SVA Checker Rules Manual

Version C-2009.06
June 2009

Comments?
E-mail your comments about this manual to:
vcs_support@synopsys.com.

SYNOPSYS®

Copyright Notice and Proprietary Information

Right to Copy Documentation

The license agreement with Synopsys permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any. Licensee must assign sequential numbers to all copies. These copies shall contain the following legend on the cover page:

> This document is duplicated with the permission of Synopsys, Inc., for the exclusive use of
>
> _____ and its employees. This is copy number_____."

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Registered Trademarks (®)

Synopsys, AMPS, Cadabra, CATS, CRITIC, CSim, Design Compiler, DesignPower, DesignWare, EPIC, Formality, HSIM, HSPICE, iN-Phase, in-Sync, Leda, MAST, ModelTools, NanoSim, OpenVera, PathMill, Photolynx, Physical Compiler, PrimeTime, SiVL, SNUG, SolvNet, System Compiler, TetraMAX, VCS, Vera, and YIELDirector are registered trademarks of Synopsys, Inc.

Trademarks (™)

AFGen, Apollo, Astro, Astro-Rail, Astro-Xtalk, Aurora, AvanWaves, Columbia, Columbia-CE, Cosmos, CosmosEnterprise, CosmosLE, CosmosScope, CosmosSE, DC Expert, DC Professional, DC Ultra, Design Analyzer, Design Vision, DesignerHDL, Direct Silicon Access, Discovery, Encore, Galaxy, HANEX, HDL Compiler, Hercules, Hierarchical Optimization Technology, HSIMplus, HSPICE-Link, iN-Tandem, i-Virtual Stepper, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, Liberty, Libra-Passport, Library Compiler, Magellan, Mars, Mars-Xtalk, Milkyway, ModelSource, Module Compiler, Planet, Planet-PL, Polaris, Power Compiler, Raphael, Raphael-NES, Saturn, Scirocco, Scirocco-i, Star-RCXT, Star-SimXT, Taurus, TSUPREM-4, VCS Express, VCSi, VHDL Compiler, VirSim, and VMC are trademarks of Synopsys, Inc.

Service Marks (SM)

MAP-in, SVP Café, and TAP-in are service marks of Synopsys, Inc.

SystemC is a trademark of the Open SystemC Initiative and is used under license.
ARM and AMBA are registered trademarks of ARM Limited.
Saber is a registered trademark of SabreMark Limited Partnership and is used under license.
All other product or company names may be trademarks of their respective owners.

Contents

1

SVA Checker

This chapter provides reference information about the SVA Checker available with VCS. SVA Checker is a static code purification core for checking assertions in SystemVerilog code. The tool contains rules that perform semantic analysis and check for coding styles that may cause problems. SVA Checker helps you identify common and subtle coding mistakes.

Note:

This manual is part of the VCS Design Checker documentation suite. For more information about Design Checker, see the *VCS/VCSi Design Checker User Guide*.

The SVA Checker rules are organized into the following categories:

ASSERT

This category contains rules that verify the structure of statements like assert property, assume property, and cover property.

ASSERT001

Message: Implication is used in cover property statement

Description	SVA Checker issues this message when it finds an implication is used in cover property statement. s |-> p; s |-> not p; or similarly with |=> appears in a cover property statement. It is not recommended to use implications in cover properties because vacuous successes may not be distinguished from real successes. Covers that contain the following forms are acceptable: A sequence, not (s |-> not p) not (s |-> p) and similarly with |=> Note that the negated implication corresponds to concatenation of a sequence with a property.
Language	SystemVerilog
Severity	Lint

Example

The following example shows SystemVerilog code that flags this rule:

```
module ASSERT001;
reg clk, rst, a, b;
```

```
a1:cover property (@(posedge clk) disable iff(rst) a ##1 b);
a2:cover property(@(posedge clk) disable iff(rst) a |-> b);
// rule ASSERT001 is flagged in the above line

a3:cover property(@(posedge clk) not ( a |-> b) );

a4:cover property(@(posedge clk) a |-> not b );
// rule ASSERT001 is flagged in the above line

endmodule
```

ASSERT002

Message: Cover property is not disabled when reset is in progress

Description	SVA Checker issues this message when the cover property is enabled and reset is in progress. Cover property statement has no *disable iff* or there is no *throughout* operator over sequence form of property.The cover property may collect invalid information during reset, which may skew the coverage results.
Language	SystemVerilog
Severity	Lint

Example

The following example shows SystemVerilog code that flags this rule:

```
module ASSERT002;
reg clk, rst, a, b, c;
```

```
a1:cover property (@(posedge clk) disable iff(rst) a ##1 b);

a4:cover property(@(posedge clk) c throughout a ##1 a |->
not b);
// rule ASSERT002 is flagged in the above line

a5:cover property(@(posedge clk) c throughout a ##1 b );

a6:cover property(@(posedge clk) a ##1 (c throughout (b ##5
c)) );
// rule ASSERT002 is flagged in the above line

endmodule
```

ASSERT003

Message: Inferred clocking expression is redefined

Description	SVA Checker issues this message when it finds a property contains a leading clock specification but the verification statements infers a different clock.
Language	SystemVerilog
Severity	Lint

Example

The following example shows SystemVerilog code that flags this rule:

```
property p;
    @clk1 s;
endproperty : p
```

```
always @clk2 a: assert property(p); // ASSERT003 fails
```

ASSERT004

Message: Referring to values at time < 0 in sampled value functions should be avoided

$past is at fault that many times: N

$rose, $fell, $stable is at fault that many times: N

Description	SVA Checker issues this message when it finds a trigger (first) boolean expressions in a verification statement is a sampled value function $past, $rose, $fell or $stable, because these functions depend on a preceding values that could be undefined at the first clock tick. The check proceeds recursively over property and sequence operators and computes the minimum sequence length to that point. If it encounters a sampled value function as the only term in a boolean expression, it compares the past cycle delay needed with the current sequence length. The past delay needed for $fell, $rose, and $stable is 1, and it is the specified value for $past or its default value is 1. For multi-clock properties and sequences, it checks each component sequence independently because nothing is known about the clock frequency ratios.
Language	SystemVerilog
Severity	Lint

Example

The following example shows SystemVerilog code that flags this rule:

```
module ASSERT004;
bit clk1, clk2, rst, a, b, c, d;

generate
begin : fail
     a0: assert property (@clk1 ##0 $rose(a));

     a1: assert property (@clk1 a ##0 $rose(b) ##1 c |-> d);

     a2: assert property (@clk1 a ##[0:2] $rose(b) ##1
      c |-> d);
// rule ASSERT004 is flagged in the above 3 lines

end

endgenerate

endmodule
```

ASSERT005

Message: Assertions outside initial block should be disabled upon reset.

Description	SVA Checker issues this message when it detects an assert or assume statement without the *disable iff* clause.
Language	SystemVerilog
Severity	Lint

Example

The following example shows SystemVerilog code that flags this rule:

```
module ASSERT005;
bit clk, a, b, c;

a1: assert property(@(posedge clk) disable iff(a) b);

a2: assert property(@(posedge clk) b);
// rule ASSERT005 is flagged in the above line

a3: cover property (@(posedge clk) b);
a4: assume property(@(posedge clk) disable iff(a) b);

a5: assume property(@(posedge clk) b);
// rule ASSERT005 is flagged in the above line


initial @(posedge clk) begin : init_b
    a1: assert property( disable iff(a) b);
    a5: assume property( b);
end
```

```
endmodule
```

ASSERT006

Message: The same property is used in both assert property and assume property

Description	SVA Checker issues this message when it finds the same property is used in both assert property and assume property. This check is performed only within a module. So, if the same property is used with different verification statements in different modules, the Checker will not flag this rule.
Language	SystemVerilog
Severity	Lint

Example

The following example shows SystemVerilog code that flags this rule:

```
module ASSERT006;
bit clk, rst, a, b, c;

property p0;
     @(clk) a;
endproperty

property p1;
     @clk b;
endproperty

property p3(x);
```

```
    @clk x;
endproperty

generate
begin : newdef
    property p0;
        @clk a;
    endproperty

end
begin : pass
    a0: assert property (p0);
    a1: assume property (p1);
    a3: cover property (p0);
    a4: cover property (p1);
end

begin : fail
    a0: assert property (p0);

    a1: assume property (p0);
    // rule ASSERT006 is flagged in the above line

    a2: assert property (p3(a));

    a4: assume property (p3(b));
    // rule ASSERT006 is flagged in the above line

end

endgenerate

endmodule
```

ASSERT007

Message: Disable iff does not contain a simple expression

Description	SVA Checker issues this message when a statement is disabled by an expression that is not a signal or its complement or not an ==, !=, ===, !== comparison of a signal or its complement with a constant expression.
Language	SystemVerilog
Severity	Lint

Example

The following example shows SystemVerilog code that flags this rule:

```
module ASSERT007 (input bit c);
bit clk, rst, a, b;

generate
begin : loc
    bit x;
end

begin : pass

a1: assert property (@(posedge clk) disable iff (rst) a);
a2: assert property (@(posedge clk) disable iff (!rst) a);

end // block: pass

begin : fail
    a1: assert property (@(posedge clk) disable iff
        (rst && b) a);
```

```
    // rule ASSERT007 is flagged in the above line


    a2: assert property (@(posedge clk) disable iff
        (!(rst && b)) a);
    // rule ASSERT007 is flagged in the above line

end

endgenerate

endmodule
```

ASSERT016

Message: Signal from a clocking event also appears in property body or in reset expression.

For occurrence in disable iff: clock signal occurs that many times in disable iff: N.

For occurrence in the property: clock signal occurs that many times in the property: N.

where N, is the number of occurrences in the verification statement.

Description	SVA Checker verifies if a signal from a clocking event also appears in the body of the property or in the disable iff expression, if it exists. The danger is that either it will always be disabled by the clock or it will always sample the same value of the signal. The clocking event is restricted to be an optional edge specifier and a signal identifier or its complement (no complex expression).
Language	SystemVerilog
Severity	Lint

Example

The following example shows SystemVerilog code that flags this rule:

```
module ASSERT016 (input bit c);
```

```
bit clk, clk1, clk2, rst, a, b, d;

generate
begin : loc
    bit x;
end

begin : pass
    a1: assert property (@(posedge clk) disable iff (rst) a);
    a2: assert property (@(posedge clk) disable iff
        (!rst)a ##2 b);
end

begin : fail
    a1: assert property (@(posedge clk) disable iff (rst)
        clk);
    // rule ASSERT016 is flagged in the above line

    a2: assert property (@(posedge clk) disable iff
        (!rst)a ##2 !clk);
    // rule ASSERT016 is flagged in the above line

end

endgenerate

endmodule
```

ASSERT017

Message: Signal from a reset condition appears in property expression.

Problem occurs that many times: N.

where N, is the number of occurrences in the verification statement.

Description	SVA Checker verifies if a signal from the disable iff expression also appears in the body of the property. The property may behave in an unexpected way.
Language	SystemVerilog
Severity	Lint

Example

The following example shows SystemVerilog code that flags this rule:

```
module ASSERT017 (input bit c);
bit clk, clk1, clk2, rst, a, b, d;

generate
begin : loc
     bit x;
end

begin : pass

     a1: assert property (@(posedge clk) disable iff (rst) a);
```

```
    a2: assert property (@(posedge clk) disable iff
        (!rst)a ##2 b);

end

begin : fail

    a1: assert property (@(posedge clk) disable iff
        (rst) rst);
    // rule ASSERT017 is flagged in the above line

    a2: assert property (@(posedge clk) disable iff
        (!rst) a ##2 !rst);
    // rule ASSERT017 is flagged in the above line

end

endgenerate

endmodule
```

ASSERT018

Message: Un-qualified assertion clocking event

Description	SVA Checker issues this message when it detects an assertion clocking event without the posedge or negedge qualifier. This may have unintended behavior because verification attempts will progress on both edges of the clock.
Language	SystemVerilog
Severity	Lint

Example

The following example shows SystemVerilog code that flags this rule:

```
module ASSERT018;
bit clk, clk1, clk2, rst, a, b, c;

generate
begin : pass
     a1: assert property(@(posedge clk) a);
     a2: assert property(@(negedge clk) a ##1
          @(posedge clk1) b);
end

begin : fail
     a1: assert property(@clk a);
     // rule ASSERT018 is flagged in the above line
     a2: assert property(@(clk) a ##1 @(clk1) b);
     // rule ASSERT018 is flagged in the above line
end
endgenerate

endmodule
```

ASSERT019

Message: Assert or assume statement that is not a bool, is without not or implication or if operator.

Description	An assertion that consists of temporal sequences or properties that do not contain negation or implication (|->, |=>, if, if-else) are mostly incorrect and will fail in most evaluation attempts, unless they are just single boolean expression invariants.
	For example,
	`a1: assert property(@(posedge clk) a ##1 b);`
	
	will fail at any clock tick, where a is not true or b is not true at the next clock tick.
	
	You may want to disallow such a sequence a ##1 b in which case the assertion should have the following form:
	`a1: assert property(@(posedge clk) not (a ##1 b) );`

Language	SystemVerilog
Severity	Lint

Example

The following example shows SystemVerilog code that flags this rule:

```
module ASSERT019;
```

```
bit clk, clk1, a, b, c, d, e;

generate
begin : pass
     a1: assert property (@clk a&b);
     a2: assume property (@clk a);
end // block: pass

begin : fail
     a1: assert property (@clk a ##1 b);
     // rule ASSERT019 is flagged in the above line

     a2: assume property (@clk a ##1 b);
     // rule ASSERT019 is flagged in the above line

end

endgenerate

endmodule
```

BUILTINFN

The rules listed under this category checks the structure of built-in (sampled value) functions.

BUILTINFN001

Message: $past shall not be used over a large number of clock cycles (> 20)

For property expression: occurs that many times in property expression: N

For disable iff expression: occurs that many times in disable iff expression: N

where N, is the number of occurrences in the verification statement

Description	SVA Checker issues this message when it finds $past system function is being used over a large number of clock cycles (> 20). You need to reformulate the property as this may impact the performance. The verification statement is checked for such usage of $past in both the property expression and in the disable iff expression and each is reported separately. If there is more than one occurrence of a $past that violates the rule, it is reported only once, but it indicates the number of occurrences.
Language	SystemVerilog
Severity	Lint

Example

The following example shows SystemVerilog code that flags this rule:

```
module BUILTFN001;
bit clk, a, b, c;

generate
begin : pass
     a1: assert property (@clk $past(a));
     a2: assert property (@clk a ##1 $past(b) |-> not(a
          ##3 $past(c)));
end

begin : fail
     a1: assert property (@clk $past(a, 21));
     // rule BUILTINFN001 is flagged in the above line

     a2: assert property (@clk a ##1 $past(b, 21) |-> not(a
          ##3 $past(c, 21)));
     // rule BUILTINFN001 is flagged in the above line

end

endgenerate

endmodule
```

PROPERTY

The rules listed under this category checks the structure of properties.

PROPERTY001

Message: A property is declared but never used in a verification statement

Description	SVA Checker issues this message when it detects a property was defined in a module but it was not instantiated in an assert, assume, or cover property statement directly or as an instance in another property that is used in a verification statement. Note:Enable rule SEQUENCE005 to detect an unused sequence.
Language	SystemVerilog
Severity	Lint

Example

The following example shows SystemVerilog code that flags this rule:

```
module PROPERTY001;
bit clk, clk1, clk2, rst, a, b, c;

generate
begin : pass
     property pg;
          @(posedge clk) a ##1 b |=> @(posedge clk) c;
     endproperty

     a1: assert property (pg);
end

begin : fail
     property p1; // rule PROPERTY001 flags here
```

```
          @(clk) a;
     endproperty
end

endgenerate

endmodule
```

PROPERTY002

Message: Consequent of suffix implication ends with * repetition

Description	SVA Checker issues this message when it detects a property ending with * repetition. For example, a property ends with s[*M:N] where, N=$. This * ending repetition is redundant, as it will match on the first M occurrences, if at all. Use [*M] only or modify the property in some other way. For example, a \|-> s[*1:$] is same as a |-> s. Similarly, not(s[M:N) is equivalent to not(s[M]).
Language	SystemVerilog
Severity	Lint

Example

The following example shows SystemVerilog code that flags this rule:

```
module PROPERTY002;
bit clk, clk1, clk2, rst, a, b, c;

generate
begin : pass
```

```
    sequence s(n);
        (a ##1 b)[*1:n];
    endsequence

    property p1;
        @(posedge clk) a ##1 s(2) |=>  s(1) ##1 c;
    endproperty

    a1: assert property (p1);
    a2: assert property (@clk a |-> s(1) within s(4) );
end

begin : fail
    sequence s(n);
        (a ##1 b)[*1:n];
    endsequence // s

    property p1;
        @clk a ##1 b |=> s(3);
    endproperty

    a1: assert property (p1);
    // rule PROPERTY002 is flagged in the above line

    a2: assert property(@clk s(2) |-> (s(2) or s(4)));
    // rule PROPERTY002 is flagged in the above line

end

endgenerate

endmodule
```

PROPERTY008

Message: within appears in the antecedent of a suffix implication

Description	SVA Checker issues this message when it detects `within` appearing in the antecedent of a suffix implication. This may impact compilation and run-time performance due to many concurrent threads and may not give the expected result. You need to rewrite in a different way. For example, in the following property, if a ##1, a can occur many times in s2, then for each such matches, s3 will be checked. (a ##1 a) within s2 |-> s3 Perhaps, the intent was (a[->1] ##1 a ##1 !a[*0:$]) intersect s2 |-> s3, in which case there is a match only if a ##1; a occurs exactly once within s2.
Language	SystemVerilog
Severity	Lint

Example

The following example shows SystemVerilog code that flags this rule:

```
module PROPERTY008;
bit clk, clk1, rst, a, b, c;

generate
begin : pass
```

```
    a1: assert property (@clk a ##1 b |-> (a ##1 a)
         within b[->1]);
    a2: assume property (@clk a |=> b within (a ##1 c));
    a3: cover property (@clk a within a ##1 b);
end

begin : fail
    a1: assert property (@clk  (a ##1 a) within b[->1] |=> a
         ##1 b);
    // rule PROPERTY008 is flagged in the above line

    a12: cover property (@clk (a within a ##1 b) ##[1:2]
          c |-> a);
    // rule PROPERTY008 is flagged in the above line

end

endgenerate

endmodule
```

PROPERTY009

Message: Sequence *or* is top-level operator in antecedent of suffix implication

Description	SVA Checker issues this message when it detects a sequence *or* is used as top-level operator in antecedent of suffix implication. Using sequence *or* in the above situation may impact performance. You need to consider splitting the property into two, one for each disjunct, especially if it is a top-level implication. This rule searches for a top-level *or* in the antecedent in each sequence component of a multi clock sequence or when it is an argument to a top-level first_match operator.
Language	SystemVerilog
Severity	Lint

Example

For example, the property

```
property P;
    Seq1 or Seq2 |-> P0;
endproperty
```

can be rewritten as

```
property P1;
    Seq1  |-> P0;
endproperty

property P2;
    Seq2 |-> P0;
endproperty
```

with a separate verification statement over each property.

PROPERTY010

Message: *And* is the top-level operator of a property.

Description	SVA Checker issues this message when it detects a property or sequence having *and* as the top-level operator. Using this may impact performance. You need to consider splitting the property into two, one for each conjunct, especially if it is a top-level implication. This rule searches for a top-level property or sequence *and* as the top-level operator or in the consequent of a suffix implication. Also, if within a component of a multi clock sequence or when it is an argument to a top-level first_match operator. A negated property is not searched because a conjunction corresponds to a disjunction when complemented.
Language	SystemVerilog
Severity	Lint

Example

For example, the property

```
property P;
    Seq0  |-> Prop1 and Prop2;
endproperty
```

can be rewritten as

```
property P1;
    Seq0  |-> Prop1;
```

```
endproperty

property P2;
    Seq0 |-> Prop2;
endproperty
```

with a separate verification statement over each property.

PROPERTY011

Message: Last boolean in antecedent of |-> or condition of if - else is the complement of first boolean in the consequent

Description	In a property where the last boolean of the antecedent of |->, or the condition of if is the complement of the first boolean of what follows, the property may always fail or pass (if negated). Such a property does not verify anything. SVA Checker issues the message under this condition. The boolean can be expressions; however, they must be syntactically identical except for the top-level complement. There is no functional verification of mutual complementarity. For example, the following cases are detected: a |-> !a where, a is some expression a |-> $fell(a) $fell(a) |-> $fell(!a); $rose(a) |-> !a etc. The correction may be to use a non-overlapping implication or a cycle delay inserted to separate the two boolean. The check ignores antecedents that have top-level sequence *and, or*, or *intersect* operations, which may have multiple threads or intersections of boolean to consider. In *within* and *throughout* operations, only the first boolean of the base sequence is considered. If the consequent is an *if - else* property, the antecedent is first checked against the *if* condition. Then, the *if* condition is checked against the branches of the *if*. If the consequent is a property or sequence operator *and / or*, the last boolean is checked against the first boolean of both branches of the operator. If the consequent property starts with an unary ##M, then the check is applied only if *M == 0.*
Language	SystemVerilog
Severity	Lint

Example

The following example shows SystemVerilog code that flags this rule:

```
module PROPERTY011;
bit clk, clk1, rst, a, b, c, d;

generate
begin : pass
    a1a: assert property (@clk  (a ##1 a) |-> a);
    a1b: assert property (@clk  (a ##1 a) |=> !a);
end

begin : fail
    a1a: assert property (@clk  (a ##1 a) |-> !a);
    // rule PROPERTY011 is flagged in the above line

    a1b: assert property (@clk  (a ##1 !a) |-> a);
    // rule PROPERTY011 is flagged in the above line

end

endgenerate

endmodule
```

PROPERTY012

Message: Open-ended or large ranges in ## should not be used in antecedent of a suffix implication

Description	SVA Checker issues this message when it detects an open-ended or large ranges in ## used in antecedent of a suffix implication. For example, ## [M:N] where, N is $ or N-M > 20 is used in the antecedent of an implication and there is no first_match applied to the subsequence to limit the number of generated threads. This usage may impact performance. You need to consider rewriting the property using [*] or [->] operators.
Language	SystemVerilog
Severity	Lint

Example

The following example shows SystemVerilog code that flags this rule:

```
module PROPERTY012;
bit clk, clk1, rst, a, b, c, d;

generate
begin : pass
     a1: assert property (@clk  ##[1:21]a |-> b);
     a6: assume property (@clk ##[0:20] b[->1] |-> a);
end
```

```
begin : fail
     a1: assert property (@clk ##[1:22] a |-> b);
     // rule PROPERTY012 is flagged in the above line


a6: assume property (@clk ##[0:21] b[->1] |-> a);
     // rule PROPERTY012 is flagged in the above line

end

endgenerate

endmodule
```

PROPERTY013

Message: Open-ended delay used without constraint in the consequent of a suffix implication.

Description	SVA Checker issues this message when it detects an open-ended delay is used without constraint in the consequent of a suffix implication. A property with open-ended ## delay or [->] or [=] used without constraining cannot fail or if negated cannot succeed. The situation is detected even when nested deep in other operators. No message is issued when the open-ended range is constrained by one of the following constructs: • There is an *intersect* operation above this range. • The range appears on the left-hand side of the *within* operator. • The range is on the right-hand-side of a *throughout* operator. Note:The rule is not checked on cover property statements.
Language	SystemVerilog
Severity	Lint

Example

Detected cases (including *not* over the property or the consequent):

```
a |-> ##[0:$] seq;
a |-> b[->N];
a |-> b[=N]
```

Accepted constraint using intersect even if seq1 contains an unbounded delay range:

```
a |-> ##[0:$] seq intersect seq1;
a |=> a[=1] within seq;
a|=> b throughout seq;
```

where, seq contains an open-ended delay or [=] or [->].

PROPERTY014

Message: Negated suffix implication without negated consequent

Description	SVA Checker issues this message when it detects a negated suffix implication without negated consequent. Property of the form *not (s1 |-> s2)* may have unexpected interpretation. You need to reconsider its usage. Note:The form *not (s1 |-> not p)* may be useful in cover property statements, since it corresponds to s1 being followed by the property p.
Language	SystemVerilog
Severity	Lint

Example

The following example shows SystemVerilog code that flags this rule:

```
module PROPERTY014;
bit clk, clk1, rst, a, b, c, d;

generate
begin : pass
     a1: assert property (@clk  not( ##[1:$]a |->
         not(##[1:10]b)));

     a4: assume property (@clk not (a ##1 b) ##[0:2]
         (a throughout (a intersect b[=1])));

end
begin : fail
     a1: assert property (@clk not (##[1:2] a |-> ##[0:$]b));
     // rule PROPERTY014 is flagged in the above line

     a4: assume property (@clk not ((a ##1 b) ##[0:2] 1'b1 |=>
         (a throughout (a intersect b[=1]))));
     // rule PROPERTY014 is flagged in the above line

end

endgenerate

endmodule
```

SEQUENCE

The rules listed under this category checks the structure of sequences.

SEQUENCE001

Message: A large finite upper bound on range

Description	SVA Checker issues this message when it finds a large finite upper bound on range (>50). It checks for bound exceeding 50 clock cycles in ## cycle delay or [*], [=] or [->] repetitions. In simulation, try to reformulate using a local variable for counting clock cycles. For use with a hybrid formal tool, consider using a global variable to count, provided only one evaluation attempt is active at any time. Otherwise, you may have to consider using a FIFO of time stamps.
Language	SystemVerilog
Severity	Lint

Example

The following example shows SystemVerilog code that flags this rule:

```
module SEQUENCE001;
bit clk, clk1, rst, a, b, c, d;

sequence s1(x, y);
     (x ##[1:y] b) or (b ##1 c);
endsequence // s1

generate
begin : pass

     a11: assert property (@clk s1(a, 50) |-> s1(a, 50));
```

```
    a12: cover property (@clk s1(a, 50));

end

begin : fail
    a11: assert property (@clk s1(a, 2) |->
     ((((a ##[1:51] b[=51])[*1:2]) and c) or (a ##51 b)));
    // rule SEQUENCE001 is flagged in the above line

    a12: cover property (@clk s1(a, 51) );
    // rule SEQUENCE001 is flagged in the above line

end

endgenerate

endmodule
```

SEQUENCE002

Message: Use Boolean expression in place of $rose or $fell

Description	This check detects cases like *a[*M:N] ##1 $fell(a), !a[*M:N] ##1 $rose(a)*, where M, N can be 1, i.e., no range, the use of a sampled-value function is not necessary. Similarly in cases like *##[M:N] a ##1 $fell(a), ##[M:N] !a ##1 $rose(a), $rose(a) |=> $fell(a)*, etc. are nested within other operators. The simplification is to replace the last sampled function by the argument with the right polarity. E.g., *$rose(a) ##1 !a* instead of *$rose(a) ##1 $fell(a).*
Language	SystemVerilog
Severity	Lint

Example

The following example shows SystemVerilog code that flags this rule:

```
module SEQUENCE002;
bit clk, a, b, c;

generate
begin : pass
     a1: assert property (@clk !a[*1:2]##1 $fell(a));
     a2: assert property (@clk a |=> a ##1 $rose(a));
end

begin : fail
```

```
    a1: assert property (@clk !a[*1:2] ##1 $rose(a));
    // rule SEQUENCE002 is flagged in the above line


    a2: assert property (@clk a |=> a ##1 $fell(a));
    // rule SEQUENCE002 is flagged in the above line

end

endgenerate

endmodule
```

SEQUENCE003

Message: Last boolean in s1 and the first one in s2 contradict each other in s1 ##0 s2.

Description	s1 ##0 s2 contradict each other on the overlapping boolean expression. The sequence cannot ever match. For example, it detects the following situations: *a ##0 !a; a ##0 $fell(a) ; $rose(a) ##0 $fell(a); a && b ##0 !(a && b);* etc. also nested in other operators.
Language	SystemVerilog
Severity	Lint

Example

The following example shows SystemVerilog code that flags this rule:

```
module SEQUENCE003;
```

```
bit clk, clk1, a, b, c;

generate
begin : pass
     a1: assert property (@clk !a[*1:2]##0 $fell(a));
     a2: assert property (@clk a |=> a ##0 $rose(a));
end

begin : fail
     a1: assert property (@clk !a[*1:2] ##0 $rose(a));
     // rule SEQUENCE003 is flagged in the above line

     a2: assert property (@clk a |=> a ##0 $fell(a));
     // rule SEQUENCE003 is flagged in the above line

end
endgenerate

endmodule
```

SEQUENCE004

Message: Use *goto* with *intersect* to limit allowed length of *until* operation.

Description	Replace *!expr[*M:N] ##1 expr by 1'b1[*M+1:N+1] intersect expr[->1]* because its implementation is usually more efficient.
Language	SystemVerilog
Severity	Lint

Example

Detected cases (including *not* over the property or the consequent):

a) simple bounded until

```
a[*2:6] ##1 !a
replace by
1'b1[*3:7] intersect !a[->1]
```

b) unbounded until

```
!a[*0:$] ##1 a
replace by
a[->1]
```

c) bounded until combined with sampled value function

```
a[*1:3] ##1 $fell(a)
replace by
1'b1[*1:3] intersect !a[->1]
```

SEQUENCE005

Message: A sequence is declared but never used.

Description	SVA Checker issues this message when it finds a sequence defined in a module but: • it is not instantiated in a verification statement or • instantiated in a property or sequence, which is then instantiated in a verification statement or • used with .ended or matched or triggered, or in an event control block. Note:Enable rule PROPERTY001 to detect an unused property.
Language	SystemVerilog
Severity	Lint

Example

The following example shows SystemVerilog code that flags this rule:

```
module SEQUENCE005;
bit clk, clk1, clk2, rst, a, b, c, d;

sequence pass_s1;
    @clk a ##1 b;
endsequence // pass_s1

c1: cover property (@clk pass_s1.ended);

sequence fail_s2; // rule SEQUENCE005 flags here
    @clk b;
endsequence // fail_s2

endmodule
```

SEQUENCE006

Message: *first_match* is used on a sequence.

Description	SVA Checker issues this message when it finds the operator *first_match* is applied to a sequence. This may increase compilation time in some cases. You need to consider rewriting the sequence in a form that admits only a single match and remove *first_match*.
Language	SystemVerilog
Severity	Lint

Example

Assuming that b is a boolean expression, M<=N and N can be $,

```
first_match (a ##[M:N] b)
```

can be replaced by

```
a ##1 (1'b1[*M:N] intersect b[->1])
```

SEQUENCE007

Message: Possible contradiction in a boolean expression.

Description	SVA Checker flags this message when it finds contradiction in a boolean expression. A boolean expression contains the equivalent of *x && !x or x & !x* at the top level of the expression. x can be any expression and the two occurrences must match exactly and ! can also be replaced by ~. Sampled value functions *$rose* and *$fell* are also taken into account and the check compensates for negated operands and opposing functions. For example, *$rose(x) && $fell(x) and $rose(x) && $rose(!a), $fell(a) && a* are flagged.
Language	SystemVerilog
Severity	Lint

Example

The following example shows SystemVerilog code that flags this rule:

```
module SEQUENCE007;
bit clk, clk1, a, b, c;

generate
begin : fail
     a1: assert property (@clk a && !a);
     // rule SEQUENCE007 is flagged in the above line

     a2: assert property (@clk a & !a |=>
```

```
        (a && !a[*0:2] ##1 (a&b)&~(a&b)));
    // rule SEQUENCE007 is flagged in the above line

end

endgenerate

endmodule
```

SEQUENCE008

Message: Possible tautology in a boolean expression

Description	SVA Checker flags this message when it finds tautology in a boolean expression. A boolean expression contains the equivalent of *x* || *!x or x* | *!x* at the top level of the expression. x can be any expression and the two occurrences must match exactly and, ! can also be replaced by ~. Sampled value functions *$rose* and *$fell* are also considered and the check compensates for negated operands and opposing functions. For example, *x* || *!$rose(x), !$fell(x)* || *!x, !$fell(x)* \|\| *!$fell(!x) and !$fell(x)* \|\| *!$rose(x)* are detected as tautology.
Language	SystemVerilog
Severity	Lint

Example

The following example shows SystemVerilog code that flags this rule:

```
module SEQUENCE008;
bit clk, clk1, a, b, c;

generate
begin : fail
     a1: assert property (@clk a || !a);
     // rule SEQUENCE008 is flagged in the above line

     a2: assert property (@clk ~a | a |=>
```

```
        (a || !a[*0:2] ##1 (a&b) | ~(a&b)));
    // rule SEQUENCE008 is flagged in the above line

end

endgenerate

endmodule
```

SEQUENCE009

Message: Possibly redundant operands in Boolean expression.

Description	SVA Checker flags this message when it finds redundant operands in a Boolean expression. A Boolean expression contains the equivalent of *x || x or x && x or x | x or x & x* at the top level of the expression. x can be any expression and the two occurrences must match exactly and ! can also be ~. Sampled value functions *$rose* and *$fell* are also taken into account and the check compensates for negated operands and opposing functions. For example, *$rose(x) || $fell(!x) and $rose(x) && $fell(!a), $fell(a) && !a* are detected.
Language	SystemVerilog
Severity	Lint

Example

The following example shows SystemVerilog code that flags this rule:

```
module SEQUENCE009;
bit clk, clk1, a, b, c;

generate
begin : pass
     a1: assert property (@clk a && !a);
     a2: assert property (@clk a && !a |=>
```

```
        (a && !a[*0:2] ##1 (a&b) | !(a&b)));
end
begin : fail
    a1: assert property (@clk a && a);
    // rule SEQUENCE009 is flagged in the above line

    a2: assert property (@clk a && a |=>
        (a || a[*0:2] ##1 (a&b)|(a&b)));
    // rule SEQUENCE009 is flagged in the above line

end

endgenerate

endmodule
```