

PSL / SVA Alignment Status and Plans



Erich Marschner, Co-Chair

2 December 2003

Alignment Subcommittee

- Goal: Alignment of PSL and SVA
 - Priorities:
 - > 0. maintain a sound formal semantic foundation - DONE
 - > 1. avoid different semantics for same syntax - DONE
 - > 2. allow same syntax for same semantics - still working on this
- Process:
 - Weekly meetings from May to October 2003
 - Definition of Alignment, PSL/SVA Formal Semantics, Mapping
 - Discussion of Syntactic Mapping / Extensions Proposals
 - Decisions within the Alignment Subcommittee
 - Ratification by FVTC



Definition of “Alignment”

- **Assumes:**
 - both SVA and PSL in a System Verilog context
 - > i.e., a SystemVerilog 'flavor' of PSL
 - standalone statements, with any influence from the environment folded in
 - > PSL default clock / SVA clock extraction has been made explicit
 - > disabling of processes in System Verilog not considered
 - mapping from SVA LRM syntax to SVA abstract syntax is defined
- **Includes:**
 - refinement of PSL semantic definition as planned for PSL v1.1 in Dec 2002
 - cleanup or adjustment of both semantic definitions to facilitate mapping
 - temporal operators (for which there is a formal semantics)
 - assert directive
- **Excludes:**
 - procedural assertions (none in PSL)
 - local variables in sequences (none in PSL) (but could perhaps be mapped to forall)
 - strong Booleans in PSL (not in SVA)
 - declarations, verification units, etc. (not formally defined)
 - directives other than assert directive



Accomplishments To Date

- Defined and/or refined formal semantics of SVA and PSL
 - Havlicek, *Recommended Changes to the SVA Formal Semantics*, 31 Aug 03
 - Fisman et al., *App. B, Formal Syntax and Semantics of Accellera PSL*, latest draft 27 Nov 03
- Defined and proved various lemmas about PSL
 - Havlicek et al., *PSL Non-Degeneracy*, latest draft 2 Oct 03
- Defined and proved mapping from SVA to PSL for primitive operations
 - Havlicek et al., *Mapping SVA to PSL*, latest draft 28 Oct 03
 - Mapping table derived from formal mapping - latest draft v0.6
- Defined a common finite trace semantics for PSL and SVA
 - Strong/Weak/Neutral semantics explanation (PSL LRM section 4.4.6)
- Clarified the various related concepts of 'strength' in PSL
 - The concept of strength (new PSL LRM section 4.4.x)
- Proposed some language changes (for clarification, simplification, or alignment)
 - clarification of operator precedence, and alignment with SVA
 - refinement of strength concept - moved to strengthless clocks
 - addition of sequences as properties (e.g., `assert a -> next {b;c;d}`), as in SVA
 - replacement of PSL 'within' property with SVA 'within' syntax/semantics



Refinement of PSL Formal Definition

- Done:
 - restatement of abort semantics [planned Dec 02]
 - refinement of finite trace semantics (strong/weak/neutral) [planned Dec 02]
 - change to strengthless clocks
 - addition of strong booleans
 - refinement of definition of $r[*0]$
 - addition of sequences as properties
 - redefinition of suffix implication based on $|->$, $|\Rightarrow$ and RHS sequences as properties
 - definition of next suffix implication: $\{r1\} |\Rightarrow \{r2\} == \{r1; true\} |-> \{r2\}$
 - operator precedence and associativity clarification/alignment
 - alignment of formal syntax with BNF
- Impact:
 - most changes are transparent to users - relate to extreme corner cases
 - some operator precedence changes ($->$, abort) affect backward compatibility
 - additions (only a few so far) make the language more capable, more consistent
 - strong/weak/neutral semantics change simplifies handling of finite traces



Refinement of SVA Formal Definition

- Done:
 - addition of finite neutral semantics (27 minor changes)
 - substantive corrections (3 changes)
 - wording change to simplify SVA/PSL alignment proofs (1 change)
 - clarification (1 change)
 - font correction (1 change)
 - correction for missing 'bar' in first-match semantics
 - addition of strong/weak/neutral mode explanation
- Impact:
 - changes have essentially no impact on users
 - strong/weak/neutral semantics clarify how implementors should deal with finite traces



Strong/Weak/Neutral Semantics

- For simulation (finite traces), we need strong/weak/neutral semantics
 - **weak** for constrained random simulation, because test will stop at an arbitrary place, and pending obligations should not be treated as failures
 - **strong**, for directed tests, because test will presumably run a full scenario to completion, and pending obligations should be treated as failures
 - **neutral**, the traditional LTL finite trace semantics, which is effectively a compromise between the weak and strong cases
- PSL originally addressed strong semantics via strong clocks
- Dec 2002 paper proposed addition of strong/weak/neutral semantics in PSL v1.1
- Mar 2003 SVA formal semantic definition adopted strong/weak semantics, but left out neutral semantics (an oversight)
- We've refined both PSL and SVA formal semantics definitions to include strong/weak/neutral semantics



Strong/Weak/Neutral Result “Modes”

- Holds strongly:
 - no bad states have been seen
 - all future obligations have been met
 - the property will hold on any extension of the path
- Holds (but not strongly):
 - no bad states have been seen
 - all future obligations have been met
 - the property may or may not hold on any extension of the path
- Pending:
 - no bad states have been seen
 - future obligations have not been met
 - (the property may or may not hold on any extension of the path)
- Fails:
 - a bad state has been seen
 - (future obligations may or may not have been met)
 - (the property may or may not hold on any extension of the path)



Syntactic Mapping

- Mapping Table shows equivalent syntactic forms in PSL, SVA
 - Derived from proven mappings between formal semantics of PSL, SVA
- Mapping Table emphasizes similarity, minimizes differences
 - Uses parallel forms where possible
 - Uses parentheses where possible
 - Assumes no implicit clock context
 - Glosses over slight differences regarding strength of sequences
 - > PSL sequences are by default weak, but can be made strong by appending ‘!’
 - > SVA sequences are strong, but simulation will treat them as weak
- Mapping Table highlights remaining differences
 - We will try to address some of these between now and mid-January
 - Some proposals exist, but the Alignment Subcommittee has not yet discussed them



Syntactic Mapping - Sequences

	SVA Form	PSL Form
	B (= a SV expr)	B (= a SV expr)
	C (= a SV event expr)	C (= a SV event expr)
	Unlocked sequences/SEREs	
	B, Rs, Rs1, Rs2	B, Rp, Rp1, Rp2
	B	B
	(Rs)	{ Rp }
	Rs1 ##1 Rs2	Rp1 ; Rp2
	Rs1 ##0 Rs2	Rp1 : Rp2
	(Rs1) or (Rs2)	{ Rp1 } { Rp2 }
	(Rs1) intersect (Rs2)	{ Rp1 } && { Rp2 }
	Rs [* 0]	Rp [* 0]
	Rs [* 1 : \$]	Rp [* 1 : inf]
	Clocked sequences/SEREs	
	Bc, Rsc, Rsc1, Rsc2	Bc, Rpc, Rpc1, Rpc2
	@ (C) (Rs)	{ Rp } @ (C)
	@ (C) B	B @ (C)
	Rsc1 ## Rsc2	Rpc1 ; Rpc2



Syntactic Mapping - Unclocked Properties

	SVA Form	PSL Form
	Unclocked properties	
	Ps	Pp
	(Rs)	{ Rp }
	not (Rs)	!{ Rp }
	(Rs1) -> (Rs2)	{ Rp1 } -> { Rp2 }
	not ((Rs1) -> (Rs2))	!({ Rp1 } -> { Rp2 })
	(Rs1) -> not (Rs2)	{ Rp1 } -> !{ Rp2 }
	not ((Rs1) -> not (Rs2))	!({ Rp1 } -> !{ Rp2 })
	disable iff (B) Ps	Pp abort B



Syntactic Mapping - Clocked Properties

	SVA Form	PSL Form
Clocked properties		
	Psc	Ppc
	@ (C) B	B @ (C)
	@ (C) (Rs)	{ Rp } @ (C)
	@ (C) not B	!(B) @ (C)
	@ (C) not (Rs)	!{ Rp } @ (C)
	@ (C) ((Rs1) -> (Rs2))	({ Rp1 } -> { Rp2 }) @ (C)
	@ (C) not ((Rs1) -> (Rs2))	!({ Rp1 } -> { Rp2 }) @ (C)
	@ (C) (Rs -> not B)	({ Rp } -> (!B)) @ (C)
	@ (C) (Rs1 -> not Rs2)	({ Rp1 } -> !{ Rp2 }) @ (C)
	@ (C) not ((Rs1) -> not (Rs2))	(!({ Rp1 } -> !{ Rp2 })) @ (C)
	@ (C) disable iff (B) Psc	(Ppc abort B) @ (C)
	(Rsc)	{ Rpc }
	not (Rsc)	!{ Rpc }
	(Rsc1) -> (Rsc2)	{ Rpc1 } -> { Rpc2 }
	not ((Rsc1) -> (Rsc2))	!({ Rpc1 } -> { Rpc2 })
	(Rsc1) -> not (Rsc2)	{ Rpc1 } -> !{ Rpc2 }
	not ((Rsc1) -> not (Rsc2))	!({ Rpc1 } -> !{ Rpc2 })
	disable iff (B) Psc	Ppc abort B



Syntactic Mapping - Assertions

	SVA Form	PSL Form
	Assertions	
	always assert property (Psc)	assert always (Ppc)
	always @ (C) assert property (Ps)	assert always (Pp) @ (C)
	initial assert property (Psc)	assert (Ppc)
	initial @ (C) assert property (Ps)	assert (Pp) @ (C)
	always if (B) assert property (Psc)	assert always (B -> Ppc)
	always @ (C) if (B) assert property (Ps)	assert always (B -> Pp) @ (C)
	initial @ (C) if (B) assert property (Ps)	assert (B -> Pp) @ (C)



Remaining Issues - Symbols

- Some repetition operators are slightly different:
 - PSL [=], [->] vs. SVA [*=], [*->]
 - Possible Resolution: Allow both forms in both languages
- Sequence concatenation operators are significantly different:
 - PSL ;/: vs. SVA ##0/##1
 - Possible Resolution: Define alternative operators |+, |# that both languages can support
- A number of operators and keywords are still significantly different:
 - PSL &&, &, | vs. SVA intersect, and, or
 - PSL ! vs. SVA not
 - PSL rose/fell vs. SVA \$rose/\$fell
 - PSL inf vs. SVA \$
 - Possible Resolution: Define SVA operators as part of a "System Verilog" flavor of PSL
- Some new operators in System Verilog appear to conflict with PSL operators:
 - PSL |->, -> (same cycle implication) vs. SVA => (implication in constraints)
 - PSL |=> (next cycle implication) vs. SVA -> (state transition)
 - Possible Resolution: Change SVA to align with PSL

Note:

Possible Resolutions still need to be discussed by the Alignment Subcommittee



Remaining Issues - Syntax

- Some clocking differences remain:

- PSL (f)@c, {r}@c vs. SVA @c (f)
- Possible Resolution: Allow, or require, @c (f), @c {r} in PSL

- PSL level-sensitive @(c) vs. SVA edge-sensitive @(c)
- Possible Resolution: Encourage SV users to avoid @(c) - non-synthesizable

- Some capability differences remain:

- PSL idiom (b[*] && {r}) vs. SVA (b throughout (r))
- Possible Resolution: Add SVA 'throughout' to PSL along with SVA 'within'

- Requirements for parenthesization are somewhat different:

- PSL compound sequences need {} vs. SVA compound sequences don't always need ()
- PSL hierarchical properties need () vs. SVA (non-hierarchical) properties need fewer ()
- Possible Resolution: Encourage use of parentheses in SVA
 - > for compatibility with future extensions as well as with PSL

Note:

Possible Resolutions still need to be discussed by the Alignment Subcommittee



Remaining Issues - Top Level

Note:

Possible Resolutions still need to be discussed by the Alignment Subcommittee

- Top-level syntax of assertions is still different:
 - PSL assert always P vs. SVA always assert property P
 - Possible Resolution: Make 'property' keyword optional in SVA
 - PSL assert P vs. SVA initial assert [property] P
 - Possible Resolution: Make 'initial' keyword optional in SVA
 - PSL assert always (B -> P) vs. SVA always if (B) assert [property] P
 - Possible Resolution: Add -> operator to SVA as alternative to 'if '
 - PSL (f) abort b vs. SVA disable iff (b) (f)
 - Possible Resolution: Add abort operator to SVA as alternative to 'disable iff '
 - PSL assert always @(c) (f) vs. SVA always @(c) assert [property] (f)
 - Possible Resolution: Allow both forms in SVA - i.e., allow PSL assertions in System Verilog



Operator Precedence, Associativity

- Current proposal (highest to lowest):
 - @ binary, left-associative
 - [*] [=] [->] unary postfix
 - SERE & && binary, left-associative
 - SERE | binary, left-associative
 - within binary, left-associative
 - : binary, left-associative
 - ; binary, left-associative
 - abort binary, left-associative
 - | -> | => binary, right-associative
 - X G F always never eventually! next* unary prefix
 - U W until* before* binary, right-associative
 - <-> -> binary, right-associative
- Changes w.r.t. PSL v1.01 are:
 1. Removed within* and whilenot* operators, replaced by within on SEREs.
 2. SERE operator precedence clarifications remove need for requiring braces.
 3. The <->, -> operators moved down in precedence, as in EDL.
 4. The abort operator has moved up in precedence, to be on the same side of both classes of implication operators.

