-committee of the FVTC, the Alignment Sub-committee included active members of the SV-AC as well, so that recommendations of the sub-committee ended up influencing both languages as they were further developed to create PSL 1.1 and System Verilog 3.1a. The remainder of this report details the goals, methods, results and conclusions of the Alignment Sub-committee’s work, along with a detailed assessment of the extent to which our results have achieved our goals.

It is our opinion that, as of January 2004, we have achieved as much alignment as is possible given the differing objectives of the two languages. In accomplishing this, we believe that we have both expanded and clearly defined the common core of syntax and semantics shared by the two languages, to the point where a user of one language will feel comfortable using the other, and simple tools can easily translate from one to the other. We thus declare our work complete.

2 Goals

The goals of the sub-committee were simple. In order of decreasing importance, they were:

1. Maintain a sound formal semantic foundation.
2. Don’t have same syntax with different semantics.
3. Allow same syntax for same semantics.

These goals and their priorities were part of the alignment roadmap approved by the Accellera Board of Directors in February 2003.

Note that a complete merge of the two languages was not a goal. Due to the differing objectives of SVA (as an embedded assertion capability of SystemVerilog) and of PSL (as a standalone assertion language with flavors allowing it to work harmoniously with both Verilog and VHDL as well as GDL, and to preserve its extendibility to other flavors as well), each language has features that the other does not. For instance, SVA has dynamic variables, which PSL does not have, and PSL has LTL-like temporal operators, which SVA does not have. No attempt was made to merge features not appearing in both languages. Rather, the above goals address only the common core.

3 Methods

The sub-committee included “champions” from both languages, as well as neutral parties. We worked by consensus, in the following manner. If it was agreed that a change to PSL was needed, this was recorded by the sub-committee for recommendation to the FVTC. If it was agreed that a change to SVA was needed, the SVA champions took it upon themselves to recommend a change to the SV-AC. If it was agreed that the languages should retain their differences due to their differing objectives, no change was recommended to either committee. The changes made in this manner as well as the differences that we decided to maintain are discussed in detail in Sections 4 and 5 below.

4 Results

We have achieved Goals 1 and 2. In addition, we have achieved Goal 3 to the extent possible given the differing objectives of the languages. We first present a summary of the changes made to each language as a result of our work, then a description of the status of Goals 1, 2, and 3.

4.1 Changes to PSL as a result of this committee's work

The following changes to PSL were recommended to (and are expected to be approved by) the FVTC as a result of this committee's work:

1. Syntactic changes
 - (a) In all flavors, the following changes are made:
 - i. Sequences are promoted to properties.
 - ii. SVA syntax (and semantics) are adopted for the `within` operator.
 - iii. Removal of the `whilenot` operator (a derivative of the old `within` operator).
 - iv. Some relaxation of the requirement that any standalone sequence be enclosed in curly braces.
 - v. Addition of new built-in functions `stable`, `isunknown`, `countones`, `onehot` and `onehot0`.
 - (b) In addition, a SystemVerilog flavor of PSL will be defined.
 - i. The PSL LRM V1.01 will point to the SystemVerilog 3.1a LRM for the boolean and modeling layers of this flavor, in the same way that the Verilog and VHDL flavors point to those specifications.
 - ii. The SystemVerilog flavor macros for the current PSL `rose`, `fell`, `prev`, and `inf` will be `$rose()`, `$fell()`, `$past()` and `$`, as in SVA. The SystemVerilog flavor macros for the new PSL built-in functions `stable`, `isunknown`, `countones`, `onehot` and `onehot0` will be `$stable`, `$isunknown`, `$countones`, `$onehot` and `$onehot0`, respectively, as in SVA.
2. Semantic changes
 - (a) Move to a strengthless clock (planned as per section B6 of PSL LRM V1.01).
 - (b) New semantics for abort operator (planned as per section B6 of PSL LRM V1.01).
 - (c) Change to semantics of weak suffix implication so that $\{a;b;c\} \dashv\rightarrow \{d;e[*];false\}$ is satisfiable (planned as per section B6 of PSL LRM V1.01).
 - (d) Refinement of finite trace semantics (weak/neutral/strong).
 - (e) Change to definition of next suffix implication ($\{r1\} \dashv\Rightarrow \{r2\}$) to $\{r1;true\} \dashv\rightarrow \{r2\}$ (was previously $\{r1\} \dashv\rightarrow \{true;r2\}$)
3. Changes to the formal description of the semantics that do not change the semantics themselves
 - (a) Addition of strong booleans.

- (b) $r[*0]$ becomes an inductive base case rather than syntactic sugar.

Note: most semantic changes are transparent to the user, as they either move the semantics closer to what would be intuitively expected by a user (for instance, a strengthless clock), or relate to extreme corner cases unlikely to be noticed in typical use.

4.2 Changes to SVA as a result of this committee's work

The following changes to SVA were recommended as a result of this committee's work (status with respect to approval by the SV-AC appears per item below):

1. Syntactic changes
 - (a) Simplify SVA's $[*]=$ and $[*->]$ to match PSL's $[=]$ and $[->]$. Status: Submitted by Surrendra Dudani to David Smith as an errata on February 8, 2004.
 - (b) Align SVA implication-like operators with PSL implication operators (i.e., switch the use of $->$ vs. $=>$ in SVA to match PSL). Status: Approved for 3.1a.
2. Semantic changes
 - (a) Addition of finite neutral semantics (27 minor changes to correct oversight in original definition). Status: Approved for 3.1a.
 - (b) Three substantive corrections (discovered while analyzing the formal semantics of SVA vs. those of PSL). Status: Approved for 3.1a.
 - (c) Correction for missing 'bar' in first-match semantics. Status: Approved for 3.1a.
 - (d) Addition of weak/strong/neutral mode explanation. Status: Approved for 3.1a.
 - (e) Change to semantics of negated booleans to eliminate the special treatment of negated booleans in the clock rewrite rules. Status: Approved for 3.1a (with Intel dissenting).
3. Changes to the formal description of the semantics that do not change the semantics themselves
 - (a) Wording change to simplify SVA/PSL alignment proofs. Status: Approved for 3.1a.

Note: most semantic changes are transparent to the user, as they relate to extreme corner cases unlikely to be noticed in typical use.

5 Description of the extent to which our results have achieved our goals

The status of our goals is as follows:

Maintain a sound formal semantic foundation. Done, as shown in the following documents:

1. “Recommended Changes to the SVA Formal Semantics”, [3]
2. “Appendix B, Formal Syntax and Semantics and Accellera PSL”, [2]

Don’t have same syntax with different semantics. Done, as shown in the following documents:

1. “Mapping SVA to PSL”, [4]
2. Mapping Table derived from formal mapping, [5]

Allow same syntax for same semantics. Done, to the extent deemed practical. For the most part, a user of one language will feel comfortable in the other. For instance, the weak and strong finite semantics of the SVA property `(a ##1 b ##1 c) |-> (d ##1 e[*0:$] ##1 f)`, when embedded in SystemVerilog code like this:

```
always @(posedge clk)
    assert property ((a ##1 b ##1 c) |-> (d ##1 e[*0:$] ##1 f));
```

are the same as those of the PSL property `{a ; b ; c} |-> {d ; e[*] ; f}`, when asserted in PSL like this:

```
assert always ({a ; b ; c} |-> {d ; e[*] ; f}) @ (posedge clk);
```

While there are some obvious surface differences (`##1` vs. `;` to advance time in a sequence, `()` vs. `{}` to group sequences), the underlying semantics are exactly the same. The significance of this cannot be emphasized enough: prior to the work of this committee, the SVA version, using strong sequence semantics, and the PSL version, using weak sequence semantics, were very different. The SVA version would have returned “unknown” on a finite path where the sequence `(a ##1 b ##1 c)` was followed by a `d` and many `e`’s (but never reached an `f`), while the PSL version would have returned “true” on the same path. The semantic changes described in Sections 4.1 and 4.2 above (in particular, the move to weak/neutral/strong semantics in PSL and the addition of neutral semantics to SVA) give the much more desirable situation that these syntactically similar assertions *hold strongly* and *fail* in exactly the same cases.

Furthermore, the strong PSL version of the same property:

```
assert always ({a ; b ; c} |-> {d ; e[*] ; f!})@(posedge clk);
```

had the value “false” on the path in question according to the PSL V1.01 semantics, again differing in a confusing manner from those of SVA. As a result of the work of this committee, the SVA and strong PSL version return exactly the same result in both languages.

Nevertheless, some syntactic differences remain. They are described in detail below. Note first that since our goal was not a complete merge of both languages

into one, we ignore features that are not in the overlap of the two languages (for instance, dynamic variables in SVA, LTL-like temporal operators in PSL). In addition, we ignore apparent differences that result from the differing objectives of each language. For instance, SystemVerilog assertions are designed to be embedded in SystemVerilog code, therefore an assertion will typically get its `always` and its clock from the enclosing SystemVerilog `always` block. On the other hand, PSL assertions are designed to be standalone, therefore it must use an `always` operator and a clock operator to get the same functionality.

The remaining syntactic differences are a direct result of the differing objectives of the languages. They are:

1. SVA declarative clock vs. PSL clock operator. In SVA, the construct `@c` is a declaration, which takes effect until the end of the scope in which it was declared. For instance, in the SVA sequence `(@c a ##1 b)` the clock `c` is declared within the parentheses and affects anything within the parentheses, in this case both `a` and `b`. In PSL, `@` is an operator, and its right operand affects the left operand. For instance, in the PSL sequence `{a;b}@c` the clock `c` is the right operand of the `@` operator, and affects its left operand `{a;b}`. This difference is inherent in the objectives of the two languages: SVA, with its dynamic variables, takes a more programming-like approach, and therefore supports clocks as declarations within a sequence. PSL takes a more static approach, and therefore cannot accommodate the idea of declarations within a sequence. Note that for this reason it is a good thing that the positioning of the `@c` in SVA vs. PSL is different, otherwise we would violate our Goal 2.
2. SVA edge-sensitive `@c` vs. PSL level-sensitive `@c`. In SVA, the construct `@c` is edge-sensitive, while in PSL, it is level-sensitive. This difference is inherent in the objectives of the two languages: SVA takes its semantics from those of the SystemVerilog construct `always @(c)`, and thus is edge-sensitive (recall that the SystemVerilog `always @(c)` is equivalent to `always @(posedge c or negedge c)`), and cannot tolerate giving different semantics to `@(c)` when attached to an `always` block vs. when attached to a sequence. PSL supports both level-sensitive and edge-sensitive clocks, using `@(c)` and `@(posedge c)`, respectively. Note that this is a second good reason that the positioning of `@c` in SVA vs. PSL is different, otherwise we would violate our Goal 2.
3. Syntactic marking of sequences. PSL requires top-level (i.e., non-nested) use of a sequence to be syntactically marked by curly braces. Thus, to assert that the sequence `a` followed by `b` followed by `c` occurs, PSL uses the syntax `{a;b;c}` while SVA does not require such marking. This difference is inherent in the objectives of the two languages. First, in SVA the only basic property construct is the sequence, while in PSL there are LTL-like temporal operators as well. Thus, in PSL it is critical to help the reader of a property by syntactically distinguishing sequences from other constructs in a way that is immediately apparent, while in SVA there is no such need. Second, in PSL the semi-colon is used as a sequence concatenation operator and as a

- statement terminator. Thus, a syntactic way to clearly mark the beginning and end of sequences is needed, in order to avoid confusion between the two.
4. Property keyword. SVA requires the use of the keyword `property` to mark a concurrent assertion, while PSL does not require such marking. This difference is inherent in the objectives of the two languages: in PSL the only kind of assertion is a concurrent assertion, while in SVA there are immediate (simulation-oriented) assertions as well. Thus, in SVA it is critical to distinguish between the two kinds of assertions, while in PSL there is no such need.
 5. Syntax of standalone assertions. In SVA the standalone (embedded in a module but not in an always block) assertion has an implicit `always` operator, while in PSL there are no implicit operators. This difference is inherent in the objectives of the two languages: an SVA standalone assertion is analogous to a SystemVerilog continuous assign, and therefore implicitly occurs `always`, while a standalone PSL assertion is not bound to a particular HDL and therefore cannot make such an analogy. For the same reason, a PSL standalone assertion is truly standalone, and is thus evaluated starting at the initial cycle, while an SVA standalone assertion appears in the context of SystemVerilog, and therefore needs prefacing with keyword `initial`, like any other SystemVerilog statement, to mark it for evaluation only at the initial cycle.
 6. Syntax of conditional assertion. SVA takes its conditional construct from SystemVerilog, and thus expresses the property that if `a` holds then property `P` should hold using the syntax `if (a) assert property P`, where `assert property P` is the assertion, and `if (a)` is the SystemVerilog `if` statement. PSL takes its conditional construct from temporal logic, and thus expresses the same thing as `assert (a -> P)`. This difference is inherent in the objectives of the two languages: SystemVerilog-dependent assertions in SVA vs. the HDL-independent temporal layer of PSL.
 7. Operator precedence. As stated above, in SVA clocks are declarations, while in PSL there is a clock operator. In addition, in SVA `disable iff` is grammatical at the top level (i.e., not an operator), while in PSL `abort` is an operator. Finally, in SVA `always` is part of an always block, while in PSL the same semantics are achieved with the always operator. For these reasons and others, it is not possible to completely align the operator precedence of SVA and PSL. Since complete alignment was not possible, it was not considered critical to align the precedence of constructs which happen to be operators in both languages. Obviously, a portable assertion can be written by parenthesizing.
 8. Sequence concatenation: `##0/##1` in SVA vs. `:/;` in PSL. In SVA, temporal (“concurrent”) assertions are evaluated only at the occurrence of a clock tick. Thus sequences in SVA are always viewed as being clocked, and the separator between elements of a sequence, `##[n]`, is viewed as a cycle delay. In PSL, temporal assertions and sequences are not required to be clocked (in which case they “tick” at the smallest unit of granularity of time as seen by the verification tool). Since the delay between successive elements in a

PSL sequence is not necessarily a cycle delay, the `##[n]` operator doesn't make sense for PSL. In essence, concatenation using `##0/##1` in SVA can be viewed as a delay in the evaluation of the next portion of the sequence, while in PSL `:/;` are temporal operators.

9. Sequence operations: `intersect/and/or` in SVA vs. `&&/&/|` in PSL. In SVA, conjunction and disjunction of sequences are represented by the keyword operators `intersect`, `and`, and `or`. This is necessary in SVA because sequences are not required to be explicitly bounded by parentheses, and therefore the conjunction/disjunction operators on sequences must be different from the corresponding logical operators. In PSL, which requires that sequences be syntactically recognizable, the same operators can and are used for both logical and sequence conjunction/disjunction.
10. Disable/abort operators: `disable iff (b) f` in SVA vs. `f abort b` in PSL. In SVA, `disable iff` cannot be nested, and thus the ability to terminate a property due to some event is expressed with a top-level grammatical construct (i.e., not as part of the assertion itself, analogous to the way a property gets a top-level `always`). In PSL, nesting of aborted properties is supported, and thus `abort` is a full-fledged temporal operator.

6 Conclusion

From a starting point in May 2003 where the semantics of two very similar assertions were completely different (see Section 5), we have reached a point where the semantics are completely aligned, there are no cases where the same syntax gives different semantics, and remaining syntactic differences are a direct result of the differing objectives of the two languages. This has been accomplished through modifications to the syntax and formal semantics of both languages.

Given the above, it is now possible to write simple scripts/tools to translate descriptions between the two languages. It will also enable tools to allow re-use of predefined sequences or properties in either language, as long as the common subset is used in the specification. This was not possible before the alignment.

Furthermore, it is our firm belief after nine months of intense work that a) the degree of alignment we have achieved is enough to allow the typical user of one language to feel comfortable in the other, and b) we have achieved as much alignment as can be expected given the very different objectives of the two languages. We thus declare our work complete, and respectfully submit this report to our respective committees, to the Technical Committee Chair, and to the Accellera board.

Acknowledgements

Roy Armoni of Intel was originally an active member of this sub-committee. However, in July of 2003 he stopped attending meetings, and due to his lack of involvement since that time declined to be listed as an author of this document.

Avigail Orni and Sitvanit Ruah of IBM joined some of the sub-committee meetings, on the issue of operator precedence.

The sub-committee would like to thank Roy, Avigail, and Sitvanit for their contributions.

References

1. R. Armoni, C. Eisner, S. Dudani, and J. Havlicek. Formal semantics of concurrent [SVA] assertions.
2. D. Fisman, C. Eisner, and J. Havlicek. Appendix B, formal syntax and semantics of Accellera PSL.
3. J. Havlicek. Recommended changes to the SVA formal semantics.
4. J. Havlicek, C. Eisner, D. Fisman, and E. Marschner. Mapping SVA to PSL.
5. E. Marschner, C. Eisner, D. Fisman, and J. Havlicek. Mapping table derived from formal mapping.