Conformal® HDL Rule Check Reference

Product Version 23.2
October 2023

Contents

1 HDL Rule Check Messages

This book is organized according to HDL rule categories. Each rule message is defined and followed by a sample case that can cause the message. The examples include a brief explanation that direct you to the pertinent lines of code.

The rule categories included in this chapter are listed below. Register Transfer Level messages are a super-set of rules that apply to both Verilog and VHDL. However, rules included in the Verilog category apply to Verilog only.

Within each category, rules are grouped in sets according to their relationship to each other. For example, DIR1.1 and DIR1.2 both relate to pragmas. Likewise, IGN3.1 and IGN3.2 are grouped because they both address how Conformal handles duplications.

Checkpoint and Restart

This category of rules applies to the use of the checkpoint and restart facility.

- CNR1.1 on page 21
- CNR1.2 on page 22
- CNR1.3 on page 23
- CNR1.4 on page 24
- CNR1.5 on page 25

CNR1.1

Message

```
Inconsistent host was used to restart the checkpoint file. It can cause unexpected
     results
```

Default Severity

Warning

Description

Checkpoint files should be created and restarted using the same host. The tool issues warnings when you try to restart a checkpoint file from a different host as this can cause unexpected results.

The warning message can be ignored when the restart is always successful. For example, in a homogeneous virtual machine computation environment, the host name might be different, but the OS version and configuration are the same. Thus, the warning message can be ignored.

CNR1.2

Message

```
Unsupported platform was used to checkpoint a process. It can cause unexpected
     results.
```

Default Severity

Warning

Description

Checkpoint and restart works on only Linux, and only on the following platforms: 32/64-bit Linux kernel versions 2.6.9-34, 2.6.9-42, 2.6.9-55, 2.6.9-67, 2.6.9-78, 2.6.9-89, 2.6.10, 2.6.14, 2.6.16, 2.6.18, 2.6.25, 2.6.26, 2.6.27, 2.6.32, 3.0.101, 3.0.13, 3.10.0-957, 3.10.0-1062, 4.4.21-69, 4.18.0-147, 4.18.0-193, 4.18.0-240, 4.18.0-305, 4.18.0-348, 4.18.0-372, 5.3.18-24, and 5.14.21-150400.

CNR1.3

Message

```
Unsupported platform was used to restart a process. It can cause unexpected results.
```

Default Severity

Warning

Description

Checkpoint and restart works on only Linux, and only on the following platforms: 32/64-bit Linux kernel versions 2.6.9-34, 2.6.9-42, 2.6.9-55, 2.6.9-67, 2.6.9-78, 2.6.9-89, 2.6.10, 2.6.14, 2.6.16, 2.6.18, 2.6.25, 2.6.26, 2.6.27, 2.6.32, 3.0.101, 3.0.13, 3.10.0-957, 3.10.0-1062, 4.4.21-69, 4.18.0-147, 4.18.0-193, 4.18.0-240, 4.18.0-305, 4.18.0-348, 4.18.0-372, 5.3.18-24, and 5.14.21-150400.

CNR1.4

Message

```
Generating a checkpoint file for a process where the GUI mode was previously enabled
```

Default Severity

Warning

Description

To avoid unexpected results, do not enable GUI mode in a process for which you plan to create a checkpoint file. Entering and exiting the GUI mode can cause unexpected results when you try to restart the checkpoint file.

CNR1.5

Message

```
Restarting a process where GUI mode was enabled
```

Default Severity

Warning

Description

This message warns you that the process you are about to restart was originally performed in GUI mode. However, since the GUI mode is automatically disabled when you use the `-restart_checkpoint` option at startup, the tool will continue this process in console mode.

To avoid this message, do not enable the GUI mode in a process for which you plan to create a checkpoint file. Entering and exiting the GUI mode can cause unexpected results when you try to restart the checkpoint file.

Directive

This category of rules applies to designs that include directives or pragmas. The following lists the Directive (DIR) rule checks.

- DIR9.2 on page 53
- DIR9.3 on page 55
- DIR9.4 on page 57

DIR1.1

Message

`built_in pragma applied to function`

Default Severity

Warning

Description

The design applies the `built_in` pragma to a function. If Conformal supports the specified pragma, it also displays the rule DIR4.2 message.

Example

In the following example, pragma `built_in` is inserted in function `my_or`. See line 4.

```
ARCHITECTURE arch OF test IS
    FUNCTION my_or  ( l, r : bit )
    RETURN bit IS
    -- pragma built_in SYN_OR
    BEGIN
      RETURN '0';
    END my_or;
BEGIN
  proc2 : PROCESS
    BEGIN
      out0 <= my_or (in1,in2);
    END PROCESS;
END arch;
```

DIR1.2

Message

`map_to_operator pragma applied to function`

Default Severity

Warning

Description

The design applies the `map_to_operator` pragma to a function. If Conformal supports the specified pragma, it also displays the rule DIR4.2 message.

Example

In the following example, the `map_to_operator` pragma is inserted in function `my_leq`. See line 5.

```
ARCHITECTURE arch OF test IS
    FUNCTION my_leq  ( l, r : bit_vector
    (3 downto 0) )
    RETURN boolean IS
    -- pragma map_to_operator LEQ_TC_OP
    BEGIN
      RETURN false;
    END my_leq;
BEGIN
  proc2 : PROCESS
    BEGIN
      out0 <= my_leq (in1,in2);
    END PROCESS;
END arch;
```

DIR1.3

Message

`return_port_name pragma applied to function`

Default Severity

Warning

Description

The design applies the `return_port_name` pragma to a function that contains this pragma. If Conformal supports the specified pragma, it also displays the rule DIR4.2 message.

Example

In the following example, the `return_port_name` pragma, on line 8, is inserted in function `DWF_div_uns`.

```
module test;
  parameter a_width = 16;
  parameter b_width = 16;

    function [a_width-1 : 0] DWF_div_uns;
       // Function to compute the unsigned quotient

       // pragma map_to_operator DIV_UNS_OP
       // pragma return_port_name QUOTIENT

       input [a_width-1 : 0] A;
       input [b_width-1 : 0] B;

       begin
         // pragma translate_off
            return 0;
         // pragma translate_on
      end
    endfunction
  endmodule
```

DIR2.1

Message

`full_case directive is detected`

Default Severity

Warning

Description

This message indicates there is a case statement that contains a `full_case` directive. If this directive is currently supported by Conformal, the DIR4.2 message will also be reported.

Conformal warns about the presence of the `full_case` directive, because this directive informs the synthesis tool to treat all outputs assigned within the case statement as synthesis don't-care assignments.

Rule check will also be issued.

Example

The following example uses the synthesis `full_case` directive in line 7.

```
module test (a, e, sel, out0);
  input a, e;
  input sel;
  output out0;
  reg out0;
    always @(sel or a or e) begin
       case(sel) // synthesis full_case
         1’b1 : out0 = a;
         default: out = e;
       endcase
    end
endmodule
```

DIR2.2

Message

`parallel_case directive is detected`

Default Severity

Warning

Description

The design includes a `parallel_case` directive used in a case statement. If this directive is currently supported by Conformal, the DIR4.2.

Conformal warns about the presence of the `parallel_case` directive, because this directive informs the synthesis tool to test all case items, even when more than one case item can possibly match the case expression.

Example

The following example includes the `synthesis parallel_case` directive on line 7.

```
module test ( a, e, sel, out0);
input  a, e;
input  sel;
output out0;
reg out0;
  always @(sel or a or e) begin
    case(sel) // synthesis parallel_case
      1'b1   : out0 = a;
      default: out0 = e;
    endcase
  end
endmodule
```

DIR3.1

Message

`synthesis/translate/compile on/off directive is detected`

Default Severity

Warning

Description

The design includes a directive belonging to one of the following classes:

- `synthesis on/off`
- `translate on/off`
- `compile on/off`

If Conformal supports this directive, it also displays the rule DIR4.2 message.

In the Conformal Equivalency Checker, you can also define your own directive to turn synthesis on and off with the following commands:

- `SET_ATTR INPUT_PRAGMA_KEYWORD <string>`
- `SET SYNTHESIS_OFF_COMMAND <string>`
- `SET SYNTHESIS_ON_COMMAND <string>`

Example

The following example includes the `synthesis translate_off` and `synthesis translate_on` directives. See lines 5 and 7.

```
module test ( clk, din, dout );
input  clk, din;
output dout;
reg dout;
// synthesis translate_off
reg   [0:0] mem [1:0];
// synthesis translate_on
always @(posedge clk)
 dout <= din;
endmodule
```

DIR3.2

Message

```
Unsupported compiler directive script is detected
```

Default Severity

Warning

Description

The design includes directives for vendor specific scripts. If Conformal supports the specified directives, it also displays the rule DIR4.2 message.

DIR4.1

Message

```
Design includes one or more HDL directives or pragmas that Conformal does not
     support
```

Default Severity

Warning

Description

Indicates that the design contains an unsupported HDL or pragma. For a list of supported pragmas/directives, refer to the "Supported Directives" chapter of the *Conformal Equivalence Checking User Guide*.

Example

In the following example, Conformal does not support the `synthesis not_defined_or_not_supported` directive. See line 6.

```
module test ( clk, in0, out0 );
input  clk, in0;
output out0;
reg out0;
  always @( posedge clk )
    // synthesis not_defined_or_not_supported
    out0 <= in0;
endmodule
```

DIR4.2

Message

```
Design includes one or more HDL directives or pragmas that Conformal supports
```

Default Severity

Warning

Description

The design includes one or more HDL directives or pragmas that Conformal supports.

Example

In the following example, Conformal supports the `synthesis full_case` directive. See line 7.

```
module test ( a, e, sel, out0);
input  a, e;
input  sel;
output out0;
reg out0;
  always @(sel or a or e) begin
    case(sel) // synthesis full_case
      1'b1 :
        out0 = a;
      default:
        out0 = e;
    endcase
  end
endmodule
```

DIR4.3

Message

```
Design includes one or more HDL directives or pragmas disabled by the user
```

Default Severity

Warning

Description

The design includes one or more HDL directives or pragmas that were disabled by the user. The command used to disable a directive or pragma is:

```
set directive off <directive_name>
```

Example

In the following example, the design includes the `conformal assertion_library` directive on line 5. However, we disabled it with the `SET DIRECTIVE OFF` command.

```
module test (a, b, q);
input [3:0] a, b;
output [3:0] q;
reg [3:0] q;
// conformal assertion_library
always @ (a or b ) begin
  q = a & b;
end
endmodule
```

Input:

```
SETUP>set directive off assertion_library
SETUP>read design test.v -verilog
```

DIR4.4

Message

```
Design includes one or more HDL directives or pragmas that Conformal ignores
```

Default Severity

Warning

Description

The specified HDL directive or pragma is not supported and will be ignored.

Example

In the following example, if the dofile contains the following line:

```
set directive off  translate_on translate_off unknown_pragma
```

The `unknown_pragma` is not a supported directive or pragma and will be ignored.

DIR5.1

Message

`Conformal directive multi_port is detected`

Default Severity

Warning

Description

Conformal generates this message when the design includes the Conformal `multi_port` directive. This directive specifies that Conformal will model the specified register with multiple clock ports, multiple data ports, and a single output port.

Note: If you do not include the `multi_port` directive, by default, Conformal creates multiple registers with outputs wired together.

Example

In the following example, the design includes the `conformal multi_port` directive in line 5.

```
module test ( clk1, clk2, in0, out0 );
input  clk1, clk2, in0;
output out0;
reg out0;
  // conformal multi_port "out0"
  always @( posedge clk1 )
    out0 <= 1'b0;
  always @( posedge clk2 )
    out0 <= in0;
endmodule
```

DIR5.2

Message

`Conformal directive clock_hold is detected`

Default Severity

Warning

Description

Conformal generates this message when the design includes the Conformal `clock_hold` directive. This directive specifies that Conformal will model the selected registers as gated-clock registers.

Example

In the following example, the design includes the `conformal clock_hold` directive on line 5.

```
module test ( clk, din, adr, out0 );
input  clk, din, adr;
output out0;
reg   [0:0] mem [1:0];
   // conformal clock_hold mem
   always @(posedge clk)
   begin
      mem[adr] <= din;
   end
   assign out0 = mem[adr];
endmodule
```

DIR5.3

Message

```
Conformal directive mem_rowselect is detected
```

Default Severity

Warning

Description

Conformal generates this message when the design includes the Conformal `mem_rowselect` directive. This directive specifies that Conformal will model the following type of RTL into a structure that lets you define the signals that are synthesized as part of the decoded word line of the `ram_array`, while all other signals on the sensitivity list are synthesized into the logic cone of the input bit line.

```
always @(clk or we or din or addr)
begin
  if (clk && we) ram_array(addr) = din;
end
```

Example

In the following example, the design includes the `conformal mem_rowselect` directive on line 7.

```
module test(clk, en, addr, din, dout);
input clk, en;
input [1:0]  addr;
input [0:0]  din;
output [0:0] dout;
reg   [0:0]  ram [3:0];
    // conformal mem_rowselect "ram clk en addr[1]"
    always @(clk or en or addr or din) begin
        if (clk && en) ram[addr[1:0]] = din;
    end
    assign dout = ram[addr[1:0]];
endmodule
```

DIR5.4

Message

```
Conformal cutpoint directive is supported
```

Default Severity

Warning

Description

The variable defined by the pragma is the cut point in the design.

Example

In the following example, `ram` is defined as the cutpoint.

```
module test(clk, en, addr, din, dout);
input clk, en;
input [1:0]  addr;
input [0:0]  din;
output [0:0] dout;
reg   [0:0]  ram [3:0];
// pragma cutpoint "ram"
always @(clk or en or addr or din) begin
  if (clk && en & ram[0] ) ram[addr[1:0]] = din;
end
assign dout = ram[addr[1:0]];
endmodule
```

DIR6.1

Message

`Ignored compiler directive is detected`

Default Severity

Warning

Description

Conformal detected the use of an ignored compiler directive. Below is a full list of ignored compiler directives:

```
`accelerate

`autoexpand_vectornets

`default_decay_time

`default_strength

`delay_mode

`delay_mode_distributed

`delay_mode_unit

`delay_mode_zero

`end_pre_16a_paths

`expand_vectornets

`line

`noaccelerate

`noexpand_vectornets

`noremove_gatenames
```

`nounconnected_drive

`pre_16a_paths

`remove_gatenames

`remove_names

`switch

`timescale

`unconnected_drive

`unprotected

`uselib

Example

In the following example, the Verilog RTL accepts compiler directives `` `celldefine `` and `` `endcelldefine `` on lines 1 and 8. All modules declared after celldefine and before `` `endcelldefine `` are marked as ASIC library cells. Currently only Conformal Constraint Designer uses this marking for finding timing paths.

```
`celldefine
module cell1 (a, b);
input a;
output b;

buf (b, a);
endmodule
`endcelldefine
```

DIR6.2

Message

```
Supported compiler directive is detected
```

Default Severity

Warning

Description

Conformal detected a supported compiler directive. Below is a full list of supported compiler directives:

```
`celldefine and `endcelldefine

`define

`ifdef, `ifndef, `elsif, `endif

`undef

`include "<filename>"

`protect   and `endprotect

`protected and `endprotected

`default_nettype net_type

`resetall
```

Example

In the following example, Conformal supports compiler directive `` `default_nettype `` wire on line 1.

```
`default_nettype wire
module test (din, dout);
input din;
output dout;
```

```
assign dout = din;
endmodule
```

DIR6.3

Message

```
Conditional compiler directive clause is effective
```

Default Severity

Note

Description

Indicates that a conditional compilation block is active.

Example

In the example below, the macro 'FOO is defined. Therefore, the `` `ifdef FOO `` clause on line-4 and the `` `else `` clause on line-7 evaluate to true.

```
1  `define FOO
2
3  module test(input in, output out);
4  `ifdef FOO           // DIR6.3
5  `ifdef BAR
6      assign out bar;
7  `else                // DIR6.3
8      assign out = in;
9  `endif
10  `else
11      assign out = bar;
12  `endif
13  `endif
14  endmodule
```

DIR7.1

Message

```
Unsupported synthesis attribute is detected
```

Default Severity

Warning

Description

Conformal detected a synthesis attribute that it does not support.

Example

In the following example, Conformal ignores the attribute `MAX_DELAY`, rendering it ineffective (see lines 9 and 10).

```
entity TEST is
  port(
    IN1  : in integer;
    OUT1 : out bit
  );
end test;

architecture RTL of TEST is
  attribute MAX_DELAY : string;
  attribute MAX_DELAY of TEST: entity is "1431";
begin
  OUT1 <= '1' when IN1 > 329 else '0';
end RTL;
```

DIR7.2

Message

`Supported synthesis attribute is detected`

Default Severity

Warning

Description

The design includes a synthesis attribute that Conformal supports.

Example

The following example defines the attribute `ENUM_ENCODING` to specify the encoding of the enumeration type (see lines 10 and 11).

```
entity TEST is
  port(
    SEL : in bit_vector(2 downto 0);
    OUT1 : out bit
  );
end test;

architecture RTL of TEST is
  type ET is (ET1, ET2, ET3);
  attribute ENUM_ENCODING : string;
  attribute ENUM_ENCODING of ET : type is "00 01 11";
  signal SIG1 : ET;
begin
  process (SEL)
  begin
    if (SEL = "000") then
      SIG1 <= ET1;
    elsif (SEL = "111") then
      SIG1 <= ET2;
    else
      SIG1 <= ET3;
    end if;
  end process;
  OUT1 <= '1' when (SIG1 = ET3) else '0';
end RTL;
```

DIR8.1

Message

`protected/endprotected directive is detected`

Default Severity

Warning

Description

The content after the pragma is protected.

Example

In the following example, line 3 is protected because it is between the `'protected` and `'endprotected` directives:

```
Pragma `protected/`endprotected used
primitive FOO_P `protected
SOME98098jlkajdGOO
`endprotected endprimitive
```

DIR9.1

Message

```
Verilog `celldefine is for timing simulation. Use the LIBERTY library instead
```

Default Severity

Ignore

Description

A Verilog module enclosed between `` `celldefine `` and `` `endcelldefine `` is recognized as a technology library cell. Conformal Constraint Designer prefers using technology cells in the LIBERTY format, and reports a rule check violation. The default severity for this rule check is `ERROR`. You can override the severity with the `SET RULE HANDLING` command.

Example

If the following module is read using `READ DESIGN` or `READ LIBRARY` command, the Conformal software reports this rule check error by default.

```
`celldefine
module TECH_CELL_BUF (Y, A);
output Y;
input A;
  buf I0(Y, A);
specify
  specparam
    tplh$A$Y = 1.0,
    tphl$A$Y = 1.0;
  (A *> Y) = (tplh$A$Y, tphl$A$Y);
endspecify
endmodule // TECH_CELL_BUF
`endcelldefine
```

DIR9.1a

Message

```
Unbalanced `celldefine and ‘endcelldefine directives in a Verilog file
```

Default Severity

Warning

Description

The Verilog file contains unbalanced `` `celldefine `` and `` `endcelldefine `` directives which might cause unexpected annotations of modules as cells. Normally each `` `celldefine `` is expected to have a matching `endcelldefine in the same Verilog file.

Example

In the following example, the `celldefine at line 1 and line 8 do not have matching `endcelldefine directives in the same Verilog file.

```
// file1.v
1 `celldefine
2  module mod1(state, data);
3  input state;
4  output data;
5  // some statements
6  endmodule
7  // expect to have a `endcelldefine here to match line 1
8 `celldefine
9  module mod2(state, data);
10  input state;
11  output data;
12  // some statements
13  endmodule
14  // expect to have a `endcelldefine here to match line 8
```

DIR9.2

Message

`Invalid to redefine a Verilog compiler directive as a macro`

Default Severity

Warning

Description

You have attempted to redefine a Verilog compiler directives that is one of the following:

```
`celldefine

`default_nettype

`define

`else

`endcelldefine

`endif

`ifdef

`ifndef

`include

`nounconnected_drive

`resetall

`timescale

`unconnected_drive

`undef
```

Example

The following `` `define `` statement (see line 1) causes a DIR9.2 rule violation:

```
‘define timescale 1'b0
module test (in,out);
input in;
output out;
assign out = in;
endmodule
```

DIR9.3

Message

`Verilog compiler directive is redefined as a macro`

Default Severity

Warning

Description

You have attempted to redefine other Verilog compiler directives that are not as follows:

```
`celldefine

`default_nettype

`define

`else

`endcelldefine

`endif

`ifdef

`ifndef

`include

`nounconnected_drive

`resetall

`timescale

`unconnected_drive

`undef
```

Example

The following `` `define `` statement (see line 1) causes a DIR9.3 rule violation:

```
'define switch 1'b1
module test (in,out);
input in;
output out;
assign out = in;
endmodule
```

The above statement causes the `` `switch `` directive to be deleted and replaced by a user-defined macro.

DIR9.4

Message

`Verilog compiler directive cannot be used inside a module`

Default Severity

Warning

Description

The following Verilog compiler directives cannot be used inside a module:

- `` `timescale ``
- `` `unconnected_drive ``
- `` `default_decay_time ``

Note: This rule violation can trigger a syntax error or unintended behavior in other EDA tools.

Example

In line 9 of the following example, use of the `` `timescale `` directive triggers the DIR9.4 rule violation:

```
// file t1.v
1 module top (in1, out1);
2 input in1;
3 output out1;
4   test u1 (.in1(in1), .out1(out1));
5 endmodule
6 module test (in1, out1);
7 input in1;
8 output out1;
9 `timescale 1ns/10ps // this should trigger DIR9.4 warning
10   assign out1 = ~in1;
11 endmodule
```

File

This category of rules applies to designs that include file issues. The following lists the File issue (FIL) rule checks.

FIL1.1

Message

```
Input file recursion detected
```

Default Severity

Error

Description

This detects that input files are included recursively.

Example

The two files in the following example are recursively included:

```
In file file1.v, has '`include "file2.v"'.
In file file2.v, has '`include "file1.v"'.
```

FIL1.2

Message

```
Failed to open file
```

Default Severity

Error

Description

This is triggered when the software attempts try to read in a file that does not exist or cannot be opened.

Example

For the following command:

```
read design test.v
```

If file 'test.v' does not exist, `FIL1.2` will be reported.

FIL1.3

Message

```
Failed to open ROM code file
```

Default Severity

Error

Description

This is triggered when the software attempts to read in a code file that does not exist or cannot be opened.

Example

For the following command:

```
generate rom primitive -code_file my_code_file
```

FIL1.3 is reported if code file `my_code_file` does not exist.

FIL1.4

Message

```
File added by automatic package search
```

Default Severity

Warning

Description

This is triggered when the software adds a file due to automatic package searching. To turn off automatic package searching, use the following command:

```
SET HDL OPTIONS -autopkgsearch off
```

Example

In test.vhd:

```
LIBRARY zb_mpep ;
USE zb_mpep.ep_pack.all;
```

However, the dofile reads in `only test.vhd`. LEC will look for the referenced packages and (`ep_pack.vhd` or `ep_pack.vhdl)` in the working directory, and read in `ep_pack.vhd` as well.

FIL1.5

Message

```
Failed to open Liberty include_file
```

Default Severity

Error

Description

This is triggered when the tool cannot find the file specified by `include_file`.

Example

In the following example, FIL1.5 is issued if the tool cannot find `nest.lib`:

```
library(test_lib){

...

include_file(nest.lib);
...

}
```

FIL1.6

Message

```
Unsupported encoding data detected
```

Default Severity

Error

Description

Triggered when the VHDL design contains unsupported encoding data, which will be ignored during parsing.

FIL1.7a

Message

```
File in revised has earlier timestamp than design file(s) in golden
```

Default Severity

Note

Description

File(s) in the golden design is more recent than the file read for the revised design. The revised design might be out-dated with respect to the golden design.

Note1: This rulecheck message will only be issued if the following hdl option is specified: set hdl option -FILE_TIMEstamp_check on

Note2: By default, up to 5 occurrences of FIL1.7a violations will be reported. The command 'set_rule_handling' command '-LIMIT <n>' option can be used to change the maximum number of occurrances reported.

FIL1.7b

Message

```
File in revised has earlier timestamp than library file(s) in golden
```

Default Severity

Note

Description

File(s) in the golden design library is more recent than the file read for the revised design. The revised design might be out-dated with respect to the golden design.

Note1: This rulecheck message will only be issued if the following hdl option is specified: set hdl option -FILE_TIMEstamp_check on

Note2: By default, up to 5 occurrences of FIL1.7b violations will be reported. The command 'set_rule_handling' command '-LIMIT <n>' option can be used to change the maximum number of occurrances reported.

FIL1.8

Message

```
File read in multiple times
```

Default Severity

Note

Description

Indicates that file with the same absolutate path is read more than once into the same space which would consume extra memory and runtime.

FIL1.9

Message

```
Duplicated liberty filenames skipped
```

Default Severity

Note

Description

Indicates that the liberty files with the same name show up more than once in a single command. If the option `-lastmod` is not specified, LEC will skip parsing these duplicated filenames to save memory and runtime.

Example

When running the following command:

```
read library -liberty  ./STD.lib ./STD.lib ./STD.lib
```

LEC will report 2 occurrences for the rule `FIL1.9`.

Hierarchy

This category of rules applies to designs that are comprised of hierarchical modules. The following lists the Hierarchy (HRC) rule checks.

- HRC9 on page 128
- HRC10.1 on page 130
- HRC10.2 on page 131
- HRC10.4 on page 133
- HRC10.5 on page 134
- HRC10.6 on page 135

HRC1.1

Message

```
A module/entity that contains an unsupported construct cannot be loaded. Conformal
     blackboxes modules/entities that contain unsupported constructs.
```

Default Severity

Warning

Description

The design includes one or more modules or entities that contain unsupported constructs. Conformal will blackbox all modules and entities that contain unsupported constructs.

Example

In the following example, module "adder" has one parameter "W" declared and is instantiated with two parameters "W" and "W2". This usage is unsupported; module "top" will be blackboxed.

(see lines 2 and 11).

```
1  module adder (in, out);
2    parameter W = 8; // one parameter declaration
3    input [W-1:0] in;
4    output [W-1:0] out;
5      assign out = in;
6  endmodule
7  module top (in, out);
8  parameter W = 8;
9  input [W-1:0] in;
10  output [W-1:0] out;
11    adder #(.W(W), .W2(W)) a1 (.in(in), .out(out)); // two parameter association
12  endmodule
```

HRC1.2

Message

```
Module/entity is not translated (blackboxed)
```

Default Severity

Note

Description

The design includes one or more modules or entities that were blackboxed with the command:

```
add notranslate module <module_name>
```

Example

Input:

```
SETUP> add notranslate module SUB
SETUP> read design SUB.v -verilog
// Note: (HRC1.2) Module/entity not translated (black boxed) (occurrence:1)
```

HRC1.3

Message

`Undefined or non-translated modules will be blackboxed`

Default Severity

Note

Description

Notes that Conformal will blackbox design modules that are either:

- Referenced, but not defined. This is triggered by the `SET UNDEFINED CELL <blackbox>` command, which specifies that all undefined modules will be created and blackboxed.
- Non-translated. This is triggered by the `ADD NO TRANSLATE MODULES` command, which specifies particular modules to be no translate and blackboxed.

Example

In the following example, the design references module `SUB` on line 9, but it is not defined.

```
module TEST (in0,in1,in2,out0);
input in0,in1,in2;
output out0;
reg    out0;
  always @ ( in0 or in1 )
    begin
      out0 = in0 | in1;
    end
  SUB inst1 (.in2(in2));
endmodule
```

HRC1.3a

Message

```
VHDL Entity is undefined and created (blackboxed)
```

Default Severity

Note

Description

Indicates that blackboxing was specified for VHDL design modules that are referenced, but not defined. This message can be generated by the following commands:

```
SET UNDEFINED CELL <blackbox>
ADD NOTRANSLATE MODULES
```

The `ADD NOTRANSLATE MODULES` command causes undefined modules to be blackboxed.

Example

Conformal this rule check for the following RTL:

```
architecture structure is

component undefined_module
  port (a,b: in bit;
     c: out bit);
end component;
begin

n1: undefined_module
  port map (r,nq,q);
n2: undefined_module
  port map (s,q,nq);
end structure;
```

HRC1.4

Message

`Module/entity is empty (blackboxed)`

Default Severity

Warning

Description

The design includes one or more empty modules or entities. Conformal blackboxes all empty modules.

Example

In the following example, Conformal blackboxes the empty `SUB` module.

```
module SUB (in2, out3);
input in2;
output out3;
endmodule
```

HRC1.5

Message

```
Module/entity is referenced recursively (blackboxed)
```

Default Severity

Warning

Description

The design includes one or more modules that are referenced recursively. Conformal blackboxes all recursively referenced modules.

Example

In the following example, module `abc` is referenced recursively and will be blackboxed (see line 4).

```
module abc (a,b);
input a;
output b;
abc gen1_x1 (a,b);
endmodule
```

HRC1.6

Message

`Module/entity that does not use an input pin has been blackboxed`

Default Severity

Warning

Description

The `-bbox_module_with_no_pi_used` option of the `SET HDL OPTION` command, when used before running the `READ DESIGN` command, blackboxes modules that do not use any input pins.

Example

In the following example, if you run `SET HDL OPTION -bbox_module_with_no_pi_used ON` before `READ DESIGN`, the module `top` will be blackboxed.

```
module top(a, b, out1);
input a, b;
output out1;
assign out1 = 1'b0;
endmodule
```

HRC1.7

Message

`A module name is used in more than one library`

Default Severity

Warning

Description

This message indicates that a module name has been duplicated in more than one library. Since the modules exist in different libraries, the tool keeps both modules. **Note:** To use the last module in different libraries, use the `-overwrite_mod` option of the `READ DESIGN` command.

Example

In the following example, the module name `bbox` is used in more than one library (in the Verilog default library and VHDL default library). Since they are in different libraries, the tool keeps them both. The instance `top.u1` will use the `bbox` from the Verilog default library.; the instance `sub.u0` will use the `bbox` in VHDL default library.

```
// bbox.v
module bbox (Z,A);
  output Z;
  input  A;
endmodule

--bbox.vhd
library ieee;
 use ieee.std_logic_1164.all;
 entity bbox is
 port ( Z    : out std_logic;
        A, B : in  std_logic);
end bbox;

architecture empty of bbox is
begin  -- empty
end empty;

--sub.vhd
library ieee;
use ieee.std_logic_1164.all;
```

```
entity sub is
  port ( Z    : out std_logic;
   A, B : in  std_logic);
end sub;

architecture rtl of sub is
  component bbox is
  port ( Z    : out std_logic;
         A, B : in  std_logic);
end component;

begin  -- rtl
  u0 : bbox port map (Z => Z, A => A, B => B);
end rtl;

// top.v
module top (Y,Z,A,B,C);
  output Y,Z;
  input  A,B,C;
  sub  u0 (.Z(Z),.A(A),.B(B));
  bbox u1 (.Z(Y),.A(C));
endmodule

// dofile
read design bbox.v top.v -noelaborate
read design bbox.vhd sub.vhd -vhdl -noelaborate
elaborate design -root top
```

HRC1.8

Message

```
The root module is a blackbox
```

Default Severity

Error

Description

Indicates that the specified root module is blackboxed by other rules.

Example

In the following example, the specified root module is blackboxed by HRC1.4. Conformal will report HRC1.8 since the specified root module is a blackbox.

```
// top.v
module top (in2, out3);
input in2;
output out3;
endmodule

// dofile
read design top.v -root top
```

HRC2.1

Message

`Module/entity instantiated from a library that has not been imported`

Default Severity

Warning

Description

The design includes one or more components that are instantiated, but Conformal did not find an imported library containing the components.

Example

In the following example, component `SUB` is instantiated but Conformal did not find an imported library (see line 8).

```
ARCHITECTURE arch OF test IS
BEGIN
  proc2 : PROCESS (in0, in1, sig2)
    BEGIN
      out0 <= in0 or in1 or sig2;
    END PROCESS;
  sig1 <= in2;
  inst1 : SUB PORT MAP (in0=>sig1, out0=>sig2);
END arch;
```

HRC2.2

Message

```
Module/entity exists in library and design
```

Default Severity

Warning

Description

One or more modules or entities exist in both design and library spaces. Conformal selects modules and entities from the design space.

To select modules and entities from the library space, add the following commands to the dofile before reading in the design:

```
add notranslate modules <module_name*> -design
```

and also add the "`-merge bbox`" option to the `READ DESIGN` command:

```
read design ... -merge bbox
```

Example

In the following example, Conformal reports HRC2.2 warnings because `lib.v` is read in for both the design and library spaces:

```
SETUP> read library lib.v
SETUP> read design  lib.v top.v
```

HRC2.3

Message

`Module/entity is renamed`

Default Severity

Warning

Description

The library space includes one or more modules or entities with the same name. Conformal will rename these library modules or entities.

Example

In the following example, the design includes the entity name `e1` in two libraries (`lib1` and `lib2`). Therefore, Conformal renames the `e1` entities as follows:

- From library `lib1`:

  ```
  ENTITY e1 IS
    PORT (min1: IN bit;
          mout1: OUT bit);
  END;
  ARCHITECTURE rtl OF e1 IS
  BEGIN
    mout1 <= not min1;
  END;
  ```

  Library 1: `e1` is renamed `lib1_e1`

- From library `lib2`:

  ```
  ENTITY e1 IS
    PORT (min1:  IN bit;
          mout1: OUT bit);
  END;
  ARCHITECTURE rtl OF e1 IS
  BEGIN
    mout1 <= not min1;
  END;
  ```

  Library 2: `e1` is renamed `lib2_e2`

HRC2.4

Message

`Ambiguous component is instantiated`

Default Severity

Warning

Description

A library component is instantiated, but more than one library contains the component.

Example

In the following example, component `e1` is instantiated, but both library `liba` and `libb` contain the `e1` component (see line 15).

```
LIBRARY liba;
LIBRARY libb;
USE liba.pkg.ALL;
USE libb.pkg.ALL;

ENTITY top IS
  PORT (in1: IN bit;
    out1: OUT bit);
END;
ARCHITECTURE rtl OF top IS
  COMPONENT e1
    PORT(min1: IN bit; mout1: OUT bit);
  END COMPONENT;
BEGIN
  inst1: e1 PORT MAP (min1=>in1, mout1=>out1);
END;
```

HRC2.5

Message

```
Same named library existed in both library and design space
```

Default Severity

Error

Description

Conformal does not allow the same named libraries exist in both library and design space. If the same named libraries exist in both library and design space, the rule violation will be issued.

Example

The following commands will trigger HRC2.5 rule violation since ’dut_lib’ exist in both library and design space:

```
    read library -mapfile dut_lib lib.v
    read design -mapfile dut_lib top.v
```

HRC2.6

Message

```
Multiple component is declared
```

Default Severity

Warning

Description

A component is declared in more than one library.

Example

In the following example, In line 13, component `c1` is declared in both `lib1`, package `pp1` and `lib2`, package `pp2`:

```
library lib1, lib2;
use lib1.pp1.all;
use lib2.pp2.all;
ENTITY top IS
  PORT (
    ccc : IN  bit;
    bbb : IN  bit;
    ooo : OUT bit
    );
END top;
ARCHITECTURE rtl OF top IS
BEGIN
  u0: c1 port map (ccc=>ccc, bbb=>bbb, ooo=>ooo);
END rtl;
```

HRC2.8

Message

```
Change name to an existing one
```

Default Severity

Warning

Description

Warns that you are changing the name of an object to an existing name of the same type in the design. The name will not be changed.

Example

In this Verilog example,

```
module mod(Z, A);
  input [1:0] A;
  output [1:0] Z;
  wire wire1, wire0;
  assign wire1 = A[1];
  assign wire0 = ~A[0];
  assign Z = {wire1, wire0};
endmodule
```

the following change name rule is to be applied:

```
mod net wire1 wire0
```

Because net '`wire1`' already exists in the design. LEC issues the following messages and does not change the name:

```
// Note: Read VERILOG design successfully
// Command: change name rename.rule -golden -summary
// Warning: The net name 'wire1' already exists in the design.
// Changing 'wire0' to this name will not take effect in the module 'mod'.
```

Similar message will also show up when user intends to change name on a port or a cell to an existing name of the same type.

HRC3.1

Message

```
Number of instance module port connections is different than the referenced module
     port declaration
```

Default Severity

Warning

Description

The design includes one or more module instances with arguments that differ from the module definition.

Example

In the following examples, instance `inst1` has 3 connections, but module `sub` has only 2 ports:

```
module sub (in2,out3);
input  in2;
output out3;
assign out3 = in2;
endmodule

module test (in2,out0);
input  in2;
output out0;
  sub inst1 (in2,in2,out0);
endmodule
```

HRC3.2

Message

```
Component is instantiated without any specified port connections
```

Default Severity

Warning

Description

Indicates that the design includes one or more components that are instantiated without port connections. When a component is instantiated, connections to the ports of the component must be specified.

Example

In the following example, module `SUB` is instantiated, but has no connected ports (see line 4).

```
module TEST (in2,out0);
input in2;
output out0;
  SUB inst1 ();
endmodule
```

HRC3.2a

Message

```
Module/entity has no I/O ports
```

Default Severity

Warning

Description

The module has no I/O ports.

Example

In the following example, module `ABC` does not have any I/O ports.

```
module ABC;
endmodule
```

HRC3.3

Message

`Undefined named port connection`

Default Severity

Error

Description

The design includes one or more module instances that refer to undefined ports.

Example

In the following example, module **test** references port **`in4`** in module **sub**, but module **sub** does not have port **in4**(see line 4).

```
module test(a, b, c, d, e);
 input a, b, c, d;
 output e;
 sub inst0 (.in1(a),.in2(b),.in3(c),.in4(d), .out1(e));
endmodule

module sub (in1, in2, in3, out1);
 input in1, in2, in3;
 output out1;
 wire out1;
 assign out1 = in1 ^ in2;
endmodule
```

```
Conformal will print out information about name of the instance where the problem
occured, the pinname, and the name of module:
```

```
// Error: HRC3.3: Undefined named port connection
//  Cannot find pin inst0/in4. No pin in4 defined in module sub.
//  (instance) on line 4 in file 'test.sv'
//  (module) on line 7 in file 'test.sv'
```

HRC3.4

Message

```
Duplicate port connection is detected
```

Default Severity

Error

Description

The design includes duplicate ports that Conformal has ignored.

Example

In the following example, Conformal will ignore the duplicate port `.c` included on line 4.

```
module test (a, b, q);
input [3:0] a, b;
output [3:0] q;
sub sub1 (.c(a), .c(b), .q1(q));
endmodule

module sub (c, d, q1);
input [3:0] c, d;
output [3:0] q1;
assign q1 = c | d;
endmodule
```

HRC3.4a

Message

`Actual net is connected to more than one port`

Default Severity

Warning

Description

Actual net is connected to more than one port in module declaration.

Example

In the following example, din is connected to both `.in1` port and `.in2` port (see line 1).

```
module sub(.in1(din), .in2(din), .out(dout));
input din;
output dout;
assign dout = din;
endmodule

module top(din, dout);
input din;
output dout;
sub sub1(.in1(din), .in2(din),.out(dout));
endmodule
```

HRC3.4b

Message

```
Grouped pins associated to blasted pins in an instantiation
```

Default Severity

Warning

Description

Indicates that during instantiation, grouped pins was used. But only individual pins (blasted pins) are declared in the sub module.

Example

In the example below, in module top, 'bus_wires' is a bus signal of range '[1:0]' connected to a grouped pin 'A' of sub module 'test'. However, in sub-module 'test', pin 'A' is blasted into 'A[0]' and 'A[1]'. In this case, Conformal will issue the HRC3.4b rule check message.

```
library(test)
{   voltage_map(VDD, 1.0);
    voltage_map(VSS, 0.0);
    cell(test_cell)
    {
        ....
        pin(A[0])
        {
            direction : input;
        }
        pin(A[1])
        {
            direction : input;
        }
        ....

}

In verilog:

module top ( input in, output out)
    wire [1:0] bus_wires;
    ...
    test_cell inst1( .A(bus_wires));
    ...
endmodule
```

HRC3.5a

Message

`Open input/inout port connection is detected`

Default Severity

Warning

Description

The design includes one or more input or inout ports with open connections.

Example

In the following example, the connection for port `.d` is open (see line 4).

```
module test (a, b, q);
input [3:0] a, b;
output [3:0] q;
sub sub1 (.c(a), .d(), .q1(q));
endmodule

module sub (c, d, q1);
input [3:0] c, d;
output [3:0] q1;
assign q1 = c | d;
endmodule
```

HRC3.5b

Message

`Open output port connection is detected`

Default Severity

Note

Description

The design includes one or more output ports with open connections.

Example

In the following example, the connection for port `.q1` is open (see line 4).

```
module test (a, b, q);
input [3:0] a, b;
output [3:0] q;
sub sub1 (.c(a), .d(b), .q1());
endmodule

module sub (c, d, q1);
input [3:0] c, d;
output [3:0] q1;
assign q1 = c | d;
endmodule
```

HRC3.5c

Message

`Expression to null port connection is detected`

Default Severity

Warning

Description

The design includes one or more null ports with open connections.

Example

In the following example, the connection between module port `c` and `d` is open (see line 4).

```
module test (a, b, q);
input [3:0] a, b;
output [3:0] q;
sub sub1 (a, x, b, q);
endmodule

module sub (c,, d, q1);
input [3:0] c, d;
output [3:0] q1;
assign q1 = c | d;
endmodule
```

HRC3.5d

Message

`Open port connection on liberty macro cell is detected`

Default Severity

Error

Description

The design includes one or more input or inout ports with open connections with liberty macro cell. It is used for detecting the open connection from the bottom up flow.

Example

In the following example, the connection of module `pkt_in[togl]` was not in liberty cell. The packed data type might have incorrect usage.

```
package my_pkg;
   // typedef definitons
    typedef struct packed
        {
        logic [4-1:0]              dest;       // packet destination address
        logic [4-1:0]              src;        // packet source address
        logic [8-1:0]              user;       // user application
        }
    xc0_pkt_hdr_tdef;
    typedef struct packed
        {
        xc0_pkt_hdr_tdef                        hdr;        // packet header
        logic [4-1:0]             data;       // packet body
        logic                                   actv;       // 1/0 pkt ready/idle
        }
    xc0_pkt_tdef;

    // transport protocol, tolerates retimes
    typedef struct packed
        {
        xc0_pkt_hdr_tdef                        hdr;        // packet header
        logic [4-1:0]             data;       // packet body
       logic                                  togl;       // toggles with new data
        }
    xc0_xpkt_tdef;

endpackage // my_pkg


module top (clk, pkt_in, pkt_out);
   import my_pkg::*;
   input logic clk;

   input  my_pkg::xc0_xpkt_tdef pkt_in;
   output  my_pkg::xc0_xpkt_tdef pkt_out;

   myblock i_myblock (.clk(clk), .pkt_in(pkt_in), .pkt_out(pkt_out));

endmodule


 cell(myblock){
       ...
       is_macro_cell: true;
       ...

       bus (pkt_in[hdr][dest]) {
       ...
       }
       ...
       bus (pkt_in[hdr][src]) {
       ...
       }
       ...
       bus (pkt_in[hdr][user]) {
       ...
       }
       ...
      bus (pkt_in[data]) {
       ...
       }
       ...
      bus (pkt_in[hdr][user]) {
       ...
       }
       ...
       pin (pkt_in[actv]) {
       ...
       }
}
```

HRC3.6

Message

```
Port connection width mis-matches. Undriven signals are floating
```

Default Severity

Warning

Description

The design includes at least one port that has port connection width mis-matches.

Example

In the following example, port_name `out1`, which is two-bit, is greater in size than `port_expr e`, which is one-bit. See line 4.

```
module test(a, b, e);
 input a, b;
 output e;
 sub inst0 (.in1(a),.in2(b),.out1(e));
endmodule

module sub (in1, in2, out1);
 input in1, in2 ;
 output [1:0] out1;
 wire [1:0] out1;
 assign out1 = {in1,in2};
endmodule
```

HRC3.6a

Message

Port connected to a constant of smaller bit-width

Default Severity

Warning

Description

Input port of a module is connected to a constant of smaller bit-width.

Example

In the following example, the expression 'A+B" is of size 4-bit, while the port 'i' of module 'sub' is of size 8-bit.

```
module test(output byte o);
    parameter [3:0] A = 4'b1000;
    parameter [3:0] B = 4'b1000;
    sub sub (.o(o), .i(A+B));

 endmodule

module sub(output byte o, input byte i);
    assign o = i;
 endmodule
```

Note: By default, the unmatched port bits will be left unconnected. The command 'set hdl option -PORTMISmatch <UNconnect | EXTend>' can be used to change this behavior.

HRC3.6b

Message

Input port connection width mis-matches. Extra bits ignored

Default Severity

Warning

Description

Input port of a module is connected to an expression in the parent module of a larger bit-width.

Example

In the following example on line 12, the variable 'accu' of size 6-bit is connected to input port 'in' of module 'mid' of size 5-bit. The extra bit ('accu[5]') is not driving anything in sub-module 'mid' and is ignored. Conformal will report HRC3.6b rule violation.

```
1 module mid (input  logic[4:0] in, output logic out);
2   assign out = & in;
3 endmodule
4
5 module test (input clk, input rst, output out);
6   reg [5:0] accu;
7   always_ff @(posedge clk or negedge rst)
8     if (~rst)
9       accu <= 6'b010101;
10     else
11       accu <= accu + 6'b1;
12   mid mid_u (.in(accu), .out(out));
13 endmodule
```

HRC3.7

Message

```
Boundary port direction might not be correct
```

Default Severity

Warning

Description

The design includes one or more module boundary ports that might be declared incorrectly because of their irregular use.

Example

In the following example, both input `in1` and input `in2`, drive output `out0`. Thus, Conformal interprets this situation to mean that input `in2` is incorrectly declared (see lines 4 and 5).

```
module test(in1, in2, out0);
input in1, in2;
output out0;
not U1 (out0, in1);
assign out0=in2;
endmodule
```

Note: For multi-driven nets, Conformal applies wire-AND resolution by default.

HRC3.8

Message

`Port positional association occurs in an instantiation`

Default Severity

Warning

Description

The design includes one or more module instances that use port positional association. In a positional association, the port_exprs connect to the ports of the module in the specified order.

Port positional association syntax:

```
module_name instance_name( port_expr1,
    port_expr2, …);
```

Example

In the following example, port_expr `a` connects to port `o0` and port_expr `b` connects to port `o1` of module `sub`. See line 9.

```
module sub (o0, o1);
 output o0, o1;
 assign o0 = 1'b0;
 assign o1 = 1'b1;
endmodule

module test(a, b);
 output a, b;
 sub s1(a, b);
endmodule
```

HRC3.8a

Message

```
Port positional association of Liberty cell occurs in an instantiation
```

Default Severity

Error

Description

Indicates that the design includes one or more module instances that use port positional association of Liberty cells. When a Liberty cell contains `pg_pin`, you cannot use positional association to connect ports to the cell.

To trigger this rule check, use the
`SET LOWPOWER OPTION -GOLDen_power_domain physical` and
`SET HDL OPTION -ERROR_OUT_POSITIONAL_ASSOCIATION_ON_LIBERTY_CELL ON`
commands.

Example

The example has a Liberty cell named `isolation_cell` that contains `pg_pin` in `test.lib`. The module `top` instantiates the `isolation_cell` using port positional association in top.v.

```
/* test.lib */
library(test)
{
  voltage_map(VDD, 1.0);
  voltage_map(VSS, 0.0);

  cell(isolation_cell)
   {
    pg_pin(VDD){
    voltage_name: VDD;
    pg_type: primary_power;
    }
    pg_pin(VSS){
    voltage_name: VSS;
    pg_type: primary_ground;
    }
    pin(A)
    {
     direction : input;
    }
    pin(Y)
    {
     direction : output;
    }
   }
}

// top.v
module top(A, B, C, D);
inout A, B, C, D;

isolation_cell ins1(A, B, C, D);

endmodule
```

To trigger ths HRC3.8a rule check, use the following commands:

```
// dofile
set lowpower option -GOLDen_power_domain physical
set hdl option -ERROR_OUT_POSITIONAL_ASSOCIATION_ON_LIBERTY_CELL ON
read library -lib test.lib
read design top.v
```

HRC3.9

Message

```
Constraint contains floating net
```

Default Severity

Warning

Description

The constraint has floating nets.

Example

In the following example, `val` is a floating net (see line 8):

```
module test(in, out);
input in;
output out;
wire val;
assign out = in;
endmodule
append_to module test;
$constraint( val == 1'b1);
endmodule
```

HRC3.10

Message

```
Input port connects to instance output pin. This might cause multiple drivers to
     the net
```

Default Severity

Warning

Description

There is an input port connected to an instance output pin. This could cause multiple drivers to the net.

Example

In the following example, the input pin `b` is connected to the instance `sub` output pin `o1`. In module `ABC` it is an input pin, and in `sub s1` is is an output pin.

```
module sub (o0, o1);
output o0, o1;
assign o0 = 1'b0;
assign o1 = 1'b1;
endmodule
module ABC(a, b);
  output a;
  input b;
  sub s1(a, b);
endmodule
```

HRC3.10a

Message

`An input port is declared, but it is not completely used in the module`

Default Severity

Warning

Description

An input port is declared, but all are part of its bits are not used in the module.

Example

In the following example, the input pin `in1` is declared, but not used in the module.

```
module sub (in1, in2, out1);
 input in1, in2;
 output out1;
 assign out1 = in2;
endmodule
```

HRC3.10b

Message

```
An input port is declared, but it is not used. Module is empty
```

Default Severity

Warning

Description

An input port is declared, but none of its bits are not used in the module, and the reason is the module is empty.

Note: By default, up to 5 occurrences per-module of HRC3.10b violations will be reported. The command 'set_rule_handling' command '-MODULE_LIMIT <n>' option to change the maximum number of occurrences reported.

Example

In the following example, .

```
module sub (input in1, input in2, output out1);
`ifdef CONNECT
assign out1 = in2;
`endif
endmodule

module test (input in1, input in2, output out1);
sub sub(.*);
endmodule
```

HRC3.11

Message

```
Too many actuals connect to formals
```

Default Severity

Error

Description

The number of actuals in the port map is more than the number of formals.

Example

In the following example, there are three actuals but only two formals (see line 20):

```
entity sub is
  port (
    o0, o1  : out bit
  );
end sub;
architecture arch of sub is
begin
  o0 <= '0';
  o1 <= '1';
end arch;
entity test is
  port (
    a   : out bit;
    b   : out bit;
    c   : in bit
  );
end test;
architecture arch of test is
begin
  u1 : sub port map (o0 => a, o1=> b, o1 =>c);
end arch;
```

HRC3.12

Message

`Port declaration error`

Default Severity

Warning

Description

The explicit port name specified by `.name` should not conflict with other implicit or explicit port names.

Example

In the example below, there are two port definitions with the same name 'a1':

```
module test(a1, .a1(a2), out);
  input a1, a2;
  output out;
   assign out = |{a1, a2};
endmodule
```

HRC3.13

Message

```
Usage of 'const ref' port is treated as 'input' port
```

Default Severity

Warning

Description

The 'const ref' port is treated as 'input' port. You will get this message when running the `READ DESIGN -systemverilog` command to read in the design and there is a violation on this rule.

Example

In the following example, in line 5, port 'din' of function func is declared as 'const ref', it will be treated as 'input' port.

```
module test(in, out);
input in;
output reg out;
function func;
const ref din;
begin
  func = din;
end
endfunction
always @(in)
  out = func(in);
endmodule
```

HRC3.14a

Message

```
Constraint not applied (object is not a DFF/DLATCH)
```

Default Severity

Warning

Description

The input instance is not a sequential object (DFF/DLATCH). Constraints specified with the `ADD INSTANCE CONSTRAINTS` command were not applied.

Example

For the following command, instance `U6/U1` is not a sequential object.

```
add instance constraints 1 /U6/U1
```

HRC3.14b

Message

```
Constraint not applied (already applied)
```

Default Severity

Warning

Description

The constraint has been set to the input instance object. Constraints specified with the `ADD INSTANCE CONSTRAINTS` command were not applied.

Example

In the following example, the same instance path is used in the second `ADD INSTANCE CONSTRAINTS` command

```
add instance constraints 1 /U6/U5 /U7/U8
add instance constraints 0 /U6/U5
```

HRC3.14c

Message

```
Constraint not applied (not found)
```

Default Severity

Warning

Description

The input instance object is not found. Constraints specified with the `ADD INSTANCE CONSTRAINTS` command were not applied.

Example

For the following command, instance `U6/U1` does not exist.

```
add instance constraints 0 /U6/U1
```

HRC3.15

Message

```
Parameter positional association occurs in an instantiation
```

Default Severity

Warning

Description

The design includes one or more module instances that uses parameter positional association.

In a positional association, the `param_exprs` are associated with the parameters of the module in the specified order in which their parameter association should occur. See Example. Port positional association syntax:

```
module_name #(param_expr1, param_expr2,...) instance_name ( port_expr1,
    port_expr2, ...);
```

Note: Positional association can cause incorrect associations between the formal parameters and the corresponding actual parameter. To avoid this, use named association instead.

Example

In the following example, parameters `p1` and `p2` of module `sub` are overridden by `1'b1` and `1'b0` which are the values defined in the instance `s1` of module `sub`. See line 11.

```
module sub (o0, o1);
 output o0, o1;
 parameter p1 = 1'b0;
 parameter p2 = 1'b1;
 assign o0 = p1;
 assign o1 = p2;
endmodule

...
module test(a, b);
 output a, b;
 sub #(1'b1, 1'b0) s1 (.o0(a), .o1(b)); // line 11
endmodule
```

HRC3.16

Message

`A wire is declared, but not used in the module`

Default Severity

Warning

Description

This is triggered when a wire is declared, but not used in the module.

Example

In the following example, the wire `w` is declared, but not used in the module.

```
module test(a, b, o);
input a, b;
output o;
wire w;
assign w = a & b;
endmodule
```

HRC3.17

Message

```
Inconsistent direction of the range of the port in a component declaration is
     detected
```

Default Severity

Warning

Description

The port direction of VHDL component declaration is inconsistent with the port direction of Liberty cell.

Example

In the following example, the port direction of VHDL component declaration is descending in `top.vhd` while the port direction of Liberty cell `mybuf` is ascending in `test.lib`.

```
/* test.lib */
library(test){
   type("BUS2_type1"){
       base_type : "array";
        data_type : "bit";
        bit_width : 2;
        bit_from : 0;
        bit_to : 1;
   }
   cell("mybuf"){
     bus(A) {
        bus_type : "BUS2_type1";
        direction : "input";
        pin("A[0]"){
           direction : "input";
     }
       pin("A[1]"){
          direction : "input";
        }
   }
       bus(Y) {
         bus_type : "BUS2_type1";
       direction : "output";
       pin("Y[0]"){
          direction : "output";
       }
         pin("Y[1]"){
          direction : "output";
        }
     }
    }
}

-- top.vhd
library ieee;
use ieee.std_logic_1164.all;

package my_pkg is
component mybuf
 port(a: in std_logic_vector(2 downto 0);
   y: out std_logic_vector(2 downto 0));
end component;

end my_pkg;

library ieee;
use ieee.std_logic_1164.all;
use work.my_pkg.all;

entity top is
  port(a: in std_logic_vector(2 downto 0);
  o: out std_logic_vector(2 downto 0));
end top;

architecture rtl of top is
begin
 U1: mybuf port map(a=>a,y=>o);
end rtl;
```

HRC4

Message

```
Multiple root modules/entities found
```

Default Severity

Warning

Description

The design includes multiple modules or entities at the top level of hierarchy; thus, any one of them can be designated as the root module. By default, Conformal selects the module with the greatest gate count.

Example

In the following example, the file `test.v` contains two possible root modules, `TEST0` and `TEST1`.

```
test.v file:
module TEST0 (in0,in1,in2,out0);
input in0,in1,in2;
output out0;
assign      out0 = in0 | in1;
endmodule

module TEST1 (in0,in1,in2,out0);
input in0,in1,in2;
output out0;
assign      out0 = in0 & in1;
endmodule
```

HRC5

Message

`DesignWare referenced but not defined`

Default Severity

Warning

Description

The design includes one or more DesignWare multiplier/divider instances, but does not define them. Conformal supplies the missing definition, using a Conformal multiplier definition.

Example

In the following example, DesignWare multiplier `DW02_mult` is instantiated (see line 9), but the design does not define it. Conformal will supply the missing `DW02_mult` definition with a Conformal multiplier definition.

```
module test(A,B,TC,PRODUCT);
  parameter     A_width = 8;
  parameter     B_width = 8;
input   [A_width-1:0]   A;
input   [B_width-1:0]   B;
input               TC;
output   [A_width+B_width-1:0]   PRODUCT;

DW02_mult mult_1 (A,B,TC,PRODUCT);
endmodule
```

HRC6.1

Message

```
Mapping empty edif cell to parameterized module is not supported
```

Default Severity

Warning

Description

The EDIF design includes an empty cell that is an instance of a parameterizable Verilog module, which is not supported.

Example

In the following example, `top.edf` is an EDIF file that contains a cell called `CTLBLK`, which is defined in a Verilog module.

`CTLBLK` module:

```
module CTLBLK( A, SUM );
parameter INIT = 16'h0000;
…
endmodule
```

Input:

```
> read design top.edf -golden -edif
```

```
// Warning: (HRC6.1) Mapping empty edif cell to parameterized module is not
supported (occurrence:1)
```

HRC7

Message

```
Module specified by the 'add notranslate modules' command cannot be found
```

Default Severity

Warning

Description

Warns that a module that was specified as a target of the ADD NOTRANSLATE MODULES command cannot be found. The ADD NOTRANSLATE MODULES command specifies library or design modules that are parsed, but will not be translated when running the READ LIBRARY or READ DESIGN command. These modules automatically become blackboxes.

Example

In the following example `mod1` is the name of module that does not exist in the design.

Input:

```
> add notranslate modules mod1
// Warning: (HRC7) Modules specified by the 'add notranslate modules' command
cannot be found (occurrence:2)
```

HRC8

Message

```
Partial sum outputs 'DW'/'CW' module is instantiated
```

Default Severity

Note

Description

Issued when any of the following modules have been instantiated: DW02_tree/DW02_multp/DW_squarep/CW_tree/CW_multp/CW_squarep.

Example

LEC would report HRC8 because in the following example module 'test' has instantiated 'CW_tree'.

```
module test( inst_INPUT, inst_OUTPUT);
parameter num_inputs  = 2;
parameter input_width = 2;

input  [num_inputs*input_width-1 : 0] inst_INPUT;
output [input_width : 0] inst_OUTPUT;

wire   [input_width-1 : 0] OUT0, OUT1;

CW_tree #(num_inputs, input_width)
U1 ( .INPUT(inst_INPUT), .OUT0(OUT0), .OUT1(OUT1));

assign inst_OUTPUT = OUT0 + OUT1;

endmodule
```

HRC9

Message

```
Generated name conflicts with existing object name
```

Default Severity

Warning

Description

Conformal will report HRC9 warning if a complex type object is expanded to conflict with existing object name.

This can be fixed by changing "set naming rule" command to avoid conflicting names.

Example

Conformal reports HRC9 with the following dofile commands and the VHDL example. Similar issue can happen in a System Verilog design if a struct type object is expanded to conflict with existing object name.

```
// dofile commands
set naming rule "_" "_" -array_delimiter -golden
set naming rule "_" "" -field_delimiter -golden
read des -vhdl t1.vhd
```

VHDL example:

```
package mypackage is
type myrecordtype is record
myindex : bit_vector(15 downto 0);
end record;
end package;
use work.mypackage.all;
entity top is
  port (out1, out2   : out bit_vector(15 downto 0);
       clk                : in  bit;
       collision1         : in  myrecordtype;
       collision2_myindex : in  bit_vector(15 downto 0));
  end top;

architecture rtl of top is
  signal collision1_myindex : bit_vector(15 downto 0);
  signal collision2         : myrecordtype;
begin
```

```
flops : process(clk)
begin
  if clk'event and clk='1' then
   collision1_myindex <= collision1.myindex;
   collision2.myindex <= collision2_myindex;
  end if;
end process;
out1 <= collision1_myindex;
out2 <= collision2.myindex;
end;
```

HRC10.1

Message

```
Array variable is abstracted as memory model
```

Default Severity

Note

Description

This message indicates that a memory model was abstracted for 2D array variable.

Example

In the following example, the number of address bits required for 'mem_var[3:0]' is '2'. But the number of address bits specified by 'raddr' is 3-bit and by 'waddr' is 4-bit.

```
module top(din, dout, raddr, waddr, clk, we);

    input [7:0] din;
    output [7:0] dout;
    input clk, we;

    input [2:0] raddr; // read address
    input [2:0] waddr; // write address

    reg [7:0] mem_var [7:0]; // memory model extracted for this variable

    always @(posedge clk)
       if ( we )
         mem_var[waddr] &lt;= din;

    assign dout = mem_var[raddr];

endmodule
```

HRC10.2

Message

`Array variable is not abstracted as memory model`

Default Severity

Warning

Description

This message indicates that an abstracted memory model was not implemented as a memory element. Instead the abstracted memory is defaults to implementation using regular logic. This condition can happen when certain requirements for implementing as an abstracted memory element are not met such as address width/data width is too small, memory access is not via indexed expression, memory element is not a DFF/DLATCH among others.

Example

In the following example, the memory address width is 1-bit and hence it's not implemented as memory model (there's a restriction that addr or data width should be greater than 1 currently)

```
module top( clk, rst_n, ina_1, in_1, in_2, ina_2, out_1);

    input clk, rst_n;
    input [2:0] ina_1;
    input in_1;
    input in_2;
    input [4:0] ina_2   ;
    output [5:0] out_1;
   reg [5:0] out_1;

    reg [4:0] mem_var  [0:1];

    always @ (posedge clk or negedge rst_n) begin

       if (!rst_n) begin
          mem_var[0] &lt;= '0;
          mem_var[1] &lt;= '0;
       end
       else begin
          out_1 &lt;= mem_var[in_1];
          if (ina_2 &gt; mem_var[~in_2]) begin
             mem_var[in_2] &lt;= '0;
```

```
            end
            else if (ina_2 &gt;= mem_var[in_2]) begin
               mem_var[in_2] &lt;= '1;
            end
         end
      end

endmodule
```

HRC10.4

Message

```
Logic gate is unsupported
```

Default Severity

Error

Description

Indicates that the logic gate abstraction fails in liberty cell extraction.

Example

```
// tran.v
module inout_module(inout1, inout2);
  inout inout1, inout2;
  tran inst1(inout1, inout2); // HRC10.4
endmodule
```

```
// dofile
read_hdl tran.v
elaborate -top inout_module
```

HRC10.5

Message

`Logic gate is abstracted as empty black box model`

Default Severity

Note

Description

This message indicates that a memory model was abstracted for 2D array variable.

Example

Indicates that the logic gate abstraction would create empty black box for the extracted liberty cell.

```
// nmos_nand.v
module nmos_nand_2( Y, A, B);
  output Y;
  input A, B;
  supply0 GND;
  wire w1;

  pullup (Y);  // HRC10.5
  nmos (Y, w1, A);
  nmos (w1, GND, B);
endmodule
```

```
// dofile
read_hdl nmos_nand.v
elaborate -top nmos_nand_2
```

HRC10.6

Message

```
Logic gate is abstracted as timing black box model
```

Default Severity

Note

Description

Indicates that the logic gate abstraction would create empty black box for the extracted liberty cell.

Example

```
// vdw_addr.v
module addr(in1, in2, out);
  input [31:0] in1, in2;
  output [31:0] out;

  assign out = in1 + in2;

endmodule
```

```
// dofile
read_hdl vdw_addr.v
elaborate -top addr
```

Ignored

This category of rules applies to designs that include constructs or statements that are either redundant or unsupported, and thus ignored by Conformal. The following lists the Ignored (IGN) rule checks. The severity level for all IGN rule check messages is *warning*.

- IGN5.2 on page 171
- IGN5.3 on page 172
- IGN5.4 on page 173
- IGN5.5 on page 174
- IGN5.6 on page 175
- IGN5.7 on page 177
- IGN6.1 on page 178
- IGN6.2 on page 179
- IGN7.1 on page 180
- IGN8.1 on page 181

IGN1.1

Message

```
Initial construct is not supported. The entire initial construct will be ignored.
```

Default Severity

Warning

Description

The `initial` statement is a SystemVerilog construct for simulation that Conformal does not support. Thus, when Conformal detects this construct in the syntax, it ignores the entire `initial` construct. Similarly, Conformal reports rule IGN1.2 for the `final` statement.

Example

In the following example, Conformal ignores the keyword `initial`. See line 5.

```
module test ( clk, din, dout );
input  clk, din;
output dout;
wire dout;
initial dout = !din;
assign dout = din;
endmodule
```

IGN1.2

Message

```
Final construct is not supported. The entire final construct will be ignored
```

Default Severity

Warning

Description

The `final` statement is a SystemVerilog construct for simulation that Conformal does not support. Thus, when Conformal detects this statement in the syntax, it ignores the entire `final` statement. Similarly, Conformal reports rule IGN1.1 for the `initial` statement.

Example

In the following example, Conformal ignores the entire `final` statement. See lines 7, 8, and 9.

```
module test(aa, bb, o1, o2);
input aa, bb;
output o1, o2;
  assign o1 = aa == bb;
  assign o2 = aa != bb;

  final begin
    $display("done");
  end
endmodule
```

IGN2.1

Message

```
Delay value(s) are ignored
```

Default Severity

Warning

Description

Conformal has ignored one or more delay values.

Example

In the following example, Conformal ignores delay value `#(10)`. See line 5.

```
module test ( clk, din, dout );
input  clk, din;
output dout;
wire dout;
not #(10) (dout, din);
endmodule
```

IGN2.2

Message

`Invalid defparam statement(s) are ignored`

Default Severity

Warning

Description

The design uses one or more `defparam` keywords incorrectly. Conformal ignores incorrectly used `defparam` keywords.

Example

In the following example, defparam assignment to interface port ’m_if.NN’ is not supported. Conformal will issue IGN2.2 and ignore the statement. See line 17, 18, 19.

```
1 interface ifc1;
2    parameter NN = 1'b0;
3    wire aa, oo;
4    modport mp(input aa, output oo);
5    assign oo = aa & NN;
6 endinterface
7
8 module sub2(ifc1.mp m_if);
9    assign m_if.oo = m_if.aa & m_if.NN;
10 endmodule
11
12 module test(input aa, output oo);
13    ifc1 u_ifc();
14    assign u_ifc.aa = aa;
15    assign oo = u_ifc.oo;
16    sub2 sub [2:0] (u_ifc);
17    defparam sub[0].m_if.NN = 4'd1;
18    defparam sub[1].m_if.NN = 4'd1;
19    defparam sub[2].m_if.NN = 4'd1;
20 endmodule
```

IGN2.3

Message

```
Unsupported defparam statement is ignored
```

Default Severity

Warning

Description

This particular usage of `defparam` is not supported; the `defparam` statement will be ignored.

Example

In the following example, the defparam statement 'sub.m_if.NN' in module test is ignored:

```
interface ifc1;
  parameter NN = 1'b0;
  wire aa, oo;
  modport mp(input aa, output oo);
  assign oo = aa & NN;
endinterface

module sub2(ifc1.mp m_if);
  assign m_if.oo = m_if.aa & m_if.NN;
endmodule

module test(input aa, output oo);
  ifc1 u_ifc();
  assign u_ifc.aa = aa;
  assign oo = u_ifc.oo;
  sub2 sub(u_ifc);
  defparam sub.m_if.NN = 1'b1;
endmodule
```

IGN3.1

Message

```
Duplicate pin/port names are detected and renamed
```

Default Severity

Warning

Description

The design includes duplicate pins or ports. . The subsequent pins or ports with the sane name are renamed. Internally, the renamed pins or ports are connected to the original pins or ports.

Example

In the following example, Conformal renames the duplicated port `dout` (see the first line)

```
module test ( dout, clk, din, dout );
input  clk, din;
output dout;
reg dout;
  always @(posedge clk)
dout <= din;
endmodule
```

IGN3.1a

Message

```
Duplicate pin/port names are detected
```

Default Severity

Warning

Description

The design includes duplicate pins or ports. For each of the duplicated pin of port, Conformal keeps the name of the pin or port the same.

Note that by default Conformal will rename the duplicated pins or ports. To keep the names the same, pass the -norename option to the read design command.

Example

In the following example, the module 'test' has two output ports named 'dout', and two input ports named 'din.'

```
module test ( dout, clk, din, dout, din );
input  clk, din;
output dout;
reg dout;
  always @(posedge clk)
dout <= din;
endmodule
```

IGN3.2

Message

```
Duplicate modules/entities are detected. Subsequent modules/entities are ignored
```

Default Severity

Warning

Description

The design includes duplicate modules or entities. Conformal ignores duplications.

Example

In the following example, the design duplicates `module test`. Conformal ignores the second `module test`.

```
module test ( clk, din, dout );
 input clk, din;
 output dout;
endmodule

module test ( clk, din, dout );
 input clk, din;
 output dout;
endmodule
```

IGN3.2a

Message

```
Duplicate modules are defined in both Liberty and Verilog. The function in Liberty
     will potentially get changed
```

Default Severity

Note

Description

The design includes duplicate modules or entities. The function in Liberty can potentially be changed because of the usage of "copy_verilog_to_liberty" in the HDL setup. For details please refer the the man page of `SET HDL OPTION -copy_verilog_to_liberty`.

IGN3.2b

Message

```
Construct name conflicts with earlier modules/entities name declaration and
     is ignored
```

Default Severity

Warning

Description

This rule is reported when the name of a construct at top level source description scope conflict with earlier definition of a module (or an entity in the case of VHDL).

Example

In the following example, 'ptest' is defined as an interface in file 'ptest.sv' but is previously defined as a 'module' in file 'test.sv'. The new definition of 'ptest' as an interface will be ignored.

File test.sv:

```
module ptest(input clk, rst, d, output reg q);
    always @(posedge clk or negedge rst)
     if (!rst)
         q <= 1'b0;
     else
         q <= d;
 endmodule

module test(input clk, rst, d, output q);
     ptest ptest(.*);
 endmodule
```

File ptest.sv:

```
interface ptest(input clk, rst);
    wire d;
    reg  q;
    modport mp(input d, output q);
endinterface
```

IGN3.2c

Message

```
Construct name conflicts with earlier package name declaration and is
     ignored
```

Default Severity

Warning

Description

This rule is reported when a new package has the same name with another earlier defined package.

Example

In the following example, 'ptest' is defined as a package in file 'ptest.sv' and also in file 'test.sv'. The new re-definition of 'ptest' in file 'ptest.sv' will be ignored.

File test.sv:

```
package ptest;
  wire parameter FOO = 10;
endpackage

module ptest(input clk, rst, d, output reg q);
  always @(posedge clk or negedge rst)
      if (!rst)
          q <= 1'b0;
      else
          q <= d;
enmoduled

module test(input clk, rst, d, output q);
      ptest ptest(.*);
endmodule
```

File ptest.sv:

```
package ptest;
  wire parameter FOO = 10;
endpackage
```

IGN3.2d

Message

```
Construct name conflicts with earlier interface name declaration and is
     ignored
```

Default Severity

Warning

Description

This rule is reported when the name of a construct at top level source description scope conflict with earlier definition of an interface

Example

In the following example, 'ptest' is defined as an module in file 'test.sv' but is previously defined in as an 'interface' in file 'ptest.sv'. The new definition of 'ptest' as a module will be ignored.

File ptest.sv:

```
interface ptest(input clk, rst);
    wire d;
    reg  q;
    modport mp(input d, output q);
endinterface
```

File test.sv:

```
module ptest(input clk, rst, d, output reg q);
    always @(posedge clk or negedge rst)
     if (!rst)
         q <= 1'b0;
     else
         q <= d;
 endmodule

module test(input clk, rst, d, output q);
     assign q = d;
 endmodule
```

IGN3.2e

Message

`Construct name conflicts with earlier class name declaration and is ignored`

Default Severity

Warning

Description

This rule is reported when the name of a construct at top level source description scope conflict with earlier definition of a class

Example

In the following example, 'ptest' is defined as a class in file 'ptest.sv' but is redefined in as a 'module' in file 'test.sv'. The new definition of 'ptest' as a module will be ignored.

File ptest.sv:

```
class ptest;
    bit d, q;
    bit clk;
    bit rst;
endclass
```

File test.sv:

```
module ptest(input clk, rst, d, output reg q);
   always @(posedge clk or negedge rst)
    if (!rst)
        q <= 1'b0;
    else
        q <= d;
endmodule

module test(input clk, rst, d, output q);
    assign q = d;
endmodule
```

IGN3.2f

Message

```
Construct name conflicts with earlier property name declaration and is
     ignored
```

Default Severity

Warning

Description

This rule is reported when the name of a construct at top level source description scope conflict with earlier definition of a property

Example

In the following example, 'ptest' is defined as a property in file 'ptest.sv' but is redefined as a 'module' in file 'test.sv'. The new definition of 'ptest' as a module will be ignored.

File ptest.sv:

```
property ptest(int d, int q, bit clk, bit rst);
    int d_sav;
    @(posedge clk)
        (!rst, d_sav = d) |=> ##1 (d_sav == q);
endclass
```

File test.sv:

```
module ptest(input clk, rst, d, output reg q);
   always @(posedge clk or negedge rst)
    if (!rst)
        q <= 1'b0;
    else
        q <= d;
endmodule

module test(input clk, rst, d, output q);
    assign q = d;
endmodule
```

IGN3.3

Message

```
Multiple declarations of same package are detected. Earlier declarations are
     ignored
```

Default Severity

Warning

Description

There are multiple declarations of the same package. Conformal ignores earlier declarations.

Example

In the following example, the first declaration for `pkg` is ignored.

```
package pkg is
  CONSTANT val1 : bit := '1';
end pkg;

package pkg is
  CONSTANT val1 : bit := '0';
end pkg;
```

IGN3.4

Message

`Duplicate modules/entities are detected. Previous modules/entities are ignored`

Default Severity

Warning

Description

There are multiple definitions for a module or entity. Conformal takes the last definition and ignores previous ones. This usually occurs when you use the `-lastmod` option with the `READ DESIGN` command.

Example

In the following example, Conformal ignores the first definition for `test` when you use the `read design -lastmod test.v -replace` command.

```
module test ( clk, din, dout );
input clk, din;
output dout;
reg dout;

always @(posedge clk)
  dout <= din;

endmodule

module test (clk, din, dout );
input clk, din;
output dout;
reg dout;

always @(posedge clk)
dout <= din;

endmodule
```

IGN3.5

Message

```
Duplicate modules/entities are detected. Local module/entity is used
```

Default Severity

Warning

Description

Duplicate modules are found and the Conformal software uses the local module. If you want to use the local module, you must use the `read design` -`localref` command.

Example

In the following example, you can use `read design -localref` to use local module `sub1` (see lines 9 for each of the following files). For each file, the instantiation of `sub1` in lines 9 will use the `sub1` defined in lines 1 (`module sub1(aa, oo)`).

File `mod1.v`

```
module sub1(aa, oo);
input aa;
output oo;
  assign oo = aa;
endmodule
module mod1(aa, oo);
input aa;
output oo;
  sub1 u0 (aa, oo);
endmodule
```

File `mod2.v`

```
module sub1(aa, oo);
input aa;
output oo;
  assign oo = !aa;
endmodule
module mod2(aa, oo);
input aa;
output oo;
  sub1 u0 (aa, oo);
endmodule
```

IGN3.6

Message

`Blackbox is replaced with the current module/entity`

Default Severity

Warning

Description

The blackbox module is replaced by a non-blackbox module.

Example

In the following example, `read library mod1.v mod2.v` selects blackbox module `sub1()` from `mod1.v`, and `read library -lastmod mod1.v mod2.v` selects blackbox module `sub2()` from `mod2.v`.

To avoid selecting blackboxes, use the `read library -bboxsolver mod1.v mod2.v` command, which selects `sub1()` from `mod2.v`, and `sub2()` from `mod1.v`.

```
File mod1.v:
module sub1(aa, oo);
 input aa;
 output oo;
// this is a bbox
endmodule

module sub2(aa, oo);
 input aa;
 output oo;
 assign oo = aa;
endmodule

File mod2.v:
module sub1(aa, oo);
 input aa;
 output oo;
 assign oo = !aa;
endmodule

module sub2(aa, oo);
 input aa;
 output oo;
 // this is a bbox
endmodule
```

IGN3.6a

Message

```
Previously blackboxed cell gets unblackboxed because the module functions are
     filled by Verilog model
```

Default Severity

Note

Description

Indicates that a previously blackboxed cell becomes unblackboxed during liberty parsing because no functions are defined, but the function is later submitted by the Verilog model.

Application

Library parsing flow when -copy_verilog_to_liberty is enabled during HDL setup.

IGN3.6b

Message

```
Liberty cell function is overwritten by function specified from Verilog model
```

Default Severity

Note

Description

Indicates a cell's function that was previously specified in Liberty gets overwritten by a function in the Verilog model.

IGN3.7

Message

```
Modules extracted as macro models in power intent library database and all of their
     submodules will not be unblackboxed by the option
     "use_verilog_function_always"
```

Default Severity

Warning

Description

The option `use_verilog_function_always` will not affect the modules that extracted as macro models in the power intent library database and all of their submodules. A module will be extracted as a macro in the power intent library database if it meets one of the following:

- Has the Liberty attributes: `is_macro_cell: true` or `pad_cell: true`
- Does not have any lowpower cell attribute defined and have multiple pins or multiple ground pins (flagged by `LIB_LINT_067`)
- Does not have any lowpower cell attribute defined and at least one boundary data pin has `is_isolated:true` attribute (flagged by `LIB_LINT_067`)

In order for the cells being parsed in the power intent library database, you must have `-lp` specified when reading them in the liberty libraries. if you want the tool to only extract the cells as macro models from the explicitly specified Liberty attribute, please use `set lowpower option -only_extract_macro_with_attribute` in the dofile.

Please check the Liberty library for any missing low power attribute and if it is expected to be extracted as a macro model.

Application

Library parsing flow when `-copy_verilog_to_liberty` is enabled during HDL setup.

IGN3.8

Message

```
Parameter/generic module is not merged into liberty module
```

Default Severity

Warning

Description

Indicates the verilog module is defined with parameter(s) and is not merged into the corresponding liberty cell during the `read library` command.

IGN3.9a

Message

```
Verilog module is not merged into the Liberty cell because USE_LIBERTY_FUNCTION is
     applied
```

Default Severity

Warning

Description

This is only triggered under flow "`-COPY_VERILOG_TO_LIBERTY`" with argument "`USE_LIBERTY_FUNCTION`". When the design instantiates liberty cell, the rule will be reported.

IGN3.9b

Message

```
Verilog module is not merged into the Liberty cell because the Verilog module is
     excluded from the command
```

Default Severity

Warning

Description

This is only triggered under flow `"-COPY_VERILOG_TO_LIBERTY"` with argument `"USE_VERILOG_FUNCTION_ALWAYS`".

When using the option `-exclude` to prevent the Verilog module function from merging to the Liberty cell, instantiating the liberty cells from excluded list will report the rule.

Example

```
Dofile:

    set_hdl_options -COPY_VERILOG_TO_LIBERTY USE_VERILOG_FUNCTION_ALWAYS -exclude
DFFQX1

     ...
```

When the design instantiates DFFQX1, IGN3.9b will be reported in the command "`read design`."

IGN3.9c

Message

```
Verilog module is not merged into the Liberty cell because the Verilog module is a
     black box
```

Default Severity

Warning

Description

This is only triggered under flow `"-COPY_VERILOG_TO_LIBERTY`" with argument `"USE_VERILOG_FUNCTION_ALWAYS`". When the Verilog module is a black box and copies this Verilog module to Liberty cell, instantiating the Liberty cell will report this rule.

Example

```
Library from Verilog:
   module DFFQX1(D, CK, Q);
      input  D, CK;
      output Q;
      wire  D, CK, IQN, Q;
   endmodule
```

When the design instantiates DFFQX1, IGN3.9c will be reported in the command "`read design`."

IGN3.9d

Message

```
Verilog module is not merged into the Liberty cell because the Verilog module is
     empty and it is not a tied cell
```

Default Severity

Warning

Description

This is only triggered under flow `"-COPY_VERILOG_TO_LIBERTY"` with argument "`USE_VERILOG_FUNCTION_ALWAYS`". When the Verilog module is empty and is not a tied cell and merges with the Liberty cell, instantiating the Liberty cell will report the rule.

Example

```
Library from Verilog:
   module DFFQX1(Pad, Top, Bot);
      inout Pad,Top, Bot;

      assign Pad = 1'b1;

   endmodule
```

When the design instantiates DFFQX1, IGN3.9d will be reported in the command "`read design`."

IGN3.9f

Message

```
Verilog module is not merged into the Liberty cell because the Liberty cell is not
     empty with USE_VERILOG_ONLY_IF_LIBERTY_EMPTY
```

Default Severity

Warning

Description

This is only triggered under flow "`-COPY_VERILOG_TO_LIBERTY`" with argument "`USE_VERILOG_ONLY_IF_LIBERTY_EMPTY`". When the Liberty cell is not empty and merges with the Verilog module, instantiating the Liberty cell will report the rule.

IGN3.9g

Message

```
Verilog module is not merged into the Liberty cell because the Liberty cell is a
     low power intent macro
```

Default Severity

Warning

Description

This is only triggered under flow "`-COPY_VERILOG_TO_LIBERTY`" with argument "`USE_VERILOG_FUNCTION_ALWAYS`".

When the Liberty cell is a low power intent macro and merges with the Verilog module, instantiating the Liberty cell will report the rule.

IGN3.9h

Message

```
Verilog module is not merged into the Liberty cell because the port of the Liberty
     cell is unmatched with the port of the verilog module
```

Default Severity

Warning

Description

This is only triggered under flow "`-COPY_VERILOG_TO_LIBERTY`" with argument "`USE_VERILOG_FUNCTION_ALWAYS`".

When the port numbers of Liberty cell and Verilog module are different or the port names are unmatched, instantiating the Liberty cell will report the rule.

IGN3.9i

Message

```
Verilog module is not merged into the Liberty cell because the Verilog module is a
     parameter/generic module and is not synthesized during read library
```

Default Severity

Warning

Description

This is only triggered under flow "`-COPY_VERILOG_TO_LIBERTY`" with argument "`USE_VERILOG_FUNCTION_ALWAYS`".

When Verilog module is a parameter/generic module and copies functions to Liberty cell, instantiating the Liberty cell will report the rule.

Example

```
Library from Verilog
   module DFFQX1(D, CK, Q);
      input  D, CK;
      output Q;
      parameter AX = 1;
      wire  D, CK, IQN, Q;

     _HDFF_verplex U$1(.Q(Q_temp), .QN(IQN), .S(1'b0), .R(1'b0), .CK(CK), .D(D));
      x_assign #(AX) uQ (Q, Q_temp);
   endmodule
```

When the design instantiates DFFQX1, IGN3.9i will be reported in command "`read design`."

IGN3.9j

Message

```
Verilog module is not merged into the Liberty cell because the Verilog file does
     not have such module
```

Default Severity

Warning

Description

This is only triggered under flow "`-COPY_VERILOG_TO_LIBERTY`" with argument "`USE_VERILOG_FUNCTION_ALWAYS`".

When there is no corresponding Verilog module with the same name as the Liberty cell, instantiating the Liberty cell will report the rule.

IGN4

Message

```
Attribute instance(s) are ignored
```

Default Severity

Warning

Description

Conformal has ignored one or more attribute instances.

Example

In the following example, Conformal ignores the attribute instance `(* a = b *)`. See line 4.

```
module test(a, b);
input a;
output b;
(* a = b *) assign b = a;
endmodule
```

IGN5.1

Message

```
Liberty state table is ignored with the -nostatetable option
```

Default Severity

Note

Description

A Liberty file contains a state table and is ignored due to the `-nostatetable` option specified in the `READ DESIGN` or `READ LIBRARY` command. Conformal does not support state tables.

Example

In the following example, while reading the Liberty library `lsi_10k.lib`, Conformal encountered 17 occurrences of the unsupported state table. Refer to the state table example below.

Input:

```
SETUP>read lib -lib lsi_10k.lib
// Parsing file lsi_10k.lib…
// Warning: (IGN5.1) Liberty State Table is not supported and is ignored
(occurrence:17)
```

State Table Example:

```
statetable ( "  D   CP ", " Q  QN") {
      table  : "  -  ~R  : - - :  N   N, \
                 H/L  R  : - - : H/L L/H";
}
```

IGN5.2

Message

```
Liberty attribute is ignored
```

Default Severity

Ignore

Description

The Liberty attribute is ignored.

Example

Some unsupported attributes are:

```
poly_template
hyperbolic_noise_above_high
hyperbolic_noise_low
hyperbolic_noise_high
steady_state_current_high
steady_state_current_low
steady_state_current_tristate
```

IGN5.3

Message

```
Liberty state table contains lowercase value(s) and is ignored
```

Default Severity

Warning

Description

The parser found lowercase value(s) in the truth table of the statetable. These must be changed to uppercase.

Example

In the following example, lines 2 and 3 shows that the edge value(s) '`~r`' and '`r`' are written in lowercase, which should be changed to uppercase:

```
statetable( " D CP" , " Q QN"  ) {
table : " - ~r : - - : N N,\
H/L r : - - : H/L L/H" ;
}
```

IGN5.4

Message

```
State_function attribute is ignored with the -nostatetable option
```

Default Severity

Note

Description

When the `READ LIBRARY` or `READ DESIGN` command uses the `-nostatetable` option, the `state_function` attribute of the Liberty file is ignored.

Example

In the following example, the Liberty file's `state_function` attribute is ignored because of the `-nostatetable` option.

Command:

```
REAd LIbrary -liberty -nostatetable test_lib.lib
```

File `test.lib`:

```
library (test_lib) {
 cell (test_cell) {
 pin (CLK) {
 direction : input;
 }
pin (E) {
 direction : input;
}
statetable ("CLK E", "EL") {
 table : "L L : - : L, \
 L H : - : H, \
 H - : - : N";
 }
 pin (EL) {
 direction : internal;
 internal_node : EL;
 }
 pin (GCLK) {
 direction : output;
 state_function : "CLK * EL"; //Ignored
 }
}
```

IGN5.5

Message

`Redefined Liberty attributes are detected. Subsequent definitions are ignored.`

Default Severity

Warning

Description

Triggered when a Liberty attribute has more than one definition. The newer one will be ignored.

Example

In the following example, the `direction` attribute is re-defined in the `pin` group. This definition will be ignored.

File `test_lib.lib`:

```
library (test_lib) {
define ("direction", "pin", "string");
}
```

IGN5.6

Message

```
Cells with no bias pins defined with sec_pdt_pin or its bias pins and PG pins name
     and configuration don't meet predefined criteria will be skipped for standard
     liberty format conversion.
```

Default Severity

Warning

Description

This rule is reported for cells being skipped for standard liberty format conversion if it falls in one of the following scenarios:

- When there is no sec_pdt_pin attribute associated with specific bias pin names VNW and VPW
- When the cell's PG pins name and configuration don't match any of following formats:

  1 primary power pin, 1 primary ground pin

  2 primary power pins VDD and VDDG, 1 primary ground pin

  1 primary power pin, 1 backup power pin, 1 primary ground pin

  1 internal power pin, 1 primary power pin, 1 primary ground pin

  1 primary power pins, 2 primary ground pins VSS and VSSG

  1 primary power pin, 1 primary ground pin, 1 backup ground pin

  1 primary power pin , 1 internal ground pin, 1 primary ground pin

  2 primary power pins VDD and VDDG, 2 primary ground pins VSS and VSSG

  1 primary power pin, 1 backup power pin, 1 primary ground pin, 1 backup ground pin

Example

In the follow example, cell A has pins defined with sec_pdt_pin attribute, however the pins name are neither VNW nor VNP; cell B has bias pins defined sec_pdt_pin attribute however the name of its two primary power pins don’t exactly match “VDD” and “VDDG”

```
Cell(A) {
    pg_pin(VDD) {
        pg_type:primary_power;
    }

    pg_pin(VDDG) {
        pg_type:primary_power;
    }
    sec_pdt_pin(BIASNW) { // pin name doesn't match "NVW"
        sec_pdt_pin_type: power;
    }
    sec_pdt_pin(BIASPW) {// pin name doesn't match "NPW"
        sec_pdt_pin_type: ground;
    }
}

Cell(B) {
    pg_pin(VDD) {
        pg_type:primary_power;
    }
    pg_pin(VDDE) { //pin name doesn't match "VDDG"
        pg_type:primary_power;
    }
    sec_pdt_pin(VNW) {
        sec_pdt_pin_type: power;
    }
    sec_pdt_pin(VPW) {
        sec_pdt_pin_type: ground;
    }
}
```

As a result, both cells don’t meet the conversion criteria and the following verbose message will be reported:

- Cells A(<Location of the cell>) with no bias pins defined with sec_pdt_pin or its bias pins and PG pins name and configuration don’t meet predefined criteria will be skipped for standard liberty format conversion.
- Cells B(<Location of the cell>) with no bias pins defined with sec_pdt_pin or its bias pins and PG pins name and configuration don’t meet predefined criteria will be skipped for standard liberty format conversion.

IGN5.7

Message

`Non-Standard Liberty attribute is ignored`

Default Severity

Warning

Description

This rule is reported for attribute being ignored for non-standard Liberty.

Example

In the following example, output_threshold_pct_fall and output_threshold_pct_rise are non-standard Liberty attribute.

```
Cell (A) {
  Pin(PAD){
    direction : inout;
    function : "din_0V8";
    is_pad : true;
    output_threshold_pct_fall : 30.001;     // ignored
    output_threshold_pct_rise : 69.999;    // ignored
    …
  }
  …
}
```

IGN6.1

Message

```
timeunit statement is ignored
```

Default Severity

Warning

Description

The `timeunit` statement is a SystemVerilog construct for simulation that Conformal does not support. When Conformal detects this statement in the syntax, it ignores the entire `timeunit` statement. Similarly, Conformal reports rule IGN6.2 for the `timeprecision` statement.

Example

In the following example, Conformal ignores the `timeunit` statement on line 2:

```
module test(aa, bb, o1, o2);
timeunit 1ps;
input aa, bb;
output o1, o2;
  assign o1 = aa == bb;
  assign o2 = aa != bb;
endmodule
```

IGN6.2

Message

`timeprecision statement ignored`

Default Severity

Warning

Description

The `timeprecision` statement is a SystemVerilog construct for simulation that Conformal does not support. When Conformal detects this statement in the syntax, it ignores the entire `timeprecision` statement. Similarly, Conformal reports rule IGN5.3 for the `timeunit` statement.

Example

In the following example, Conformal ignores the `timeprecision` statement on line 2:

```
module test(aa, bb, o1, o2);
timeprecision 0.1ps;
input aa, bb;
output o1, o2;
  assign o1 = aa == bb;
  assign o2 = aa != bb;
endmodule
```

IGN7.1

Message

`trireg net is modeled as a latch to hold value`

Default Severity

Warning

Description

A `trireg` net can model a charge storage node whose charge decays over time. The Conformal software does not support charge storage and charge decays. As a result, the software models the `trireg` net as a latch so that it can hold value.

Example

In the following example, Conformal ignores the `trireg` statement on line 3.

```
module test (clk, in1, out1);
  input clk;
  input trireg in1;
  output reg out1;

  always @(clk)
     out1 = in1;

  endmodule
```

IGN8.1

Message

```
Program block is ignored
```

Default Severity

Warning

Description

SystemVerilog program blocks are not supported. The tool ignores the entire program block.

Example

In the following example, Conformal ignores the program block (between `program...endprogram`):

```
program test(input wire data, output logic oo);
endprogram

module top();
logic data = 0;
wire oo;

test prg_test(data, oo);
endmodule
```

IMX Messages

This category of rules warns about intent mismatch.

IMX1.1

Message

```
Overflow bit discarded during expression evaluation
```

Default Severity

Warning

Description

Indicates that the evaluation of an arithmetic expression results in an overflow bit that is later discarded.

Example

In two examples below, the expression 'P1 + (-1)' is unsigned and is evaluated as '32'h3 + 32'hffffffff' which results in the overflow bit being discarded:

```
module test(in, out);
 input [31:0] in;
 output reg [31:0] out;

 parameter P1 = 32'd3;
 parameter PO = P1 + (-1); // IMX1.1: Overflow bit discarded

 always @*
  out = P1 + (-1) + in; // IMX1.1: Overflow bit discarded

endmodule
```

IMX1.2

Message

```
Index expression is set as an unsigned value during expression self-determined
     evaluation
```

Default Severity

Warning

Description

Indicates that the index expression is evaluated to be an unsigned expression.

Example

In the two examples below, the index expressions are evaluated as unsigned expression due to 'SZ' being unsigned. Consequently, 'P1' becomes a big positive number.

```
module test(in, sel, out);
 input  [15:0] in, sel;
 output [15:0] out;
 parameter P1 = -1;
 parameter SZ  = 32'd16;
 wire [(SZ+P1):0] r1;   // IMX1.2:(SZ+P1) evaluated as unsigned
 assign out=in[sel+P1]; // IMX1.2:(sel+P1)evaluated as unsigned
endmodule
```

Note: If P1 is positive, no violation is issued as there is no difference whether the expressions are evaluated as signed or unsigned.

IMX1.3

Message

```
Constant negative value converted to unsigned
```

Default Severity

Warning

Description

Indicates that a constant negative value is converted to an unsigned positive number

Example

In the example below, the right expressions of the shift operator are considered as unsigned, hence the results of the shift operations are always zero.

```
module test(in, out);
 input  [15:0] in;
 output [15:0] out;
 parameter BASE = 16'd3;
 parameter P3 = 3;
 parameter P4 = 4;
 assign out = BASE<<(P3-P4); // IMX1.3: left-shift by huge number
endmodule
```

IMX1.4

Message

```
Overflow bit discarded in assignment caused sign-reversal
```

Default Severity

Warning

Description

Indicates that an assignment of a signed constant expression value to a smaller-size, signed left-hand variable, has caused sign-reversal (from a positive number to a negative number, or vice versa).

Example

In the following example, the assignment of w1 triggers this rule:

```
module test(in, out);
 input  [15:0] in;
 output [15:0] out;
 parameter P128 = 128;
 parameter P256 = 256;
 wire signed [15:0] w1;
 assign w1 = (P128*P256)+1;
// IMX1.4: ((P128*P256)+1) == 32769
// but it is -32767 as 16-bit signed value
 assign out = in + w1;
```

endmodule

IMX1.5

Message

```
Multiple references to operand of increment/decrement or assign operator in
     assignment lhs
```

Default Severity

Error

Description

Indicates that the operand of an increment/decrement operator on the RHS of an assignment is the same variable assigned in the LHS of the assignment.

Example

```
module test(input [3:0] in, output reg [7:0] out);
  parameter N = 12;
  always @(in)
  begin
   int ff; ff = 0;
   for (int ii=4; ii < N; ++ii)
     ff = in + ff++; // IMX1.5 error
   out = ff;
end
endmodule
```

To clarify the design intent for the example above, the statement with the violating increment/ decrement operator can be split into multiple statements, such as (depending on the intent):

```
ff = in + ff; ff++;
```

or:

```
ff = in + ff;
```

IMX1.5a

Message

```
Multiple references to operand of increment/decrement or assign operator in
     expression. Order of evaluation not defined in System Verilog.
```

Default Severity

Warning

Description

Indicates that the operand of an increment/decrement or assign operator is referenced multiple times in an expression.

Example

```
module test(input clk, input [7:0] sel, input [7:0] a, b,
   output reg [7:0] out);
   reg [7:0] ii;
   always @(posedge clk)
      begin
         ii = 0;
         repeat (4) begin
            out += ii++ + ((ii == sel)? a : b); // <= IMX1.5a
      end
   end
endmodule
```

Clarify the design intent by splitting the statement into multiple statements, such as in the example below if the intent is to increment 'ii' after the statement in question:

```
   out += ii + ((ii == sel)? a : b);
   ii++;
```

IMX1.6

Message

```
Range expression has sign-mismatch
```

Default Severity

Warning

Description

Indicates that the range of the inside expression has sign mismatch.

Example

In the following example, the operands of the range expression on line 5 have different signs: the left operand is signed, but the right operand is unsigned.

```
module test(input signed [3:0] in, output reg [1:0] out);
  always @*
    begin
      case (in) inside
        [0:4'h9]: out = 2'b00; // line 5: IMX1.6 violation
        default: out = 2'b11;
      endcase
    end
endmodule
```

Note: The intent of the code above is ambiguous. To clarify the intent of the code, specify both operands of the range expression with the same sign. For example, if the intent of the code is unsigned value range between 0 and 9, the range can be rewritten as: [5'h0:5'h9].

IMX1.7

Message

```
Inside range is empty
```

Default Severity

Warning

Description

Indicates that the resulting range of the inside expression is empty. The expression is always false.

Example

In the following example, the range expression on line 5 is empty.

```
module test(input signed [3:0] in, output reg [1:0] out);
  always @*
    begin
      case (in) inside
      [3:0] : out = 2'b00; // line 5: IMX1.7 violation
      default: out = 2'b11;
      endcase
    end
endmodule
```

Note: Most likely, the intended range in the expression on line 5 above is between 0 and 3. If that is the intent, the code can be rewritten as follow:

```
case (in) inside
  [0:3] : out = 2'b00; // line 5: IMX1.7 violation
  default: out = 2'b11;
endcase
```

IMX1.8

Message

```
Leading bit(s) evaluate to '1'(s) due to operands resizing
```

Default Severity

Warning

Description

This message will be printed when operands resizing caused all the leading bits of an expression to be evaluated to '1's.

Example

In this example, the size of the comparison expression on line-7 is 32-bit due to the present of integer '0'. Therefore, the terms 'cond1' and 'cond2' are zero-extended to 32-bit. The expression '(cond1 ~^ cond2)' evaluates to '(28'hfff_ffff,(cond1 ~^ cond2))' triggering the IMX1.8 violation.

```
1  module test(cond1, cond2, in1, in2, out);
2  input [3:0] cond1, cond2,in1,in2;
3  output [3:0] out;
4  reg [3:0] out;
5
6  always @(cond1 or cond2 or in1 or in2) begin
7      if ((cond1 ~^ cond2) == 0) // IMX1.8
8          out = in1;
9      else
10          out = in2;
11    end
12  endmodule
```

Note: In the example above, the if-condition on line-7 is always false. To correct this situation, specify the correct size of the expressions in the condition, such as: 'if ((cond1 ~^ cond2) == 3'd0)'.

IMX1.9

Message

```
Expression with side-effect within short-circuit expression
```

Default Severity

Warning

Description

In a short-circuit expression evaluation, the evaluation stops once the result can be determined, and subsequent parts of the expression cannot be evaluated at all. This message will be printed if the potentially skipped sub-expression has side effects, such as an assign expression, non-pure function, or function with output.

Example

In the following example, IMX1.9 will be issued on line-13 since evaluation of the assign expression '(out2=func(sel))' is or might be dependent upon whether 'in' evaluates to true or false.

```
1  module test(input in, input [1:0] sel,
2              output reg out1, output reg [3:0] out2);
3
4      function automatic reg [3:0] func(input [1:0] x);
5        func = 4'b0;
6        func[x] = 1'b1;
7      endfunction
8
9      always @*
10      begin
11        out1 = 1'b0;
12        out2 = 4'b0;
13        if (in && (out2=func(sel)) )
14            out1 = 1'b1;
15      end
16  endmodule
```

It is recommended to recode the RTL that has a short-circuit expression with side-effects. Some tools may not honor the short circuit evaluation. Also, the short circuit evaluation side-effects might be unintentional.

IMX1.9a

Message

`Expression with side-effect within ternary expression`

Default Severity

Warning

Description

The true branch or false branch of a ternary expression contains expression that has side-effect.

Example

In the following example, the true branch of the ternary expression modify the value of variable 'out[2]' conditionally.

```
module test(input [1:0] in, output logic [2:0] out);

  function func(input func_in, output func_out);
      func = func_in;
      func_out = func_in;
  endfunction

  always_comb
     out[1:0] = (in == 0)? func(in[0], out[2]) : '0;
endmodule
```

IMX1.10

Message

```
Procedure 'always_latch' or 'always_comb' with edge sensitive event control
```

Default Severity

Error

Description

The 'always_latch' or 'always_comb' block contains ’posedge’ or ’negedge’ event control which cause it to behave as an edge triggered device (a flop).

Example

In the following example, the 'always_latch' special form indicates that it is a latch, but the "@(posedge clk)" clause on line 3 causes it to behave like a flop. Conformal will report IMX1.11 rule check violation.

```
1  module top(input logic clk, enable, address, data_in,output logic [1:0] buffer);
2
3   always_latch @(posedge clk)
4   begin
5       if (enable) buffer[address] <= data_in;
6   end
7 endmodule
```

IMX1.11

Message

```
Procedure 'always_ff' with irregular event control
```

Default Severity

Warning

Description

The 'always_flop' block has irregular event control(s), such as level sensitive event control, multiple event controls within the block.

Example

In the following example, the 'always_ff' special form on line 3 is expecting an edge sensitive event control such as '@(posedge clk)'. Instead, it is followed by a level sensitive event control '@(clk)'. Conformal will issue IMX1.11 rule check message.

```
1  module top(input logic clk, enable, address, data_in,output logic [1:0] buffer);
2
3   always_ff @(clk)
4   begin
5       if (enable) buffer[address] <= data_in;
6   end
7 endmodule
```

IMX1.12a

Message

`Stream expression size is smaller than the target size`

Default Severity

Warning

Description

The target variable is larger than the stream expression of a streaming concatenation.

Example

In the example below, the target variable 'q1' on line 5 and 'q2' on line 6 are of size 48 while the stream is of size 32. For both cases, Conformal will issue IMX1.12a.

```
1 module ptest
2   #(parameter int SZ1 = 32, int SZ2 = 48)
3     (input [SZ1-1:0] in, output [SZ2-1:0] q1, q2);
4
5  assign q1 = { >> 8 {in}};  // IMX1.12a reported
6  assign q2 = { << 8 {in}};  // IMX1.12a reported
7
8  endmodule
9
10 module test(input [31:0] in, output [47:0] q1, q2);
11  ptest #(.SZ1(32), .SZ2(48)) ptest(.in(in), .q1(q1), .q2(q2));
12 endmodule
```

```
Note: If the target variable of the streaming concatenation is larger than the
stream, the stream will be zero-filled on the right regardless of the streaming
operator.
```

IMX1.12b

Message

`Stream expression size is larger than the target size`

Default Severity

Error

Description

The stream expression of a streaming concatenation is larger than the target variable.

Example

In the example below, the target variable 'q1' on line 5 and 'q2' on line 6 are of size 16 while the stream is of size 32. For both cases, Conformal will issue IMX1.12b.

```
1 module ptest
2   #(parameter int SZ1 = 32, int SZ2 = 16)
3   (input [SZ1-1:0] in, output [SZ2-1:0] q1, q2);
4
5  assign q1 = { >> 8 {in}};  // IMX1.12b reported
6  assign q2 = { << 8 {in}};  // IMX1.12b reported
7
8 endmodule
9
10 module test(input [31:0] in, output [15:0] q1, q2);
11  ptest #(.SZ1(32), .SZ2(16)) ptest(.in(in), .q1(q1), .q2(q2));
12 endmodule
```

`Note: It shall be an error if the stream in a streaming concatenation is larger than the target variable.`

IMX1.12c

Message

```
Assigned stream size is smaller than source expression size
```

Default Severity

Warning

Description

The stream target in a streaming concatenation is smaller than the source expression.

Example

In the example below the assignment targets on line 9 and on line 10 are stream of type 'S1' of size 24, while the source expression 'in' are of size 32. For both cases, Conformal will issue IMX1.12c.

```
1 typedef struct packed {
2      byte a, b, c; // S1 is of size 24
3 } S1;
4
5 module ptest 6     #(parameter int SZ1 = 32)
7      (input [SZ1-1:0] in, output S1 q1, q2);
8
9      assign { >> 8 {q1}} = in;  // IMX1.12c reported
10      assign { << 8 {q2}} = in;  // IMX1.12c reported
11
12 endmodule
13
14 module test(input [31:0] in, output S1 q1, q2);
15      ptest #(.SZ1(32)) ptest(.in(in), .q1(q1), .q2(q2));
16 endmodule
```

```
Note: If the target stream of the streaming concatenation assignment is smaller
than the source expression, the bits will be consumed from the left regardless of
the streaming operator.
```

IMX1.12d

Message

```
Assigned stream size is larger than source expression size
```

Default Severity

Error

Description

The stream target in a streaming concatenation is larger than the source expression.

Example

In the example below the assignment targets on line 9 and on line 10 are stream of type 'S1' of size 96, while the source expressions 'in' are of size 32. For both cases, Conformal will issue IMX1.12d.

```
1 typedef struct packed {
2      int a, b, c; // S1 is of size 96
3 } S1;
4
5 module ptest
6     #(parameter int SZ1 = 32)
7      (input [SZ1-1:0] in, output S1 q1, q2);
8
9      assign { >> 8 {q1}} = in;  // IMX1.12d reported
10      assign { << 8 {q2}} = in;  // IMX1.12d reported
11
12 endmodule
13
14 module test(input [31:0] in, output S1 q1, q2);
15      ptest #(.SZ1(32)) ptest(.in(in), .q1(q1), .q2(q2));
16 endmodule
```

```
Note: It shall be an error if the target stream of the streaming concatenation
assignment is larger than the source expression.
```

IMX1.12e

Message

```
Stream expression element of type unpacked union may cause simulation/
     synthesis mismatch
```

Default Severity

Error

Description

Stream expression of an unpacked union with elements of different size can produce different results on different tools and hence is flagged as error.

Need to reference the actual elements in the union to resolve the issue.

Example

In the following packed union type example, the errors are shown by comments:

```
typedef union {
    // mismatch sizes here
    shortint sload;  // size 16
    byte bload[4]; // size 32
} Packet;
module test(output [31:0] out);
    Packet p2;
    assign p2.sload = 16'hcccc;
    assign out = {<<  {p2} }; // this expression can produce ambiguous results
endmodule
```

```
Note: One can re-write the expression as below to get expected result:
    assign out = {<<  {p2.sload} };
```

LEF

This category of rules applies to LEF macros.

LEF_PARSE1a

Message

```
No LEF macros defined
```

Default Severity

Note

Description

Indicates that there are no macros defined in the LEF file.

Example

In the following example, there are no LEF macros defined:

```
via myvia
viarule default ;
cutsize 0.1 0.1 ;
layers metal1 vial2 metal2 ;
cutspacing 0.1 0.1 ;
enclosure 0.05 0.01 0.01 0.05 ;
rowcol 1 2 ;
end myvia
```

LEF_PARSE1b

Message

`Missing END statement for LEF Macro definition`

Default Severity

Note

Description

Indicates there is no corresponding END statement for the LEF macro definition.

Example

In the following example, there is no END statement in LEF macro `LevShon`:

```
MACRO LevSh
  CLASS  CORE ;
  PIN A
    DIRECTION INPUT ;
  END A
  PIN Y
    DIRECTION OUTPUT ;
  END Y
  PIN VSS
    DIRECTION INOUT ;
    USE GROUND ;
  END VSS
  PIN VH
    DIRECTION INOUT ;
    USE POWER ;
  END VH
  PIN VL
    DIRECTION INOUT ;
    USE POWER ;
  END VL

MACRO LevLH
 ...
```

LEF_PARSE2a

Message

```
No LEF macro pins defined
```

Default Severity

Note

Description

Indicates that there is no pin defined in the LEF macro.

Example

In the following example, there is no LEF macro pin defined in macro `LevSh`:

```
MACRO LevSh
  CLASS  CORE ;

END LevSh
```

LEF_PARSE2b

Message

```
Missing END statement for LEF Macro Pin definition
```

Default Severity

Note

Description

Indicates that there is no corresponding END statement for the LEF macro definition.

Example

In the following example, there is no corresponding END statement with pins `Ei` and `Eo` in macro `LG_GSW`:

```
MACRO LP_GSW
  CLASS  CORE ;
  PIN Ei
    DIRECTION INPUT ;
  PIN Eo
    DIRECTION OUTPUT ;
  PIN Gi
    DIRECTION INOUT ;
    USE GROUND ;
   END Gi
 END LP_GSW
```

LEF1

Message

```
No power and no ground pins defined for macro
```

Default Severity

Note

Description

Indicates that there is no power and no ground pin defined in the macro.

Example

In the following example, there is no power and no ground pin defined in macro `LevIso`:

```
MACRO LevIso
 CLASS  CORE ;
 PIN A
  DIRECTION INPUT ;
 END A
 PIN En
  DIRECTION INPUT ;
 END En
 PIN Y
  DIRECTION OUTPUT ;
 END Y
 PIN VSS
  DIRECTION INOUT ;
 END VSS
 PIN VH
  DIRECTION INOUT ;
 END VH
 PIN VL
  DIRECTION INOUT ;
 END VL
END LevIso
```

LEF5

Message

`All pins of the macro defined as ground pins`

Default Severity

Note

Description

Indicates that there are only ground pins in the macro.

Example

In the following example, there are only ground pins, Gi and Go and no power and data pin in the macro LevIso:

```
MACRO LevIso
 CLASS  CORE ;
 PIN Gi
  DIRECTION INOUT ;
  USE GROUND ;
 END Gi
 PIN Go
  DIRECTION INOUT ;
  USE GROUND ;
 END Go

END LevIso
```

LEF6

Message

`All pins of the macro defined as power pins`

Default Severity

Note

Description

Indicates that there are only power pins in the macro.

Example

In the following example, all the pins in the macro are power pins:

```
MACRO LevIso
 CLASS CORE ;
 PIN Vi
  DIRECTION INOUT ;
  USE POWER ;
 END Vi
 PIN Vo
  DIRECTION INOUT ;
  USE POWER ;
 END Vo
END LevIso
```

LEF7

Message

`Pin missing direction statement`

Default Severity

Note

Description

Indicates that there is no direction statement in the pin.

Example

In the following example, there is no direction statement in `PIN Y`:

```
MACRO LevIso
 CLASS  CORE ;
 PIN Y
 END Y
 PIN Vo
  DIRECTION INOUT ;
  USE POWER ;
 END Vo
END LevIso
```

LRM

This category refer to rules available with stricter LRM compliance check as defined by the set_hdl_option -LRM_Compliance option.

LRM1.1

Message

`Forward reference of a hierarchy identifier is not recommended`

Default Severity

Error

Description

The design includes an implicit declaration of structure variable which is not recommended.

Example

In the following example, the design assigns structure variable tmp first and declares it latter. See lines 6 and 7.

```
1 typedef struct  {
2    logic f;
3 } my_t;
4 module top(input  in, output  out);
5    assign a = in;
6    assign tmp.f = in;
7    my_t tmp;
8 endmodule
```

LRM1.2

Message

`Function connection can not be implicitly declared`

Default Severity

Error

Description

The design includes an implicit declaration as a function connection which is not recommended.

Example

In the following example, the design assign structure variable tmp first and declare it latter. See line 6.

```
1 module Top(output out, input in);
2    function int func ( logic a,logic b);
3       return a|b;
4    endfunction
5    assign y = in;
6    assign out = func(x, y);
7 endmodule
```

LRM1.3

Message

`Streaming operator in non-assignment operation is not recommended`

Default Severity

Error

Description

The design includes usage of a streaming operator that is not a direct assignment statement but is part of a ternary statement for example, which is not recommended.

The severity can be changed to an error by the option: 'set hdl option -lrm_compliance on'.

Example

In the following example, the statement on line 7 uses a streaming operator to assign 'in' to 'out' within a ternary expression.

```
1   module test(out,in, ctrl);
2    output logic [7:0] out;
3    input logic [7:0] in;
4    input bit ctrl;
5
6    // out is being assigned the value of 'streaming' operator in a ternary
statement
7    assign {>>{out}} = (ctrl) ? {>>{in}} : {$bits(out){1'b0}};
5   endmodule
```

LRM2

Message

```
Combination of net declaration(s) and net assignment declaration(s) in a
     net declaration list only supported in System Verilog
```

Default Severity

Error

Description

In a verilog design, a net declaration list contains both net declaration(s) and net assignment declaration(s). This is legal in System Verilog, but illegal in Verilog.

A design will be red in as a System Verilog design if the ’read_design’ command is specified with one of the following options: '-SV', '-SYStemverilog', '-SV09', '-SV12', or '-SVA'.

Example

In the following example, the net declaration list includes both net declaration for 'a' and net assignment declaration 'b = ~in'.
See line 2.

```
1 module Top(output out, input in);
2    wire a, b = ~in;
3    assign a = in;
4    assign out = a | b
5 endmodule
```

Miscellaneous Messages

This category of rules are for miscellaneous messages that are not classified to any other category. The following lists the MSG rule checks:

ERR_IN_PROTEC

Message

```
Error in protected file
```

Default Severity

Error

Description

A rule violation occurred that pertains to an encrypted or protected file or module.

Example

For example:

```
read design protect.vp
```

You would receive this message if an error occurred against `protect.vp`, as this file is encrypted/protected.

PARSE_ERROR

Message

```
Parsing syntax error
```

Default Severity

Error

Description

The input file has a syntax error.

Example

In the following example, it lacks of a semicolon in the end of the assignment.

```
module syntax_error(AAA, BBB);
  input AAA;
  output BBB;
  assign BBB = AAA //; make this line syntax error
endmodule
```

PARSE_WARN

Message

```
Parsing syntax error in nontranslate module
```

Default Severity

Warning

Description

The input file has a syntax error in a notranslate module. The severity is Warning because the error occurs in a notranslate module.

Example

The following example is missing a semicolon at the end of an assignment. If the `syntax_error` module is added to a notranslate module before reading in the design, the module triggers the `PARSE_WARN` message.

```
module syntax_error(AAA, BBB);
  input AAA;
  output BBB;
  assign BBB = AAA ;// make this line syntax error
endmodule
```

Register Transfer Level

This category of rules applies to designs that are written in the register transfer level of abstraction. The following lists the Register Transfer Level (RTL) rule checks.

- RTL13.5 on page 454
- RTL13.6 on page 455
- RTL13.7 on page 457
- RTL13.8 on page 458
- RTL13.8a on page 459
- RTL13.9 on page 460
- RTL13.10 on page 461
- RTL14 on page 463
- RTL14.1 on page 464
- RTL15 on page 465
- RTL15.1 on page 466
- RTL15.2 on page 467
- RTL15.2a on page 468
- RTL15.3 on page 469
- RTL15.4 on page 470
- RTL15.4a on page 471
- RTL16.1 on page 473
- RTL16.2 on page 474
- RTL16.3 on page 475
- RTL17 on page 476
- RTL17.1 on page 477
- RTL17.2 on page 478
- RTL17.3 on page 479
- RTL17.4 on page 480
- RTL17.5 on page 482
- RTL18.1 on page 483
- RTL18.2 on page 484

RTL1.1

Message

```
Variable/signal is assigned by more than one concurrent statement
```

Default Severity

Warning

Description

The design includes one or more cases where a variable or signal is assigned by two or more concurrent statements.

Example

In the following example, the design concurrently assigns output `out0.d0`. See lines 9 and 10. Note that for line 11,

```
1   typedef struct packed {
2       reg d0, d1;
3   } pkt_t;
4
5 module SEN(sel,in0,in1,in2,out0);
6       input [2:0] sel;
7       input in0, in1, in2;
8       output wire pkt_t out0;
9)  assign out0.d0 = sel[0]? in0 : 1'bz;
10  assign out0.d0 = sel[1]? in1 : 1'bz;11;
11  assign out0.d1 = sel[2]? in2 : 1'bz;
12 endmodule
```

Note: The **set wire resolution <and | or | wire>** command can be used to select the wire resolution function. By default, the wire resolution function is set to **and**.

RTL1.1a

Message

```
Output of always_comb/always_latch/always_ff assigned in multiple concurrent
     statements
```

Default Severity

Error

Description

Indicates that the variable on the left-hand side of assignment in an always_comb, always_latch, or always_ff block is also assigned in other concurrent statement(s).

Example

In the following example, variables *r1* and *r2* are assigned in the *always_comb* block at line 5 and line 6, and also in the *always @** block on line 11 and line 12.

```
1  module test (input in1, in2, output logic out1, out2);
2      reg r1, r2;
3      always_comb
4      begin
5          r1 = in1 ^ in2;
6          r2 = in2 & in1;
7          out1 = r1 | r2;
8      end
9      always @*
10      begin
11          r1 = in2 & in1;
12          r2 = in1 ^ in2;
13          out2 = r1 & r2;
14      end
15  endmodule
```

Note: the System Verilog *always_comb, always_latch,* and *always_flop* constructs especially prohibit any variable assigned on the lhs to be written by any other processes. If the intention for variables *r1* and *r2* are to be used locally as temporary variables, then the declaration *reg r1*, *r2;* need to be put within or local to each always block.

RTL1.2

Message

`Variable/signal is assigned by multiple non-blocking assignments`

Default Severity

Warning

Description

The design includes one or more cases where a variable or signal is assigned by two or more non-blocking assignments.

Example

In the following example, variable `out0` is assigned by a non-blocking assignment. See lines 7 and 8.

```
1  module VLGT (clk,rst,in0,in1,out0);
2  input clk,rst,in0,in1;
3  output out0;
4  reg out0;
5  always @(posedge clk)
6  begin
7    out0 <= in0 & in1;
8    out0 <= in1 & in0;
9  end
10 endmodule
```

Note: In the example above, only one flop will be created and the last non-blocking assignment will overwrite all previous non-blocking assignments.

RTL1.2a

Message

`Variable/signal on instance INOUT port might have multiple drivers`

Default Severity

Warning

Description

The variable or signal on the INOUT instance port might have multiple drivers.

Example

In the following example, in line 4 , ports `VSSA` and `VSSG` are defined as `INOUT` ports in the library file:

```
module mod1_G(Y,A,VDDA,VSSA,VDDG,VSSG);
  input   A, VDDA, VSSA, VDDG, VSSG;
  output  Y;
  KAQ2 I0(.Y(I0_Y), .A(A),    .VDD_A(VDDA), .VSS_A(VSSA), .VDD_G(VDDG), \
    .VSS_G(VSSG));
  KAQ2 I2(.Y(Y),    .A(I0_Y), .VDD_G(VDDG), .VSS_G(VSSG), .VDD_A(VDDA), \
    .VSS_A(VSSA));
endmodule
```

RTL1.3

Message

```
Variable/signal is assigned by both blocking and non-blocking assignments in the
     same block
```

Default Severity

Error

Description

The design includes one or more cases where a variable or signal is assigned by both blocking and non-blocking assignments.

Example

On line 7 of the following example, variable `out0` is assigned by a *blocking* assignment, and on line 9, variable `out0` is assigned by a *non-blocking* assignment.

```
module VGT (rst,in0,in1,out0);
input rst,in0,in1;
output out0;
reg out0;
always @(rst or in1 or in0) begin
  if (rst)
    out0 = in1 & in0;
  else
    out0 <= in0 & in1;
end
endmodule
```

RTL1.3a

Message

```
Variable/signal is assigned by both blocking and non-blocking assignments in
     different blocks
```

Default Severity

Warning

Description

The design includes one or more cases where a variable or signal is assigned by both blocking and non-blocking assignments in different blocks.

Example

On line 5 of the following example, variable out0 is assigned by a non-blocking assignment in one block, and on line 7, variable out0 is assigned by a blocking assignment in a different block.

```
module VGT(input clk, ain, bin, output reg zout);
reg in1;
always @(posedge clk)
begin
  in1 <= ain;
end
always @(posedge clk)
  in1 = bin;
always_comb
  zout <= in1;
endmodule
```

RTL1.4

Message

```
Assignment with LHS bit width is greater than RHS bit width
```

Default Severity

Warning

Description

The design includes one or more assignments with a left-hand-side (LHS) bit width greater than the right-hand-side (RHS) bit width.

Example

In the following example, the assignment of `in0` to `out0` has mismatched bit widths. See line 6.

```
module VT (clk,in0,out0);
input clk,in0;
output [3:0] out0;
reg [3:0] out0;
always @(posedge clk) begin
  out0 <= in0;
end
endmodule
```

RTL1.4a

Message

`Bit width mismatch in VHDL assignment`

Default Severity

Error

Description

In an VHDL assignment, the left-hand-side (LHS) and right-hand-side (RHS) bit widths must match.

Example

The following example triggers this error message, because the widths of the RHS and LHS of the assignment on line 9 do not match:

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity top is
4   port (ii: in std_logic_vector(2 downto 0);
5         oo: out std_logic_vector(3 downto 0));
6 end top;
7 architecture rtl of top is
8 begin
9   oo <= ii;
10 end rtl;
```

RTL1.5a

Message

```
Assignment with RHS bit width is greater than LHS bit width
```

Default Severity

Warning

Description

The design includes one or more assignments with a right-hand-side (RHS) bit width greater than the left-hand-side (LHS) bit width.

Example

1. In the following example, the design assigns a two-bit vector logic `2'b01` to a one-bit output `out0` (see line 3).

   ```
   module test(out0);
   output out0;
   assign out0 = 2'b01;
   endmodule
   ```

2. In the following Verilog example, the self-determined bit width for the RHS expression is 5, while the LHS is only 4.

   ```
   x[3:0] = a[4:0] + b [4:0];
   ```

RTL1.5b

Message

```
Potential loss of RHS msb or carry-out bit
```

Default Severity

Warning

Description

The design includes one or more assignments where there is potential for the right-hand-side (RHS) to lose its most-significant-bit (MSB) or carry-out bit.

Example

1. In the following example, the self-determined bit width of the RHS is 4 and the LHS is also 4; however, `x[3:0]` does not take the potential carry-out from the RHS.

   ```
   x[3:0] = a[3:0] + b[3:0];
   ```

2. In the following Verilog example, the self-determined bit width for the RHS is 4 and the LHS is also 4; however, arithmetically, the product is 8 bits and `x[3:0]` only takes the lower 4 bits of the total 8 bits.

   ```
   x[3:0] = a[3:0] * b[3:0];
   ```

RTL1.5c

Message

`Index computation might overflow the resolved expression size`

Default Severity

Warning

Description

Indicates that when computating the index in a bit-select or a part-select expression, the bit-width of the index expression is not enough to hold the result and some significant bits might be lost.

Example

In the following example, the index expression on line 4 `'5'h3*bss'` is of size '5' `(max(sizeof(5'h3),sizeof(bss)) = max(5,4) = 5)`. However, the maximum value of the index expression is '45' `(5'h3 * max_value(bss) = 3 * 15 = 45)`, which needs 6-bits to represent.

```
1 module test(input [47:0] arr, input [3:0] bss, output reg [2:0] out);
2    always @(arr or bss)
3    begin
4       out = arr[5'h3*bss +: 3]; // line 4
5    end
6 endmodule
```

Note: To resolve this violation, specify sufficient bits to perform the index calculation correctly. The index expression on line 4 above can be recoded as follow to be of size '`6-bit`':

```
3 ...
4       out = arr[6'h3*bss +: 3];
5 ...
```

RTL1.5d

Message

`Index computation might cause unsigned integer underflow`

Default Severity

Warning

Description

Indicates that the index of a bit-select or a part-select expression resolved to an unsigned integer, but the result of the computation might be a negative number and cause unsigned integer underflow.

Example

In the following example, the index expression 'idx - 4'b1' on line 4 is unsigned, but can result in a negative number when the value of 'idx' is 4'b0. This will trigger the RTL1.5d violation.

```
1  module test(input [15:0] in, input [3:0] idx, output reg out);
2      always @*
4         out = in[idx -4'b1]; // RTL1.5d violation
6  endmodule
```

To avoid this violation, specify the condition under which the index is valid;

```
1  module test(input [15:0] in, input [3:0] idx, output reg out);
2      always @*
4         if (idx >= 4'b1)
5            out = in[idx -4'b1];
6         else
7            out = 1'b0;
9  endmodule
```

RTL1.5e

Message

```
Lhs bit width is insufficient in shift expression
```

Default Severity

Warning

Description

The number of bits on the lhs of a shift operator (<<. <<<. >>. >>>) is insufficient for the shift operation.

Example

In the following example, in module ’bot’, the value of parameter ’P2’ is 4, and the size of parameter ’P1’ is 1-bit as inherited from its parameter override value of 1’b1. Hence, the shift operator on line 3 is performing a 4-bit right shift operation on a 1-bit value. The result is always '0'.

```
1  module bot #(parameter P1 = 8'hff, parameter P2 = 1)
2              (input in1, output out1);
3      localparam P3 = P1 >> P2; // IMX1.10
4      assign out1 = in1 + P3;
5  endmodule
6
7  module test(input  in, output out);
8      bot #(.P1(1'b1), .P2(4)) B1(.in1(in),.out1(out));
9  endmodule
```

RTL1.6

Message

```
Blocking assignment is in sequential always block
```

Default Severity

Warning

Description

The design includes one or more blocking assignments in a sequential *always* block.

Example

On lines 7 and 9 of the following example, the design uses blocking assignments:

```
module VLGT (clk,rst,in0,in1,out0);
input clk,rst,in0,in1;
output out0;
reg out0;
always @(posedge clk) begin
  if (rst)
    out0 = in1 & in0;
  else
    out0 = in0 & in1;
end
endmodule
```

Note: Using blocking assignment in sequential always block can cause race condition in simulation. To avoid this warning, always use non-blocking assignment in sequential always block, unless the assignment is to a variable local to the always block

RTL1.7

Message

```
Non-blocking assignment is in combinational always block
```

Default Severity

Warning

Description

The design includes one or more non-blocking assignments in a combinational *always* block.

Example

On lines 7 and 9 of the following example, the design uses non-blocking assignments:

```
module VT (sel,in0,in1,out0);
input sel,in0,in1;
output out0;
reg out0;
always @(sel or in1 or in0) begin
  if (sel)
    out0 <= in1 & in0;
  else
    out0 <= in0 & in1;
end
endmodule
```

RTL1.8

Message

```
Latch is assigned by blocking assignments
```

Default Severity

Warning

Description

The design includes one or more latches that are coded using a *blocking* assignment. We recommend that sequential elements be coded using a non-blocking assignment rather than a blocking assignment.

Example

In the following example, latch `Q` is coded using a blocking assignment (see line 7):

```
module Dlat(data, en, Q);
input data, en;
output Q;
reg Q;

always @( en or data)
        if (en) Q  = data;
endmodule
```

RTL1.9

Message

`Parameter bit width does not match RHS bit width`

Default Severity

Warning

Description

The design includes one or more parameter bit widths that do not match with right-hand-side (RHS) bit widths.

Example

In the following example, parameter `my_values` is declared as a 3-bit vector, but was assigned a 4-bit vector (see line 3):

```
module test (Z);
output [3:0]Z;
parameter signed [2:0] my_values = 4'b1010;
assign Z = my_values;
endmodule
```

RTL1.9a

Message

```
Parameter/Generic is set from command line
```

Default Severity

Note

Description

Indicates that the value of a design parameter or generic was set from the command line.

Example

In the following example, the tool issues this message because the value of parameter "`my_values`" was set to 6 fron the command line:

```
// test.v
module test(Z);
output [3:0] Z;
parameter my_values = 0;
  assign Z = my_values;
endmodule

// In Conformal dofile
read design test.v -parameter -int my_values 6 -root test
```

RTL1.9b

Message

```
Parameter value in an instantiation is not within the valid range
```

Default Severity

Warning

Description

Indicates that the instance parameter data is out of the valid value range of the parameter declaration.

Example

In line 2 of the following example, p1 is declared as a 4-bit signed parameter (valid data range is from -8 to 7). However, in line 11, p1 is associated with a 4-bit unsigned value (15) which is out of p1's valid value range.

```
1 module sub(aa, oo);
2 parameter signed [3:0] p1 = -1;
3 input aa;
4 output oo;
5   assign oo = aa && (p1 < 0);
6 endmodule
7
8 module t1(aa, oo);
9 input aa;
10 output oo;
11   sub #(.p1(4'b1111)) u1(.aa, .oo);
12 endmodule
```

Note: Only checks single-bit/simple array type parameter data.

RTL1.9c

Message

```
Parameter bit width is sufficient to hold RHS value but does not match RHS bit width
```

Default Severity

Warning

Description

Indicates that there is a size mismatch between the left-hand side (LHS) and right-hand side (RHS) of a parameter assignment, but the LHS is large enough to hold the RHS value.

Example

In the following example, the LHS and RHS of the parameter assignment in line 4 has a size mismatch (LHS is 6, RHS is 32). However, the RHS expression evaluates to the value '32', which will fit into the LHS of size '6'. Hence, RTL1.9c will be issued instead of RTL1.9.

```
module test (z);
output [5:0]z;
 parameter int a = 256;
 parameter logic [5:0] b =  a >> 3; // line 4
 assign z = b;
endmodule
```

RTL1.9d

Message

`Parameter/Generic resolution has error`

Default Severity

Error

Description

Indicates that parameter resolution at a child instance has failed or has error. This rule check points to module instance statement.

Example

In the following example, passing value ‘0’ to parameter ‘RC’ in instantiation statement on line 12 causes zero replication error in the assignment to parameter RSTVAL in the child module on line 3. Rule check message RTL1.9d will be issued on the instantiation statement on line 12. Note that another rule check message (SV5.3 NULL expression is not allowed) will be issued on the parameter assignment on line 3.

```
1  module dut #(
2       parameter integer RC = 16,
3       parameter RSTVAL = {RC{1'b1}}
4      )
5      (input wire [RC-1:0] d,
6       output reg [RC-1:0] q
7      );
8      assign q = RSTVAL & d;
9  endmodule
10
11  module test(input [3:0] d, output [3:0] q);
12     dut #(.RC(0)) dut (.q(q), .d(d));
13  endmodule
```

RTL1.9e

Message

```
Parameter override has illegal combination
```

Default Severity

Error

Description

The parameter value or type override to the actual parameter is illegal.

Example

In the following example, the parameter override is a type parameter while the actual parameter is a regular parameter.

```
typedef int myT1;
typedef int myT2;

module bot #(parameter myT1 P1 = '0, parameter type T1 = myT1)
            (input T1 in1, output T1 out1);
    assign out1 = in1 + P1;
endmodule

module test(input  myT1 in, output myT1 out);
    bot #(myT1) B1(.in(in),.out(out));
endmodule
```

Note: Conformal will issue RTL1.9e rule violation, followed by additional message describing the problem. For example

```
// Error: RTL1.9e: Parameter override has illegal value
//  Type override to regular parameter 'P1'
//  in file 'test.sv'
```

RTL1.9f

Message

```
Defparam overrides local parameter
```

Default Severity

Note

Description

The defparam statement overrides a local parameter from the same module.

Example

In the following example, the ’defparam’ statement on line 3 overrides the value of parameter ’A’ from the same module. The value of parameter ’A’ is set to ’2’.

```
1 module test(output [31:0] foo);
2   parameter A = 1;
3   defparam A = 2;
4   bar #(.B(a)) bar(foo);
5 endmodule
```

RTL1.9g

Message

`Defparam overrides module instance parameter`

Default Severity

Warning

Description

Parameter value specified by defparam statement overrides the parameter override value specified in the module instantiation statement

Example

In the example below, the value of parameter 'RAND_INIT' in module instance 'ptest_u' assigned by the defparam statement on line 13 overrides the value specified by the module instantiation statement on line 14.

```
1  module ptest(input clk, rst, input [3:0] d, output reg [3:0] q);
2      parameter RAND_INIT = 0;
3      always @(posedge clk or negedge rst)
4      begin
5         if (!rst)
6             q <= RAND_INIT;
7         else
8             q <= d;
9      end
10  endmodule
11
12  module test(input clk, rst, input [3:0] d, output [3:0] q);
13      defparam  ptest_u.RAND_INIT = 4'b0101;
14      ptest #(.RAND_INIT(4'b0001)) ptest_u(.*);
15  endmodule
```

RTL1.11

Message

```
Variable is assigned by both continuous and procedural statements
```

Default Severity

Error

Description

A variable is assigned by both continuous and procedural statements.

Example

In the following example, variable `t1` is assigned in lines 5 and 8.

```
module test (a, b,t1);
input a,b ;
output t1 ;
logic t1 ;
assign  t1 = a & b;
always @(a or b )
  begin
    task1 (t1);
end
task  task1 (output c);
  begin
    c = a | b;
  end
endtask
endmodule
```

RTL1.12

Message

`Variable(SV:logic/bit) is assigned by multiple continuous statements`

Default Severity

Error

Description

A variable is assigned by multiple continuous statements.

Example

In the following example, variable `t1` is assigned in lines 5 and 6.

```
module test (a, b,t1);
input a,b ;
output t1 ;
logic t1 ;
assign  t1 = a & b;
assign  t1 = a | b;
endmodule
```

RTL1.12a

Message

```
Members of union are assigned by multiple continuous statements. This can cause a
     mismatch between synthesis and simulation
```

Default Severity

Warning

Description

Different members of a variable of type union are assigned by multiple continuous statement. Since a variable does not have wire resolution function associated with it, the value of the variable potentially depends on the order of execution.

Example

In the following example, '`a[0]`' is a union and assignment to '`a[0].f1`' on line 7 and assignment to '`a[0].f2`' on line 8 are in face overlapping. In this case, Conformal will issue `RTL1.12a` rule check message.

```
1  typedef enum { one=1, two=2, three=3, four=4 } e_type;
2  module test(input [1:0] in, output e_type out);
3    localparam e_type v4 = one;
4    assign out = v4.next(in);
5  endmodule
```

```
1  module test( input int in0, output int out [1:0]) ;
2    typedef union {
3       int f1; shortreal f2;
4    } value_t;
5    value_t a[1:0];
6
7    assign a[0].f1 = in0;
8    assign a[0].f2 = 1.0;
9
10    assign a[1].f1 = in0;
11    assign out = {a[0].f1, a[1].f1};
12
13  endmodule
```

RTL1.13

Message

`Mismatched enumeration types are in the assignment`

Default Severity

Error

Description

There are mismatched enumeration types in the assignment.

Example

In the following example, line 4 has a mismatched enumeration type (`my_curve` and 2).

```
1  typedef enum {circule, ellipse, freeform} Curve;
2  module test(output Curve my_curve);
3    always_comb
4        my_curve = 2;
5  endmodule
```

RTL1.13a

Message

```
Different enumeration types with similar contents are in the assignment
```

Default Severity

Warning

Description

The lhs and rhs of an assignment are enum of different types. However, both have the same name, contents, and types.

Example

In the following example, module ’bottom' in file 'bottom.sv' and module ’mid' in file 'mid.sv' have references type 'tile_t' from include file 'define.sv'. However, the two files 'bottom.sv' and 'mid.sv' are from two separate 'read_design' commands, and hence the type 'tile_t's in module ’bottom' and in module ’mid' are different types from different compilation units. Conformal will report RTL1.13a on assignment to parameter 'TILE' when instantiating module 'bottom' from module 'mid'.

```
File: dofile

read_design -sv bottom.sv -noelab
read_design -sv test.sv mid.sv -noelab
elaborate -root test

File: define.vh


...
typedef enum int {
    TILESIZE_32X4 = 2, TILESIZE_8X4  = 1, TILESIZE_4X4  = 0
} tile_t;

//File: bottom.sv

`include "define.vh"
module bottom
  #(parameter SZ1 = 6, SZ2 = 6, tile_t TILE = TILESIZE_32X4)
   (input [SZ1-1:0] in, output [SZ2-1:0] out);
  assign out = (in >> TILE) << TILE;
endmodule
```

```
File: mid.sv

`include "define.vh"
module mid
  #(parameter SZ1 = 4, SZ2 = 4)
   (input [SZ1-1:0] in, output [SZ2-1:0] out);
  localparam tile_t TILE = (SZ1 > 4)? TILESIZE_8X4 : TILESIZE_4X4;
  bottom #(.SZ1(SZ1), .SZ2(SZ2), .TILE(TILE))
         bottom_u (.in(in), .out(out));
endmodule
```

RTL1.14

Message

`Mismatched enumeration types are in the expression`

Default Severity

Warning

Description

There are mismatched enumeration types in the expression.

Example

In the following example, in line 6, enumeration types are mismatched.

```
module test (output integer out1) ;
enum {IDLE=2'b00, S1, S2} state;
assign state = S1;
always @*
begin
  out1 = (state == 4)? 2'b00:state;
end
endmodule
```

RTL1.15

Message

```
Mismatched types between expression and target
```

Default Severity

Error

Description

There are mismatched types in the assignment.

Example

In the following example, the error is reported because of the assignment of mismatched types between `oo` (type `t1_t`) and `aa` (type `t2_t`) on line 7:

```
1 typedef struct {bit a, b, c;} t1_t;
2 typedef struct {bit a, b, c, d;} t2_t;
3
4 module t1(aa, oo);
5 input t1_t aa;
6 output t2_t oo;
7 assign oo = aa;
8 endmodule
```

RTL1.16

Message

`Array shape/size mismatch`

Default Severity

Warning

Description

There is a bit width array mismatch.

Example

```
library ieee;
use ieee.std_logic_1164.all;
entity top is
  port (z: out std_logic;
         a: in std_logic_vector(3 downto 0) );
end top;
architecture rtl of top is
   function LSB(x: std_logic_vector(7 downto 0)) return std_logic is
   begin
    return x(0);
  end LSB;
begin
   z <= LSB(a); -- bit width of a and x mismatch
end rtl;
```

RTL1.17

Message

`Comparison expression operands types mismatched`

Default Severity

Error

Description

There are mismatched types in the comparison expression.

Example

The following example triggers this error because the comparison expression in line 6 has mismatched types; var1 is an unpacked type S2 and 10 is an integer type:

```
1 typedef struct { bit [7:0] bit8; } S1;
2 typedef S1 S2;
3 module test(input S2 var1, output shortint out1);
4   always_comb
5   begin
6     if (var1 == 10) begin
7       out1 = 1;
8     end
9   end
10 endmodule
```

RTL1.19

Message

```
Conditional operator true and false expression do not have equivalent data types
```

Default Severity

Error

Description

Incompatible types in the conditional operator true and false expressions.

Example

The following example triggers this error because the conditional operator has incompatible types in true and false expressions in line 7:

```
1 module test(sel, in1, in2, out);
2   input sel;
3   input [1:0] in1;
4   input in2 [0:1];
5   output out [0:1];
6
7   assign out = sel ? in1 : in2;
8
9 endmodule
```

RTL1.20

Message

```
Multiple assignment to the same variable in concatenation
```

Default Severity

Warning

Description

The same variable or signal is included more than once in the left-hand-side concat expression of an assignment statement.

Example

In the following example, the variable 'out[6]' is assigned more than once in the same assignment statement.

```
module test (in, out, out6);
 input  [7:0] in;
 output reg [6:0] out;
 output reg out6;

 always @(in)
 {out[6], out} = in;
endmodule
```

RTL2.1

Message

```
Variable is referenced before the assignment. This can cause a mismatch between
     synthesis and simulation.
```

Default Severity

Warning

Description

The design includes one or more variables that are referenced before they are assigned. This can cause a mismatch between simulation and synthesis. In synthesis tools and also Conformal, the current value of the variable is used. In simulation, the previously assigned value is used.

Example

On line 8 of the following example, the design references variable `sig1`, but does not assign it until line 9:

```
module TPS (clk,in1,in2,out0);
input clk,in1,in2;
output out0;
reg    out0;
  always @ (in1 or in2)
    begin
      reg sig1;
      out0 = sig1;
      sig1 = in1 | in2;
    end
endmodule
```

RTL2.1a

Message

```
Variable is referenced before assignment in subprogram. Possible simulation
     mismatch
```

Default Severity

Warning

Description

For initialized variable in subprogram, simulation mismatch might occur when the variable is referenced before the assignment. For the Conformal Equivalence Checking software and synthesis tools, the variable will default to 0. However, for simulation, the variable assumes previously assigned value.

Example

In the following example, line 16 references a variable 'data' before its assignment. This can cause a simulation or synthesis mismatch.

```
1  module foo(current, data);
2  input current;
3  output [1:0] data;
4  function [1:0] generate_data;
5  input current;
6  reg next;
7  reg[1:0] data;
8  begin
9      case (current)
10        1'b0:
11           if(next) begin
12               data = 2'h1;
13           end
14        1'b1: next = 2'h1;
15      endcase
16      generate_data = data;
17  end
18  endfunction
19    assign data = generate_data(current);
20  endmodule
```

RTL2.2

Message

`Variable is referenced but never assigned`

Default Severity

Warning

Description

The design includes one or more variables that are referenced but never assigned in the design.

Example

In the following example, the design references variable `da0`, but never assigns it:

```
module SEN (clk,rst,out0);
input clk,rst;
output out0;
reg    out0;
reg da0;
 always @ ( posedge rst or posedge clk )
   begin
     if (rst) begin
         out0 <= 1'b0;
     end
     else begin
          out0 <= da0;
     end
   end
 endmodule
```

RTL2.3

Message

```
Externally defined signal reference is not supported
```

Default Severity

Warning

Description

Conformal did not support one or more externally defined signal references when they were used in multiple entities. Conformal only supports one entity that uses the externally defined signal.

Example

In the following example, the design references the externally defined signal `glob1` in multiple entities, which causes Conformal to error out.

Note: Ellipses (…) denote characters that are present in the file, but not shown in this example.

```
PACKAGE pack IS
  SIGNAL glob1 : boolean;
END PACK;
USE work. pack.all;
ENTITY test IS
…
END test;
ARCHITECTURE arch OF test IS
…
  glob1 <= in0 or in1;
END arch;
ENTITY test2 IS
…
END test2;
ARCHITECTURE arch OF test2 IS
…
  glob1 <= in0 or in1;
END arch;
```

RTL2.3a

Message

```
Signal declared in package is used in LHS
```

Default Severity

Warning

Description

Conformal did not support externally defined signal references.

When a signal declared in package and used in architecture, the signal would be localized.

If the signal was used in LHS, the rule violation would be issued.

Example

In the following example, the design references the externally defined signal glob1 in line 10, and it is in the LHS:

Note: Ellipses (...) denote characters that are present in the file, but not shown in this example.

```
1 PACKAGE pack IS
2   SIGNAL glob1 : boolean;
3 END PACK;
4 USE work. pack.all;
5 ENTITY test IS
6 ...
7 END test;
8 ARCHITECTURE arch OF test IS
9 ...
10 glob1 <= in0 or in1;
11 END arch;
```

RTL2.4

Message

`Externally defined signal reference is supported`

Default Severity

Note

Description

Conformal supports one or more externally defined signal references. Conformal only supports the first entity that uses the externally defined signal.

Example

In the following example, the design uses signal `glob1` although it is defined outside entity `TEST` (see line 13).

```
1  package pack is
2    signal glob1 : boolean;
3  end pack;
4  use work. pack.all;
5  entity TEST is
6    port (
7      in0, in1, in2 : in boolean;
8      out0 : out boolean
9    );
10 end test;
11 architecture arch of test is
12 begin
13   glob1 <= in0 or in1;
14   out0 <= glob1 and in2;
15 end arch;
```

RTL2.5

Message

```
Net is referenced without an assignment. Design verification will be based on
     set_undriven_signal setting.
```

Default Severity

Warning

Description

The specified wire is not assigned. It is connected to an object indicating that the wire has been read at least once. Such usage leads to redundant code, undefined states, and errors. To avoid this, assign wires properly. If unassigned wires are not used, remove them. This rule honors the synthesis off or on pragmas. This implies that a wire is considered unassigned when assignment is made inside synthesis off or on pragma and its declaration is outside the pragma.

Example

In the following example, `g1` (line 5) is an undriven net:

```
1 module unassigned_wire (in1, out1);
2   input in1;
3   output out1;
4   wire g1;
5   assign  out1 = in1 & g1;
6 endmodule
```

RTL2.5a

Message

`Self-driven net converted to undriven net`

Default Severity

Warning

Description

Self-driven net(s) with no other driver is converted into undriven net(s).

Example

In the example below, the assignment on line 7 assigns the nets tmp[4:0] to itself.

```
1 module test(a, b, sel, z);
2   input a, sel;
3   input [5:0] b;
4   output [5:0] z;
5
6   wire [5:0] tmp;
7   assign tmp = {a, tmp[4:0];
8   assign z = sel? tmp : b;
9 endmodule
```

RTL2.5b

Message

`Self-driven inout port converted to undriven port`

Default Severity

Warning

Description

Self-driven inout port(s) with no other driver is converted into undriven port(s).

Example

In the example below, in the assignment on line 4, inout port '`z[4:0]`' is assigned to itself. Conformal will model bit 4 to bit 0 of port 'z' as undriven and reports RTL2.5b rule violation.

```
 1  module test (in, z);
 2     input in;
 3     inout [5:0] z;
 4     assign z = { in, z[4:0] };
 5  endmodule
```

RTL2.6

Message

`Shared variables are not supported`

Default Severity

Warning

Description

Shared variables are not supported. To avoid this error, remodel the design.

Example

In the following example, `shared variable counter` (line 8) is not supported:

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  entity shared_variables is
4    port (DataIn : in integer);
5  end entity shared_variables;
6  architecture shared_variables_arch of shared_variables is
7    subtype ShortRange is integer range 0 to 1;
8    shared variable counter : ShortRange := 0;
9  begin
10 PROC1:process
11   begin
12     counter := counter + 1;
13     wait;
14   end process PROC1;
15 PROC2:process
16   begin
17     counter := counter - 1;
18     wait;
19 end process PROC2;
20 end architecture shared_variables_arch;
```

RTL2.7

Message

```
Dynamic slicing has been detected. Such expressions are not supported and are
     ignored.
```

Default Severity

Warning

Description

The left and right bounds of the part range cannot be evaluated to constant values. Such expressions are not supported by synthesis tools and are ignored.

Example

In the following example, the dynamic slice in line 11 is not supported:

```
1  Library IEEE;
2  use IEEE.std_logic_1164.all;
3  Entity TEST is
4  Port (
5      index: in integer range 3 downto 0;
6      data_out : out std_logic_vector (7 downto 0));
7  End TEST;
8  Architecture ARCH_TEST of TEST is
9  Signal data : std_logic_vector (7 downto 0);
10  Begin
11     data_out (index downto 0) <= data(index downto 0);
12  End ARCH_TEST;
```

RTL2.8

Message

`Signal types BUS and REGISTER are not supported for synthesis`

Default Severity

Warning

Description

The `BUS` and `REGISTER` signal type are not supported by the synthesis tools and are ignored.

Example

In the following example, the `bus` signal type in line 5 is not supported:

```
1  entity EG1 is
2      port (a: in BIT; z: out bit);
3  end;
4  architecture RTL of EG1 is
5  signal sig1: bit bus;
6  begin
7      process
8      begin
9        if a = '1' then z <= '1';
10       else  z<= '0';
11       end if;
12    end process;
13 end;
```

RTL2.9

Message

`Guarded assignment requires GUARD signal`

Default Severity

Error

Description

The `GUARD` signal is not declared for guarded assignment.

Example

In the following example, the `guarded` signal is not declared for line 9's guarded assignment:

```
1  entity EG1 is
2     port (a,b: in bit; z: out bit);
3  end;
4  architecture RTL of EG1 is
5  signal val: bit;
6  begin
7      guarded_block: block
8      begin
9        val <= guarded a and b;
10    end block;
11    z <= val;
12 end;
```

RTL2.10

Message

```
GUARD is not declared
```

Default Severity

Error

Description

The `GUARD` signal is not declared.

Example

In the following example, the `guard` signal in line 8 is not declared:

```
1 entity EG1 is
2     port (a,b: in bit; z: out bit);
3 end;
4 architecture RTL of EG1 is
5 begin
6     guarded_block: block --(a = '1')
7     begin
8       z <= '1' when guard else '0';
9     end block;
10end;
```

RTL2.11

Message

`Implicitly declared GUARD signal cannot be updated`

Default Severity

Error

Description

The implicitly declared `GUARD` signal cannot be updated. The implicit guard signal will be declared for a guarded block. This is illegal to update the implicitly declared `GUARD` signal.

Example

In the following example, the `guard` signal in in line 8 is illegal:

```
1  entity EG1 is
2      port (a,b: in bit; z: out bit);
3  end;
4  architecture RTL of EG1 is
5  begin
6      guarded_block: block (a = '1')
7      begin
8        guard <= (b= '1');
9        z <= '1' when guard else '0';
10     end block;
111 end;
```

RTL2.12

Message

```
Invalid redeclaration of GUARD signal
```

Default Severity

Error

Description

The redeclaration of `GUARD` signal is invalid. The implicit `GUARD` signal will be declared for a guarded block. It is illegal to redeclare `GUARD` signal.

Example

In the following example, the `guard` signal in in line 7 is invalid:

```
1  entity EG1 is
2      port (a,b: in bit; z: out bit);
3  end;
4  architecture RTL of EG1 is
5  begin
6      guarded_block: block (a = '1')
7      signal guard : boolean;
8      begin
9        z <= '1' when guard else '0';
10     end block;
11 end;
```

RTL2.13

Message

`Undriven pin is detected`

Default Severity

Warning

Description

Undriven pin(s) are detected in the design.

Example

In the following example, pin `in1` of instance `test` is undriven (see line 4):

```
1  module top (din1, dout1);
2  output dout1;
3  input din1;
4  test t1(.in2(din1),.out1(dout1));
5  endmodule
6  module test(in1,in2, out1);
7    input in1, in2;
8    output out1;
9    assign  out1 = in1 & in2;
10 endmodule
```

RTL2.13a

Message

```
Undriven pin is detected. Module is empty
```

Default Severity

Warning

Description

Undriven pin(s) are detected in the design in an empty module.

Note: By default, up to 5 occurrences of RTL2.13a violations will be reported. The command 'set_rule_handling' command '-MODULE_LIMIT <n>' option can be used to change the maximum number of occurrences reported.

Example

In the following example, module 'sub' is empty. Hence output 'out1' is undriven and will be reported as RTL2.13a violation.

```
module sub (input in1, input in2, output out1);
`ifdef CONNECT
assign out1 = in2;
`endif
endmodule

module test (input in1, input in2, output out1);
sub sub(.*);
endmodule
```

RTL2.14

Message

```
Package variable is referenced. Possible simulation mismatch
```

Default Severity

Warning

Description

In SystemVerilog, packages provide a declaration space, which can be shared by other building blocks. Referencing a non-constant variable of a package can lead to a simulation and synthesis mismatch.

Example

In line 2 of the following SystemVerilog example, the logic variable `r` is shared by all modules. In the simulation, `r` is changed when input `a` or `b` is changed. Therefore, the output ports `c` and `d` of module `top` have the same identical value. In synthesis, the reference to the package variable is implementation dependent, and could cause the output ports '`c`' and '`d`' to have different values.

```
1  package pkg;
2      logic r;
3      task write(input logic x);
4          r = x;
5      endtask
6  endpackage
7
8  module top(input a,b, output c,d);
9      sub1    u1(.x(a),.z(c));
10      sub1    u2(.x(b),.z(d));
11  endmodule
12
13  module sub1(input x, output z);
14      always @*
15          pkg::write(x);
16      assign z = pkg::r;
17  endmodule
18
```

RTL3.1

Message

`Variable/signal is unassigned in asynchronous set/reset`

Default Severity

Warning

Description

The design includes one or more variables or signals that are unassigned in one of the conditional branches in asynchronous set/reset branches.

Example

In the following example, the design assigns output `out0` in the `else` branch, but not in the `if` branch of the asynchronous set/reset. See line 11.

```
1  module SEN (clk,rst,in0,out0,out1);
2  input clk,rst,in0;
3  output out0,out1;
4  reg    out0,out1;
5    always @ ( posedge rst or posedge clk )
6      begin
7        if (rst) begin
8          out1 <= 1'b0;
9        end
10       else begin
11         out0 <= in0;
12         out1 <= !in0;
13       end
14     end
15 endmodule
```

RTL3.2

Message

`Assignment of X is in asynchronous set/reset branch`

Default Severity

Warning

Description

The design includes one or more X value assignments in asynchronous set/reset branches.

Example

In the following example, the design assigns `X` value to `out0`. See line 8.

```
1  module OPW (clk,rst,in0,out0);
2  input clk,rst,in0;
3  output out0;
4  reg    out0;
5    always @ ( posedge rst or posedge clk )
6      begin
7        if (rst) begin
8          out0 <= 1'bx;
9        end
10       else begin
11         out0 <= in0;
12       end
13     end
14 endmodule
```

RTL3.3

Message

`Non-constant value assignment is in asynchronous set/reset value`

Default Severity

Warning

Description

The design includes one or more non-constant value assignments in asynchronous set/reset branches.

Example

In the following example, the design assigns a non-constant value to `out0`. See line 8.

```
1  module OPW (clk,rst,in0,in1,out0);
2  input clk,rst,in0,in1;
3  output out0;
4  reg    out0;
5    always @ ( posedge rst or posedge clk )
6      begin
7        if (rst) begin
8          out0 <= in1;
9        end
10       else begin
11         out0 <= in0;
12       end
13    end
14 endmodule
```

RTL3.4

Message

```
DFF/DLAT is with both asynchronous set and reset connections
```

Default Severity

Warning

Description

The design includes one or more DFFs or DLATs that have both asynchronous set and reset connections.

Example

On line 8 of the following example, the design uses `rst` for an asynchronous reset. The design then uses `set` for an asynchronous set on line 10.

```
1  module test (clk,set,rst,in0,in1,out0);
2  input clk,set,rst,in0,in1;
3  output out0;
4  reg    out0;
5    always @ ( posedge rst or posedge set or
6      posedge clk )
7      begin
8        if (rst)
9          out0 <= 1'b0;
10       else if (set)
11         out0 <= 1'b1;
12       else
13         out0 <= in0;
14     end
15 endmodule
```

RTL3.5

Message

```
Variable/signal is assigned without using asynchronous set/reset
```

Default Severity

Ignore

Description

Design style check to report if the RTL codes infer to a DFF gate that does not use any asynchronous set or reset signal.

Example

On line 6 of the following example, the design assigns output `out0` without any asynchronous set or reset signal:

```
1 module SEN (clk,rst,in0,out0);
2 input  clk,rst,in0;
3 output out0;
4 reg    out0;
5 always @(posedge clk)
6   out0 <= in0;
7 ...
8 endmodule
```

RTL4.1

Message

```
Enum encoding is applied to enum type
```

Default Severity

Note

Description

The design includes one or more `ENUM_ENCODING` attributes to declare enum type in VHDL. For Verilog, use the `synthesis enum` directive to declare enum type.

Example

On line 6 of the following example, the enum encoding is applied to enum type of `bufferA`:

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.all;
3
4  ENTITY test IS
5  PORT (
6      bufferA : BUFFER std_ulogic);
7  END test;
```

RTL4.2

Message

```
Multiple wait statements. FSM encoding might be different
```

Default Severity

Error

Description

The design includes one or more always/process blocks containing multiple wait statements. Conformal creates a Finite State Machine (FSM) when it encounters multiple wait statements. We recommend that you verify that the FSM encoding is what you expected.

Example

On lines 8 and 12 of the following example, the design uses the wait statement:

```
1  module Medge (Clock, Cond, In1, In2, In3, Out1);
2  input Clock, Cond, In1, In2, In3;
3  output Out1;
4  reg Out1;
5
6  always
7    begin
8      @ (posedge Clock);
9      Out1 = In1;
10     if (Cond)
11       begin
12         @ (posedge Clock);
13         Out1 = In2;
14       end
15     else
16       begin
17         @ (posedge Clock);
18        Out1 = In3;
19     end
20   end
21
22 endmodule
```

RTL4.2a

Message

```
Multiple wait statements in library module. FSM encoding might be different
```

Default severity

Warning

Description

The library cell includes one or more always/process blocks containing multiple wait statements. Conformal creates a Finite State Machine (FSM) when it encounters multiple wait statements. We recommend to black-box the library cell or verify if FSM encoding is as expected.

Example

The always block in the example below has multiple edge-sensitive wait statements.

```
module test(input clk, input [2:0] en, input [3:0] d, output reg [3:0] q);
    always
    begin @ ( posedge clk )
      while ( en == 3'b000 )
      begin @ ( posedge clk);
        q = d;
      end
    end
endmodule
```

RTL4.3

Message

```
Enum value size is different than the declared data type
```

Default Severity

Warning

Description

The `enum` value size is different than the declared data type.

Example

The declared type 'light' size is 2, but the `enum` value GREEN size is 3:

```
module enum10(input clk,output reg out1);
typedef enum reg signed [1:0] { RED = 2'sb10,YELLOW,GREEN = 3'b000} light;
light mylight;
always @(clk)
begin
  mylight=RED;
  out1 = mylight;
end
endmodule
```

RTL4.4

Message

`Encoding format has too many values`

Default Severity

Warning

Description

There are more encoding values than necessary.

Example

In line 4, there are more encoding values than necessary:

```
1  package test is
2      attribute ENUM_ENCODING : string;
3      type fifo_valid_state_type is (full1, full2, full3, empty);
4     attribute ENUM_ENCODING of fifo_valid_state_type : type is "0001 0011 0010
0000 0100 0101 0111 0110 1110";
5  end test;
```

RTL4.5

Message

```
Encoding format has duplicate values
```

Default Severity

Warning

Description

The encoding of an enumeration has duplicate values.

Note: This rule is not triggered if you are reading in a system package.

Example

```
1 type COLOR is (RED, GREEN,  YELLOW);
2   attribute ENUM_ENCODING : string;
3   attribute ENUM_ENCODING of COLOR : TYPE is "01 11 11 "  <== line3
```

RTL4.6

Message

```
Enum value specification has error
```

Default Severity

Error

Description

An enum value specification in the enum definition has error.

Example

In the enum definition below on line-6, the enum literal ’S3’ is assigned the value of 2’b10. The value 2’b10 has been assigned previously to enum literal ’S2’. Conformal will report RTL4.6.

```
1 package mpkg;
2   typedef enum [1:0]
3     S0 = 0
4     S1 = S0 + 2’b1,
5     S2 = S1 + 2’b1,
6     S3 = S1 + 2’b1 // RTL4.6
7   } states_e;
8 endpackage
```

In the enum definition below on line-6, enum literal ’S4’ has the value of ’4’ which is outside the representable range of the enum base-type ’reg [1:0]’. Conformal will report RTL4.6.

```
1 package mpkg;
2   typedef enum [1:0] {
3     S0 = 0, S1 = 1, S2 =3,
4     S4 = 4 // RTL4.6
5   } states_e;
6 endpackage
```

RTL4.7

Message

```
Enum base-type defaults to 32-bit integer
```

Default Severity

Warning

Description

The base-type of an enum is not specified and defaults to 32-bit integer.

Example

In the enum definition below, the base-type of the enum ’states_e’ is not specified. Conformal will set the base-type to the default type which is 32-bit integer and report RTL4.7.

```
package mpkg;
  typedef enum {
    S0, S1, S2, S3
  } states_e;
endpackage
```

RTL5.1

Message

`Overlapped case items are in parallel case statement`

Default Severity

Warning

Description

The design includes multiple `case_items` that potentially match the `case_expressions` in parallel case statements.

A parallel case statement syntax:

```
case (case_expression) //synthesis Parallel_case case_item1 :
//case_item_statement1;
case_item2 : case_item_statement2;
default    : case_item_statement3;
endcase
```

In simulation, only the first matching case item is considered. In synthesis and also in Conformal, all matching case items are considered and the matching outputs are OR-ed together.

Example

On lines 8 and 9 of the following example, case_items `2'b1?` and `2'b?1` match the case_expression `sel` when `sel` is equal to `2'b11`.

```
1 module test ( a, b, sel, out0);
2 input  a, b;
3 input  [1:0] sel;
4 output out0;
5 reg out0;
6   always @(sel or a or b) begin
7     casez (sel)  //synthesis parallel_case
8       2'b1?  : out0 = a;
9       2'b?1  : out0 = b;
10       default: out0 = 1'bx;
11     endcase
12   end
13 endmodule
```

RTL5.1a

Message

`Overlapped case items`

Default Severity

Warning

Description

Indicates that the design includes multiple `case_items` that match the `case_expressions` in case statements.

Example

In the following example, case item 2'b01 is overlapped on line 8 and 11.

```
1 module top (clk, sel, oo);
2 input clk;
3 input [1:0] sel;
4 output reg [1:0] oo;
5 always @(posedge clk)
6   case (sel)
7     2'b00: oo <= 2'b11;
8     2'b01: oo <= 2'b10;
9     2'b10: oo <= 2'b01;
10    2'b11: oo <= 2'b00;
11    2'b01: oo <= 2'b10;
12  endcase
13 endmodule
```

RTL5.1b

Message

`Parallel case statement may have overlapping case items`

Default Severity

Warning

Description

The design includes parallel case statement with case-item expressions where Conformal does not perform or fails to complete overlapping case-items check.

In simulation, only the first matching case item is considered. In synthesis and also in Conformal, all matching case items are considered and the matching outputs are OR-ed together.

Example

In the example below, the case-items on line 4 and line 5 contains complex, non-constant boolean expressions. Conformal does not perform analysis on such expressions due to their potential prove complexity.

```
1  module test(input [7:0] in, input a, b, c, output reg [7:0] out);
2        always_comb begin
3            case (1'b1) // synthesis full_case parallel_case
4               ((a  ^ b) & ~c) :  out = in;
5               ((a ~^ b) & ~c) :  out = ~in;
6               default : out = '0;
7            endcase
8        end
9  endmodule
```

RTL5.1c

Message

Case items have nonconsequential overlapping condition

Default Severity

Note

Description

Indicates that the design includes multiple `case_items` that match the `case_expressions` in case statements. However, the overlapping conditions do not effect the output(s).

Example

In the following example, case item 2'b01 are overlapping on line 8 and 11, but the output of the case statement are not affected since the rhs are the same.

```
1 module top (clk, sel, oo);
2 input clk;
3 input [1:0] sel;
4 output reg [1:0] oo;
5 always @(posedge clk)
6   case (sel)
7     2'b00: oo <= 2'b11;
8     2'b01: oo <= 2'b10;
9     2'b10: oo <= 2'b01;
10    2'b11: oo <= 2'b10;
11    2'b01: oo <= 2'b10;
12  endcase
13 endmodule
```

RTL5.1d

Message

`Unique case statement may have overlapping case items`

Default Severity

Warning

Description

The design includes a unique case statement with case-item expressions where Conformal does not perform or fails to complete overlapping case-items check.

In simulation, only the first matching case item is considered. In synthesis and also in Conformal, all matching case items are considered and the matching outputs are OR-ed together.

Example

In the example below, the case-items on line 4 and line 5 contains complex, non-constant boolean expressions. Conformal does not perform analysis on such expressions due to their potential prove complexity.

```
1  module test(input [7:0] in, input a, b, c, output reg [7:0] out);
2        always_comb begin
3            unique case (1'b1)
4               ((a  ^ b) & ~c) :  out = in;
5               ((a ~^ b) & ~c) :  out = ~in;
6               default : out = '0;
7            endcase
8        end
9  endmodule
```

RTL5.2

Message

```
Non-binary case items are in case statement
```

Default Severity

Warning

Description

The design includes one or more case_items with non-binary values in case statements.

A case statement syntax:

```
    case  (case_expression)
    case_item1 :  case_item_statement1;
    case_item2 :  case_item_statement2;
    default    :  case_item_statement3;
endcase
```

Example

In the following example, the design includes a `Z` value for a case_item. See line 9.

```
module test ( a, b, c, sel, out0);
input  a, b, c;
input  sel;
output out0;
reg out0;
  always @(sel or a or b or c) begin
    case(sel)
      1'b1   : out0 = a;
      1'bz   : out0 = b;
      default: out0 = c;
    endcase
  end
endmodule
```

RTL5.3

Message

`Case expressions/items are resized`

Default Severity

Warning

Description

The design includes one or more case item sizes that do not match the case expression sizes.

Example

On lines 8 and 10 of the following example, the case items have a single bit, while the case expression (line 3) is a 2-bit vector.

```
module test ( a, b, c, sel, out0);
input  a, b, c;
input  [1:0] sel;
output out0;
reg out0;
  always @(sel or a or b or c) begin
    case(sel)
      1'b1 :
        out0 = a;
      1'b0 :
        out0 = b;
      default:
        out0 = c;
    endcase
  end
endmodule
```

RTL5.4

Message

`Partial case items are in full case statement`

Default Severity

Note

Description

The design includes a partial case enumeration in a full case statement.

Example

The following example contains a potential partial case enumeration. The `2'b00` and `2'11` case items are not enumerated in the `full_case` statement.

```
module test (out, a, b, select);
output out;
input a, b;

input [1:0] select;
reg out;

always @(select or a or b) begin
   case (select) /* synthesis full_case */
         2'b01: out = a;
         2'b10: out = b;
   endcase
end
endmodule
```

RTL5.5a

Message

Default case item with non-X assignment(s)

Default Severity

Note

Description

The default case item has assignment with non-X value(s).

Example

In the following example, assignment ’oo’ is assigned a binary, non-X value in the default clause. Note: this rule can be used to check for coding consistency.

```
module top (clk, sel, oo);
input clk;
input [1:0] sel;
output reg [1:0] oo;
always @(posedge clk)
  case (sel)
    2'b00: oo <= 2'b11;
    2'b01: oo <= 2'b10;
    2'b10: oo <= 2'b01;
    default: oo <= 2'b00;
endcase
endmodule
```

RTL5.6

Message

Case statement with no default

Default Severity

Note

Description

Indicates that the case statement has no default clause

Example

In the following example, the case statement has no default clause. Note: this rule can be used to check for coding consistency.

```
module top (clk, sel, oo);
input clk;
input [1:0] sel;
output reg [1:0] oo;
always @(posedge clk)
  case (sel)
    2'b00: oo <= 2'b11;
    2'b01: oo <= 2'b10;
    2'b10: oo <= 2'b01;
    2'b11: oo <= 2'b10;
 endcase
 endmodule
```

RTL5.7

Message

Default case item with inconsistent assignment(s).

Default Severity

Note

Description

Indicates that there is inconsistent assignment(s) in the default clause of a case statement.

Example

In the following example, variable ’oo’ is not assigned in the default clause.

```
module top (clk, sel, oo, oo2);
input clk;
input [1:0] sel;
input reg [1:0] oo, oo2;
always @(posedge clk)
  case (sel)
    2'b00: oo <= 2'b11;
    2'b01: oo <= 2'b10;
    2'b10: oo <= 2'b01;
    default: oo2 <= 2'b10;
  endcase
endmodule
```

RTL6.1

Message

`X created due to the assignment of value X`

Default Severity

Warning

Description

One or more assignments use value X. Conformal generates a *don't care* for any value X assignment.

Example

In the following example, the design assigns a `1'bx` value to output `o`. See line 6.

```
module test(o,i);
output o;
reg o;
input i;
always
o = 1'bx;
endmodule
```

RTL6.1a

Message

X created due to default case item with X assignment(s)

Default Severity

Warning

Description

One or more assignments use value X in the default clause of a case statement. Conformal generates a *don't care* for any value X assignment.

Example

In the following example, the ’x’ assignment happens in the default clause of a case statement.

```
module top (clk, sel, oo, oo2);
input clk;
input [1:0] sel;
input reg [1:0] oo, oo2;
always @(posedge clk)
  case (sel)
    2'b00: oo <= 2'b11;
    2'b01: oo <= 2'b10;
    2'b10: oo <= 2'b01;
    default: oo2 <= 2'bxx;
  endcase
endmodule
```

RTL6.2

Message

```
Integer value range constraint is added
```

Default Severity

Warning

Description

When you read in the design, you specified a constraint on integers for a specific range. The switch of the command that you used to apply the constraint is `-rangeconstraint`.

Example

In the following example, integer `int1` has a value range from -1024 to 1024 (see line 5).

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.all;
3  ENTITY test IS
4  PORT (
5  int1 : IN integer range -1024 to 1024 ;
6  bufferA : BUFFER bit
7  );
8  END test;
9  ARCHITECTURE rtl OF test IS
10  BEGIN
11      PROCESS
12      BEGIN
13          IF int1 = 4095 THEN
14              bufferA <= '0';
15          ELSE
16              bufferA <= '1';
17          END IF;
18      END PROCESS;
19  END RTL;
```

RTL6.3

Message

`X created when divisor equals to zero`

Default Severity

Warning

Description

The design contains one or more divisors that are signals or variables, and not constants. Thus, there is a chance that they will equal zero. In these cases, Conformal creates `X`.

Example

In the following example, the divisor `B` is an input signal, thus Conformal creates `X` (see line 4).

```
1 module test(A, B, QUOTIENT);
2 input [7:0] A, B;
3 output [7:0] QUOTIENT;
4 assign QUOTIENT = A / B;
5 endmodule
```

RTL6.3a

Message

```
Dividend size less than divider size; extra leading bit(s) set to X when divisor
is zero
```

Default Severity

Warning

Description

The design contains a divide operator where the divident size is less than the divisor size.

Example

In the example below, on line 3, the size of the divider (the divide expression) is the larger of the dividend (4 bits) and the divisor (5 bits), which is 5 bits. Bit 5 of the divider output is redundant. Conformal will set the value the extra bit to 'X' when the divisor is '0', or '0'(s) otherwise. Conformal will report RTL6.3a.

```
1  module test(input [4:0] div, output [4:0] out);
2      assign out = 4'b0100 / div;
3  endmodule
```

RTL6.4

Message

`Enum value constraint is added`

Default Severity

Warning

Description

Conformal recognized the `synthesis enum fsm_state` directive, thus it added a constraint to Enum value.

Example

In the following example, in lines 3 through 9, Conformal constrains enum states `ra_state` and `next_state` to a one-hot condition. Need to specify `-enumconstraint` option of `read design` command.

```
1  module test (masterclk, reset_n, mem_write);
2  input masterclk, reset_n, mem_write;
3  parameter [4:0] // synthesis enum fsm_states
4  iddle = 5'b00001,
5  step01 = 5'b00010,
6  step02 = 5'b00100,
7  step03 = 5'b01000,
8  step04 = 5'b10000;
9  reg [4:0] /* synthesis enum fsm_states */ ra_state, next_state;
10  always @(ra_state or mem_write) begin
11  case (ra_state)
12  iddle: begin
13  next_state = step01;
14  end
15  //...
16  default: begin
17  next_state = iddle;
18  end
19  endcase
20  end
21  endmodule
```

RTL6.4a

Message

`Enum value might need constraint.`

Default Severity

Warning

Description

Conformal found that the `enum` data type does not cover all the bits that represent the data type, and a constraint might need to be added by using `-enumconstraint` in `READ DESIGN` to constrain the `enum` value.

Example

```
package data_types is
   type my_enum is (ONE, TWO, THREE);
end data_types;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
USE WORK.data_types.ALL;

entity test is
port(
 sig3 : my_enum;
 z : out std_logic_vector(1 downto 0)
);
end entity;

ARCHITECTURE test OF test IS

function to_slv(rec: my_enum) return std_Logic_vector is
   variable slv : std_logic_vector(1 downto 0);
begin
   slv := conv_std_logic_vector(my_enum'POS(rec), slv'length);
return slv;
end function;

BEGIN
   z <= to_slv(sig3);
END test;
```

RTL6.5

Message

```
priority if and unique if statements are incomplete and can lead to simulation
     mismatch
```

Default Severity

Warning

Description

There are incomplete `priority if` and `unique if` SystemVerilog statements. The `priority if` and `unique if` SystemVerilog statements ensure that all conditions are complete; they ensure that default `else` statements exist and `if/else` if conditions cover all conditions.

Example

In the following example, the incomplete `priority if` statement in line 4 triggers this rule check, while the complete `priority if` statement in lines 11-12 do not trigger this rule check.

```
module test1(input aa, output oo);
     reg oo;
     always @* begin
          priority if (aa) oo = !aa; // incomplete condition
     end
     endmodule

     module test2(input aa, output oo);
     reg oo;
     always @* begin
          priority if (aa) oo = !aa;
          else             oo = aa; // complete condition
     end
     endmodule
```

RTL6.6

Message

`unique if statements may have redundant or overlapping conditions`

Default Severity

Note

Description

There are possible redundant or overlapping conditions in `unique if` statements. Structural checks generate this message; a more detailed analysis is done during compare.

Example

In the following example, there are overlapping conditions in the `unique if` statement that will cause a rule check report. See line 5.

```
module test3(input aa, bb, output oo);
     reg oo;
     always @* begin
       unique if (aa && bb) oo = !aa;
       else if   (aa || bb)        oo = aa;
       else                        oo = 1'b0;
     end
     endmodule
```

RTL6.7

Message

`Value X in operand(s) of an arithmetic expression`

Default Severity

Warning

Description

The operand(s) of an arithmatic expression contains the value 'x'. Where appropriate, partial 'x' value will be fully extended to all 'x'.

Example

In the following example, on line 15, when 'sel' is 2'b00 or 2'b11, the second operand '{4'b0000, y}' of the arithmatic operator '+' will have the value of '{4'b0000,6'bxxxx}'.

```
1 module test(sel,a,data1,data2,Z);
2 input [1:0] sel;
3 input [9:0]  a;
4 input [5:0]  data1,data2;
5 output reg [9:0] Z;
6   always @*
7   begin
8       reg [5:0] y;
9       case(sel)
10          2'b01: y = data1;
11          2'b10: y = data2;
12          default: y =  6'bx;
13      endcase
14      Z = a + {4'b0000,y};
15  end
16 endmodule
```

Conformal will issue rtl rule check RTL7.6a and transform the partial 'x' value into all 'x', similar to the assignment on line 9 below:

```
7   always @*
8   begin
9       Z = a + ((sel==2'b01)? {4'b0000,data1} :
10               (sel==2'b10)? {4'b0000,data2} :
11               10'bx);
12  end
```

RTL6.8

Message

```
Value X or Z in the right operand of a shift expression
```

Default Severity

Warning

Description

The right operand of a shift expression contains value 'X' or 'Z'. The expression will evaluate to all Xs

Example

In the following example, on line 4, the right operand of the shift operator contains 'X'. The expression will evaluate to 2'bxx.

```
1 module top1(top1_o0);
2     output wire[1:0] top1_o0;
3
4     assign top1_o0 = (2'b11 << 1'bx); // <= RTL6.8
5 endmodule
```

RTL6.9

Message

```
Value Z in expression causes expression to evaluate to X
```

Default Severity

Warning

Description

The expression contains operands that has 'Z' values and result of the evaluation of the expression will be 'X' or contains 'X's.

Example

In the following example, on line 5, parameter 'BAR' on the rhs of the unary-or expression contains 'Z's. Result of the expression contains 'X's.

```
1 module test(input [1:0] sel, input [3:0] in, output reg [3:0] out);
2     parameter BAR = 4'b11zz;
3     always @(*)
4     begin
5         out = in | BAR; // RTL6.9
6     end
7 endmodule
```

Note: 'X's in the expression above most probably are interpreted as Don't-Cares, which will mask the 'Z's from the original expression.

RTL7.1

Message

```
Design includes comparison that uses X and Z values
```

Default Severity

Warning

Description

The design includes one or more comparisons that use X or Z values as comparators.

Example

In the following example, on line 7, the design compares reset `rst` with an `X` value. The comparison will be interpreted as 1’b0 or false and the ’else’ branch on line 10 is taken.

```
1 module test (clk,rst,in0,out0);
2 input clk,rst,in0;
3 output out0;
4 reg    out0;
5   always @ ( posedge clk )
6     begin
7       if (rst == 1'bx)
8         out0 <= 1'b0;
9       else
10        out0 <= in0;
11    end
12 endmodule
```

Note: Comparison with 'X' or 'Z' values can potentially cause mismatches between synthesis and simulation results. Synthesis has the freedom to interpret 'X' as don't care and ’Z’ as floating net and hence can choose the 'then' or the 'else' branch for optimal implementation. Simulation on the other hand follows a predefined set of rules to choose between the 'then' or the 'else' branch, and the choice is not necessarily the same as synthesis. For example, in case of logical equality (==) or inequality (!=) whenever there is any ambiguity because of the 'X' or 'Z' value, the 'else' branch will be choosen.

RTL7.2

Message

```
Gate or transistor primitive is using weak attributes
```

Default Severity

Warning

Description

One or more gates or transistor primitives are using weak attributes.

Note: Not all weak attributes will be used. In the case of a multiple-driven net, the gate with the strong attribute will drive the output.

Example

In the following example, the design uses a weak attribute for a buffer primitive. See line 5.

```
1 module test (clk,rst,in0,out0);
2 input clk,rst,in0;
3 output out0;
4 wire   out0;
5 buf (weak0, weak1) (out0, in0);
6 endmodule
```

RTL7.3

Message

`Array index in RHS might be out of range`

Default Severity

Warning

Description

The design includes an index where its right hand side might be out of range.

Example

In the following example, it uses an index where its right hand side might be out of range. See line 10.

```
1 module test (clk, idx, in0, out0);
2 input clk;
3 input [3:0] idx;
4 input [3:0] in0;
5 output out0;
6 reg out0;
7
8  always @ ( posedge clk)
9    begin
10      out0 <= in0[idx];
11    end
12  endmodule
```

RTL7.4

Message

```
Array index in LHS might be out of range
```

Default Severity

Warning

Description

The design includes an index where its left hand side might be out of range.

Example

In the following example, it uses an index where its left hand side might be out of range. See line 10.

```
1 module test (clk, idx, in0, out0);
2 input clk;
3 input [3:0] idx;
4 input in0;
5 output [3:0] out0;
6 reg [3:0] out0;
7
8   always @ ( posedge clk)
9     begin
10        out0[idx] <= in0;
11      end
12    endmodule
```

RTL7.5

Message

`Input signal is assigned by logic values`

Default Severity

Warning

Description

The design includes one or more primary input signals that are assigned by logic values.

Example

In the following example, the design assigns logic value `1` to primary input `a`. See line 4.

```
1 module test(a, b);
2  input a;
3  output b;
4  assign a = 1'b1;
5  assign b = 1'b1;
6 endmodule
```

RTL7.5a

Message

```
Input signal is assigned by port directly
```

Default Severity

Warning

Description

The design includes one or more primary input signals that are directly assigned by another port.

Example

In the following example, the design assigns a port to primary input a.

See line 4.

```
1 module test(a, b);
2  input a;
3  output b;
4  assign a = b;
5 endmodule
```

RTL7.6

Message

`Value X or Z is treated as 0 in conditional or select expression`

Default Severity

Warning

Description

Conformal treats the `1'bx` or `1'bz` in a conditional or select expression as `1'b0`.

Example

In the following example, on line 4, the `4'bz` value in the conditional expression is treated as `1'b0`.

```
1  module test(input [4:0] in, output reg [1:0] out);
2      always @*
3      begin
4          if (in < 4'hz)
5              out = 2'b00;
6          else
7              out = 2'b11;
8      end
9  endmodule
```

RTL7.6a

Message

`Value X or Z is used as ternary branch condition, selection, or range expression`

Default Severity

Error

Description

Reported when value x/z is used as a condition in a ternary expression.

Example

Conformal reports RTL7.6a with the following RTL design.

```
module test(a, b, o);
  input a, b;
  output o;
  wire cond;
  assign cond = 1'bx;
  assign o = cond ? a : b;
endmodule
```

Even synthesizers will optimize the design to

```
assign o = b;
```

but the design intent is unclear.

RTL7.7

Message

`Real number rounded to integer value`

Default Severity

Error

Description

Conformal rounds real numbers to integer values.

Note: The severity of this rule will change to error starting in the 23.10-p100 release.

Example

In the following example, on line 2, input `din` has a real value of `2.2`. However, Conformal truncates this value at `.2`.

```
1 module test (din, dout);
2   input [2.2:0] din;
3   output [2:0] dout;
4
5   assign dout = din;
6 endmodule
```

RTL7.8

Message

`Overflowed integer is truncated`

Default Severity

Warning

Description

The design contains one or more integers that overflow. Thus, Conformal truncates them.

Example

In the following example, Conformal truncates `integer b`;. See line 3.

```
1 module test (b);
2 output b;
3 integer b;
4
5 always
6
7    b = 5147483648;
8 endmodule
```

RTL7.8a

Message

```
Overflowed sized integer number is truncated with bit value(s) lost
```

Default Severity

Warning

Description

Literal of a sized integer number has more bits than the specified size. Trunctating the number caused bit value(s) to be lost.

Example

In the example below, on line 5, integer literal 4'b10010 is truncated to 4'b0010, resulting in lost of significant bit value of ’1’.

```
1  module top(addr, out);
2     input [4:0] addr;
3     output out;
4     wire addr_dec;
5     assign out = addr[4:0]==4'b10010;
6  endmodule
```

RTL7.8b

Message

`Overflowed sized integer number is truncated with no bit value(s) lost`

Default Severity

Warning

Description

Literal of a sized integer number has more bits than the specified size. Trunctating the number does not cause any bit value(s) lost.

Example

In the example below, on line 5, integer literal 4'b00010 is truncated to 4'b0010. No value is lost from the truncating the significant bit value of ’0’.

```
1  module top(addr, out);
2     input [4:0] addr;
3     output out;
4     wire addr_dec;
5     assign out = addr[4:0]==4'b00010;
6  endmodule
```

RTL7.9

Message

`Sign overflowed integer is truncated`

Default Severity

Warning

Description

The design contains one or more integers that overflow at the sign bit. Thus, Conformal truncates them.

Example

In the following example, Conformal truncates `integer b`;. See line 3.

```
1 module test (b);
2 output b;
3 integer b;
4
5 always
6  b = 2147483648;
7
8endmodule
```

RTL7.9a

Message

`Signed integer overflowed`

Default Severity

Error

Description

The range of a signed integer type should be within the bounds `-2147483647` and `+2147483647` (inclusive). This message indicates that a signed integer has overflowed (gone beyond the bounds).

Example

In the following example, the result of `2 ** 31` is `2147483648` overflows. See line 4.

```
1 entity top is
2 end top;
3 architecture rtl of top is
4 constant n: integer := 2 ** 31;
5 begin
6 end rtl;
```

RTL7.10

Message

`Comparison with signed and unsigned operands`

Default Severity

Warning

Description

The design includes both signed and unsigned operands used in a relational expression.

Example

In the following example, Conformal compares the signed operand `in1` (on line 4) to the unsigned operand `in2` (on line 5). Conformal converts `in1` to an unsigned operand first, and then compares it with `in2`. An unsigned comparison results.

```
1 module test (out0, in1, in2);
2
3 output out0;
4 input signed [31:0] in1;
5 input unsigned [31:0] in2;
6
7   assign out0 = in1 > in2;
8
9 endmodule
```

RTL7.10a

Message

`Comparison with different data sizes`

Default Severity

Warning

Description

The left-hand-side (LHS) and right-hand-side (RHS) data sizes of the comparison are different.

Example

In the following example, Conformal compares the operand `in1` and operand `ABC` (line 5). The data sizes of operand `in1` and operand `ABC` are different.

```
1 module top(out0, in1);
2 output out0;
3 input [1:0] in1;
4 parameter ABC = 3'b001;
5 assign out0 = ( (in1 == ABC) ) ? 1'b1:1'b0;
6 endmodule
```

RTL7.10b

Message

`Ternary branches have different data sizes`

Default Severity

Warning

Description

The left-hand-side (LHS) and right-hand-side (RHS) data sizes of the ternary branches are different.

Example

In the following example, the operand in1 and operand in2 of ternary branch (on line 6) have different data sizes.

```
1 module top(out0, sel, in1, in2);
2 output out0;
3 input sel;
4 input in1;
5 input [1:0] in2;
6 assign out0 = ( (sel == 1'b1) ) ? in1:in2;
7 endmodule
```

RTL7.10c

Message

```
Bit-wise operation with different data sizes
```

Default Severity

Warning

Description

The left-hand-side (LHS) and right-hand-side (RHS) data sizes of the bit-wise operation are different.

Example

In the following example, Conformal compares the operand `in1` and operand `ABC` (line 5). The data sizes of operand `in1` and operand `ABC` are different.

```
1 module top(out0, in1);
2 output out0;
3 input [1:0] in1;
4 parameter ABC = 3'b001;
5 assign out0 = ( (in1 & ABC) ) ? 1'b1:1'b0;
6 endmodule
```

RTL7.10d

Message

```
Comparison with positive constant of different signedness
```

Default Severity

Warning

Description

Indicates that there is a comparison between a variable and a positive constant of different signedness.

Example

In the following example, the comparison on line 4 is between an unsigned variable 'in1' and a signed number '4' and will cause RTL7.10d to be issued.

```
1 module test (out0, in1);
2 output out0;
3 input unsigned [31:0] in1;
4 assign out0 = in1 > 4; // line 4
5 endmodule
```

RTL7.10e

Message

`Relational comparison of different data sizes and signed operand(s)`

Default Severity

Warning

Description

Indicates that the relational comparison involved operands of different sizes, and the smaller of two operands is signed.

Example

In the following example, the ternary expressions on line 6 and line 7 involves relational comparisons ('>=', '<') with different operands sizes, and with signed smaller operands. Conformal will report RTL7.10e in both cases.

```
1  module test (output out[1:0],
2               input [8:0] sig_a,
3               input signed [8:0] sig_c,
4               input signed [7:0] sig_d
5               );
6      assign out[0] = (sig_a >= sig_d) ? 1'b1 : 1'b0;
7      assign out[1] = (sig_c < sig_d) ? 1'b1 : 1'b0;
8  endmodule
```

RTL7.11

Message

`Implicit signed expression is converted to unsigned`

Default Severity

Warning

Description

The design includes a signed expression that Conformal has converted to an unsigned expression implicitly.

Example

In the following example, Conformal converts the signed operand `in1` to unsigned in the conditional expression. See line 7.

```
1 module test (out0, cond1, in1);
2
3 output unsigned [32:0] out0;3
4 input cond1;
5 input signed [31:0] in1;
6
7   assign out0 = cond1 ? out0 : in1;
8 endmodule
```

RTL7.11a

Message

`Implicit unsigned expression is converted to signed`

Default Severity

Warning

Description

A Verilog parameter is declared as signed, but the actual parameter value is an unsigned value.

Review the Verilog codes for potential misinterpretations in arithmetic operations.

Example

In line 2 of the following example, p1 is declared as a 4-bit signed parameter. However, in line 11, p1 is associated with a 4-bit unsigned value.

```
1 module sub(aa, oo);
2 parameter signed [3:0] p1 = -1;
3 input aa;
4 output oo;
5   assign oo = aa && (p1 < 0);
6 endmodule
7
8 module t1(aa, oo);
9 input aa;
10 output oo;
11   sub #(.p1(4'b1111)) u1(.aa, .oo);
12 endmodule
```

RTL7.11b

Message

```
Unsigned reference with index/part selection to a signed variable
```

Default Severity

Warning

Description

Indicates that a value becomes unsigned when using index/part selection to reference a signed variable or wire.

Example

In the following example, using index selection of signed wire "a" becomes an unsigned assignment, and no sign extension occurs.

See line 7 and 8.

```
1 module top ( b,z);
2   parameter WA = 1;
3   parameter WB = 4;
4   input signed [3:0]b;
5   output signed[5:0] z;
6   wire signed [WA - 1:0] [WB - 1:0] a;
7   assign a[0] = b;
8   assign z = a[0];
9 endmodule
```

RTL7.11c

Message

```
Increment or decrement operator in index/part-select expression
```

Default Severity

Note

Description

This rule check indicates that increment or decrement operator has been used within an index or part-select expression.

Example

In the example below, the indexes of the lhs expressions on line 9, line 10, line 11, and line12 contain post-increment expressions 'idx++'. Conformal will issue RTL7.11c on the post-increment expressions.

```
1  module test(input en, input [1:0] sidx, input [3:0] d, output reg [3:0] q);
2      always @(*)
3      begin
4         if (!en) begin
5             q = '0;
6         end else begin
7             reg [2:0] idx;
8             idx = sidx;
9             q[idx++ % 4] = d[0];
10             q[idx++ % 4] = d[1];
11             q[idx++ % 4] = d[2];
12             q[idx++ % 4] = d[3];
13          end
14       end
15  endmodule
```

RTL7.11d

Message

```
Struct or Array literals type in ternary expression is determined to be signed from
     the lhs context
```

Default Severity

Warning

Description

Indicate that a structure or array literal expression is in the branch of a ternary expression. The signed-ness of the structure or array literal is set to signed as determined from the signed-ness of the lhs context.

Example

In the example below on line 7, expression ''{default:0}' is signed since the lhs 'out' is signed. Conformal will report rule check message RTL7.11d.

```
1  module test(input clk, rst, sel, input logic signed [7:0] d,
2              output logic signed [15:0] out);
3      always @(*) begin
4          if (~rst)
5              out <= '{default:0};
6          else begin
7              out <= sel? d : '{default:0};
8          end
9      end
10  endmodule
```

RTL7.12

Message

```
Unsized integer number is truncated to 32 bits
```

Default Severity

Warning

Description

Conformal truncated an unsized integer literal to 32-bit.

Example

In the following example, Conformal truncates `'hf00000000` to `h00000000`. See line 2.

```
1 module test6(input aa, output oo);
2    assign oo = ('hf00000000 > 32'h0)?aa:!aa;
3 endmodule
```

Note; By default, Conformal truncates an unsized integer literal to 32-bit. The command "set hdl option -UNSIZED_CONSTANT_TRUNCATE off" can be used to instruct Conformal to not truncate the unsized integer literal to 32-bit

RTL7.12a

Message

```
Unsized integer number should be truncated to 32 bits, but the rule is turned off
```

Default Severity

Warning

Description

By default, Conformal truncates an unsized integer literal to 32-bit. When "set hdl option -UNSIZED_CONSTANT_TRUNCATE off" is specified, Conformal does not truncate the unsized integer literal to 32-bit.

Example

In the following example, Conformal by default truncates `'hf00000000` to `h00000000`. See line 2.

```
1 module test6(input aa, output oo);
2    assign oo = ('hf00000000 > 32'h0)?aa:!aa;
3 endmodule
```

```
Specify the hdl option "set hdl option -UNSIZED_CONSTANT_TRUNCATE off" to not
perform the truncation.
```

RTL7.13

Message

`Logical operator is applied to multiple-bit operand`

Default Severity

Warning

Description

The logical operator is applied to multiple-bit operand when the operand is not a single-bit expression.

Example

In the following example, the logical operator `&&` is applied to multiple-bit operand `tmp` (see line 5):

```
1 module test(in, out);
2 input in;
3 output out;
4 wire [2:0] tmp;
5 assign out = in && tmp;
6 endmodule
```

RTL7.14

Message

```
Loop exceeds maximum iterations
```

Default Severity

Warning

Description

An index variable should be sufficiently large enough to hold both the initial and the final values that the loop index could take. If the index variable is small, the loop will be executed fewer number of times. This can lead to unpredictable behavior.

Example

In the following example, 'I' can only take values from 0 to 3 as its size 2 bits. Line 8 of the code shows that the index of the for loop ranges from 0 to 5. Since the loop index variable 'I' is not large enough to hold the final value of the loop index, the loop may never terminate.

```
1 module neg_BITUSD_loop_idx_bitsel(a, b, c);
2   input [3:0] a, b;
3   output [3:0] c;
4   reg [3:0] c;
5   always @(a or b)
6   begin: P
7     reg [1:0] I;
8     for (I = 2; I <= 5; I = I + 1)
9     begin: loop2
10      c[I] = a[I] | b[I];
11    end
12    end
13 endmodule
```

RTL7.14a

Message

`Integer overflow in index expression`

Default Severity

Warning

Description

Index expression overflow in a for-loop can make the condition always true and leads to an infinite loop.

Example

In the following example, the index expression of the for-loop condition is `I <= 5`. However, variable '`I`' has only 2-bits, where the valid range is 0 to 3. Therefore, the condition `I <= 5` is always true and the for-loop becomes an infinite loop.

```
module test(a, b, c);
input [3:0] a, b;
output [3:0] c;
reg [3:0] c;
  always @(a or b)
    begin: P
    reg [1:0] I;
    for (I = 2; I <= 5; I = I + 1)
    begin: loop2
        c[I] = a[I] | b[I];
     end
   end
 endmodule
```

Conformal stops unfolding a loop when the loop exceeds the maximum number of iterations. By default, the maximum is 8192 iterations. To set a different maximum, use the following command:

```
SET HDl Options -MAX_FOR_LOOP_SIZE <integer_value>
```

RTL7.15

Message

```
Null slice is not supported (blackboxed)
```

Default Severity

Warning

Description

The specified datatype or node is defined with a null range, and it cannot hold any value. In VHDL, a range is considered to be a null range, if the subset of values specified by the range is empty (a decreasing range with a 'to' direction, or an increasing range with a 'downto' direction). Examples of null range are (3 to 0) and (1 downto 8). The synthesis tools do not support null range.

Example

In the following example, the null slice in line 9 is not supported:

```
1 entity TEST1 is
2          port (
3            in1: in integer ;
4            out2: buffer bit;
5            out3: out bit
6            ) ;
7 end TEST1 ;
8 architecture ARCH_TEST1 of TEST1 is
9 type myvector is array(0 downto 7) of BIT;
10 begin
11 process (in1)
12 variable vec: myvector;
13 variable index: integer;
14 variable temp: bit;
15 begin
16             vec:= ('1', '1', '1', '1', '1', '1', '1', '1') ;
17             temp:= vec(in1);
18 end process;
19 end ARCH_TEST1 ;
```

RTL7.15a

Message

```
Direction of VHDL slice differs from the prefix range
```

Default Severity

Error

Description

The VHDL object reference has a range direction that is different from its object declaration. This is an error in VHDL if a decreasing range was used with a `to` direction, or an increasing range was used with a `downto` direction.

Example

In the following example, at line 2, the `gnd_bus` is declared with decreasing direction, but it is used with increasing direction at line 9. This trigger the rule violation RTL7.15a.

```
1 entity top is
2   port (gnd_bus   : in bit_vector(3 downto 0);
3         comb_bus  : out bit_vector(3 downto 0)
4        ) ;
5 end top;
6
7 architecture rtl of top is
8 begin
9     comb_bus <= gnd_bus(0 to 3); -- ERROR: RTL7.15a
10
11 end rtl;
12
```

RTL7.16

Message

`Variable index is out of the defined range`

Default Severity

Error

Description

A bit or part select reference in an expression is found to have an index specification outside of the defined range of the variable. This can lead to unexpected simulation or synthesis results. Cadence recommends that you change the reference so that the index or subrange falls within the valid range.

Example

In the following example, index `[4]` is out of `a`'s range (see line 7):

```
1 module test1 (a, b, out1);
2 input [3:0] a;
3 input [3:0] b;
4 output out1;
5 reg out1;
6 always @(a or b)
7     out1 = a[4] & b[3];
8 endmodule
```

RTL7.16a

Message

`Variable index on variable with no range`

Default Severity

Error

Description

A scalar variable with no range is indexed with index 0.

Example

In the example below, variables 'in' and 'out' are scalar variables with no range. The index expressions 'in[0]' and 'out[0]' on line 2 will trigger RTL7.16 violations.

```
1 module test(input in, output out);
2     pbuf pbuf(.a(in[0]), .o(out[0])) ;
3 endmodule
4
5 module pbuf(input a, output o);
6     assign o = a;
7 endmodule
```

RTL7.16b

Message

`Variable index might be out of the defined range`

Default Severity

Warning

Description

A bit or part select reference in an expression is found to have an index specification outside of the defined range of the variable. However, the out-of-range expression occurs within a conditional block that might be, but cannot be conclusively determined during design elaboration to be unreachable.

Example

In the example below on line 7, index of expression 'a[ii+1]' may go out-of-range for loop index 'ii' is equal to 'SIZE - 1'. However, the expression may be unreachable if the condition on line 6 is always false for that particular value of 'ii'.

```
1  module test
2     #(parameter integer A = 0, B = 0, SIZE = 4)
3      (input [SIZE-1:0] a, input [SIZE-1:0] b, output reg out1);
4      always @(a or b) begin
5          for (int ii=0; ii<SIZE; ++ii) begin
6              if (my_pkg::calc_threshold(ii, A, B))
7                  out1 = a[ii+1] & b[ii];
8              else
9                  out1 = a[ii] & b[ii];
10          end
11      end
12  endmodule
```

RTL7.17

Message

```
Exponentiation operator is unsupported
```

Default Severity

Warning

Description

Some synthesis tools do not support the exponentiation operator. Cadence recommends that you remodel your HDL source code.

Example

In the following example, `a ** b` (line 15) is not supported:

```
1  Library IEEE;
2     use IEEE.std_logic_1164.all;
3     use IEEE.std_logic_arith.all;
4  entity TEST1 is
5      port (
6               a : in INTEGER;
7               b : in INTEGER;
8               out1 : out INTEGER
9               ) ;
10  end TEST1 ;
11  architecture ARCH_TEST1 of TEST1 is
12  begin
13  process (a, b)
14  begin
15      out1 <= a ** b;
16  end process;
17 end ARCH_TEST1 ;
```

RTL7.18

Message

```
Argument size to integer type conversion is too large
```

Default Severity

Warning

Description

The argument width is too large for conversion `CONV_INTEGER`. The maximum argument width is 32 for a 'signed' argument or 31 for an 'unsigned' argument.

The tool does truncations for the following:

- Unsigned (31 bits) conversion to integer
- Signed (32 bits) conversion to integer

Example

In the following example, on line 10, the argument for 'conv_integer' is 32 bit width 'unsigned', is too large (maximum if 31 for an 'unsigned' argument).

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.std_logic_arith.all;
4 entity IntCtrl is
5   port( IRQ            : out    integer;
6      IntCtrlReg_th    : in std_logic_vector(31 downto 0));
7   end IntCtrl;
8   architecture rtl of IntCtrl is
9   begin
10    IRQ <= conv_integer(unsigned(IntCtrlReg_th));
11 end rtl;
```

Note: The tool automatically truncates bit 31 to '0'. To override the default behavior, the following command/option can be used:

```
set hdl option -unsigned_conversion_overflow on
```

RTL7.18a

Message

`Expression has ambiguous type`

Default Severity

Error

Description

Indicates that an expression defines a type that can be interpreted in more than one way.

Example

In line 12 of the following example, the expression using "=" is ambiguous in that there are two candidates. One is a predefined function that returns boolean; the other is user-defined function that returns `std_ulogic`.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  entity test is
4     port(q : out std_ulogic; d : in std_ulogic_vector(3 downto 0));
5  end;
6  architecture rtl of test is
7  function "="  ( l,r : std_ulogic_vector ) return std_ulogic is
8   begin
9     return ('1');
10  end "=";
11 begin
12  q <=  std_ulogic(d = "1000");
13 end;
```

RTL7.18b

Message

```
Expression type or size might not be properly resolved
```

Default Severity

Warning

Description

This message will be printed if the expression type or size cannot be properly resolved, or the resolution is ambiguous.

Example

In the following example on line 2, the untyped parameter 'VAL' on the lhs is initialized with an unsized literal '1 on the rhs:

```
1  module ptest (input [31:0] in, output [31:0] out);
2        parameter VAL = '1;
3        assign out = in + VAL;
4  endmodule
```

Note: The untyped literal '1 in the example above will be resolved to 1'b1. This may not be the original intention of the code.

Clarify the intention by using an explicit sized number, for example:

```
2        parameter VAL = 1'b1;
3        assign out = in + {$size(in){VAL}};
...
```

RTL7.19

Message

`Added constraint on integer overflow for arithmetic operation ADD/SUB`

Default Severity

Warning

Description

A constraint is added on the integer overflow for the `ADD` or `SUB` arithmetic operation.

Example

In the following example, in line 11, if `(CONV_INTEGER(a) + b) > 2**31-1` (overflow), the software can derive a constraint on this overflow, and then compare can ignore the different interpretation synthesis tool made when `(CONV_INTEGER(a) + b) > 2**31-1` occurs.

```
1 LIBRARY IEEE;
2 USE IEEE.std_logic_1164.ALL;
3 USE IEEE.STD_LOGIC_ARITH.ALL;
4 USE IEEE.STD_LOGIC_UNSIGNED.ALL;
5 ENTITY test IS
6   PORT ( a, b : IN natural;
7          z : OUT integer );
8 END test;
9 ARCHITECTURE rtl OF test IS
10 BEGIN
11     z <= CONV_INTEGER(a) + b ;
12 END rtl;
```

RTL7.20

Message

```
Size value in size'(expr) casting is too large
```

Default Severity

Error

Description

The size value in `size'(expr)` casting is too large. Use a positive integer value as the size value.

Example

In the following example, the size `'hffffffff'` in line 4 is too large:

```
1 module test(in, out);
2 input in;
3 output out;
4 assign out = 'hffffffff'(in);
5 endmodule;
```

RTL7.21

Message

```
Real variables are not supported (blackboxed)
```

Default Severity

LEC: Error

CCD: Warning

Description

Real variables and real type data assigned to non-parameter variable are not supported in synthesis models. They are converted to 32-bit integers when encountered in the static verification tool flows. If RTL7.21 rule severity is set to Warning, modules with real variables will be blackboxed.

In addition, real number will be rounded to integer value when assigned to non-real type variables.

Example

In the following example, `r1` (line 5 and 7) is not supported:

```
1  module real_in_cmp (b, out1);
2  input [1:0] b;
3  output out1;
4  reg out1;
5  real r1;
6  always @(b)
7  r1 = b;
8  endmodule
```

RTL7.22

Message

`Array index is trimmed`

Default Severity

Warning

Description

When specifying the `-trimindex` option of the `SET HDL OPTION` command before running the `READ DESIGN` command, if the index in a vector bit or part-select expression the index has more bits than necessary with respect to vector address space, one or more most-significant bits will be trimmed from the index.

Example

In the following example, after running `SET HDL OPTION -trimindex` and before `READ DESIGN`, `vec[index]` is considered to be equivalent to `vec[index[2:0]]`, and `vect[index+: 2]` is considered to be equivalent to `vec[index[2:0] +: 2]`:

```
module test();
reg b;
reg [1:0] t;
reg [7:0] index;
reg [5:0] vec;
    assign vec[index] = b;
    assign t = vec[index +:2];
endmodule
```

RTL7.22a

Message

```
Array index size is trimmed by using guide file data.
```

Default Severity

Note

Description

The array index size in the RTL is trimmed by the array index size data specified in the guide file.

Example

RTL7.22b

Message

```
Array index size is conflict with the implementation guide file, no index truncation
     performed.
```

Default Severity

Warning

Description

The array index size in the RTL conflict with the array index size data specified in the guide file.

Example

RTL7.23

Message

```
Signed bit-string is not supported in Verilog 1995
```

Default Severity

Warning

Description

The `1'sb1` signed literal syntax is not supported in Verilog-1995.

Example

In the following example, the signed bit-string in line 5 and 6 is not supported:

```
1 module test(clk, in);
2 input clk;
3 input [7:0] in;
4
5 parameter SB = 1'sb1;
6 parameter SH = 32'sh12345678;
7
8 endmodule
```

RTL7.24

Message

```
System task/function is not supported in Verilog 1995
```

Default Severity

Warning

Description

The system task/function such as `$signed()` and `$unsigned` is not supported in Verilog-1995.

Example

In the following example, the system function called in line 6 and 7 is not supported:

```
1 module test(clk, in);
2 input clk;
3 input [7:0] in;
4 reg [7:0] regA, regB;
5 always @(posedge clk) begin
6  regA = $unsigned(in);
7  regB = $signed(in);
8 end
9 endmodule
```

RTL7.25

Message

```
Illegal operand for constant expression
```

Default Severity

Error

Description

The size of the part-select or slice must be constant, but the position can be variable.

Example

In the following example, index `a` and `b` must be constant:

```
module test();
integer a, b;
wire [7:0] bitvec;
wire [3:0] partsel;
    assign partsel = bitvec[a+: b];
endmodule
```

RTL7.26

Message

```
Direction of slice differs from the prefix range
```

Default Severity

Warning

Description

This message indicates that the direction of part-select slice does not match the declared direction.

Example

On line 10, the direction of part select `to_stdlogicvector(bit_vector'(x))(0 to slice - 1)` is ascending. However, the prefix range of built-in function to_stdlogicvector is descending (see declaration of `to_stdlogicvector`). The mismatch triggers this rule check.

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity top is
4   port(oo : out std_logic_vector(0 to 13));
5 end;
6 architecture arch of top is
7 function SLICE (x: in bit_vector; slice: in integer) return std_logic_vector
8 is variable result: std_logic_vector(slice - 1 downto 0);
9 begin
10    result := to_stdlogicvector(bit_vector'(x))(0 to slice - 1);
11    return result;
12 end;
13   constant tmp: std_logic_vector(0 to 13):=SLICE(X"08b5",14);
14 begin
15   oo <= tmp;
16 end;
```

RTL7.27

Message

```
Wrong slice size specified
```

Default Severity

Warning

Description

An expression in the HDL design contains an invalid or incorrect slice size.

Example

The following example triggers this rule check because the stream operator << in line 2 specifies an incorrect size size of zero, which makes the the module a black box.

```
1  module top (input [3:0] i, output [3:0] o);
2      assign o = {<< 0 {i}};
3  endmodule
```

RTL7.28

Message

```
Size in a size'(expression) casting operation evaluates to a non-positive
```

Default Severity

Error

Description

In SystemVerilog, the size in the `size'(expression)` casting operations must evaluate to a positive integral value. The tool returns this error message if the size is zero, negative, or contains and X or Z value.

Example

In this Verilog example:

```
1  module test(input int i, output int o, o2, o3, o4);
2      assign o  = '1 '(i);
3      assign o2 = 'b1111 '(i);
4      assign o3 = 'x '(i);
5      assign o4 = 4'bzzzz '(i);
6  endmodule
```

LEC gives the error as follows:

```
SETUP> read design -sv test.v
// Parsing file test.v ...
// Error: RTL7.28:  Target width in static cast is non-positive
//  Unsized literal 'x is evaluated to non-positive value
//  on line 4 in file 'test.v'
// Error: RTL7.28:  Target width in static cast is non-positive
//  Expression is evaluated to non-positive value
//  on line 5 in file 'test.v'
```

RTL7.29

Message

```
Range of concatenation was illegal, shift the range
```

Default Severity

Warning

Description

Indicates that a concatenation range has a negative value and the range will be shifted.

Example

Conformal would report RTL7.29 with the following VHDL design in VHDL87

```
library ieee;
use ieee.std_logic_1164.all;
entity top is port(
A B : in std_logic_vector (3 down 0);
C : out std_logic_vector (0 to 7);
end;

architecture arch of top is

function con(a,b : in std_logic_vector) return std_logic_vector is

begin
     return a & b;
end

begin
     C <= con(A, B);
end;
```

In VHDL 87, the concatenation would have range start as the left of the L operand. In this case, the con(A, B) would become a std_logic_vector (3 down to -4). But it is not correct, the array would be shifted to (7 downto 0).

RTL7.30

Message

`Enum value is illegal`

Default Severity

Warning

Description

When evaluating an expression of type enum, the resulting enum value is illegal.

Example

In the following example, the value of enum variable 'v4' is 'four' and is the last value of enum type 'e_type'. Evaluation of 'v4.next()' on line 4 resulted in an illegal value.

```
1  typedef enum { one=1, two=2, three=3, four=4 } e_type;
2  module test(input [1:0] in, output e_type out);
3    localparam e_type v4 = four;
4    assign out = v4.next();
5  endmodule
```

RTL7.31

Message

```
Enum method has error or unsupported usage
```

Default Severity

Error

Description

The tools encounter enum method or usage of enum method that has error or unsupported usage. The containing module will be blackboxed.

Example

In the following example, the argument 'in' to the enum 'next()' method on line 4 is not a constant value, and in general, is not supported by synthesis tools.

```
1  typedef enum { one=1, two=2, three=3, four=4 } e_type;
2  module test(input [1:0] in, output e_type out);
3    localparam e_type v4 = one;
4    assign out = v4.next(in);
5  endmodule
```

RTL7.32

Message

`X or Z assignment to a 2-state variable`

Default Severity

Warning

Description

The rhs expression of an assignment to a 2-state variable in System Verilog is a constant with 'x' or 'z' value(s).

Example

In the example below, variable 'st1' is a 2-state variable (of type byte). All the 'x's values on the rhs of assignment to 'st1' on line 5 are converted to '0's.

```
1  module test (input clk, output byte out1);
2      byte st1; // 2 state
3      always@(posedge clk)
4      begin
5          st1 = 'x;
6          out1 <= st1;
7      end
8  endmodule
```

Note: In System Verilog, a 2-state variable can only assume the value of '0' or '1'. This could result in interpretation mismatch between simulation and synthesis.

RTL7.32a

Message

`4-state to 2-state variable assignment`

Default Severity

Warning

Description

The rhs expression of an assignment to a 2-state variable in System Verilog is an expression that resolved to a 4-state value.

Example

In the example below on line 5, 'st1' is a 2-state variable (of type byte), but the rhs expression ’in1 & ’x’ resolved to a 4-state value.

```
1  module test (input clk, input byte in1, output byte out1);
2      byte st1; // 2 state
3      always@(posedge clk)
4      begin
5          st1 = in1 & 'x;
6          out1 <= st1;
7      end
8  endmodule
```

Note: In System Verilog, a 2-state variable can only assume the value of '0' or '1'. Hence, in simulation, if the resulting value contains 'x' or 'z' it will be converted to '0' and this could result in mismatches between simulation and synthesis.
`RTL7.32a` will not be issued in the case of direct assignment of a 4-state variable or function to a 2-state variable. In such case, it is assumed that the assignment intent is to convert the 4-state value into a 2-state value.

(see rule `RTL7.32f`)

RTL7.32b

Message

```
Case equality/inequality comparison of constants with X or Z
```

Default Severity

Warning

Description

The operands of a case equality (===) or case inequality (!==) operator are both constants with 'x's or 'z's.

Example

In the example below, the condition on line 6 will evaluate to 'true'. The first operand 'st' is resolved to 8'bxxxxxxxx and the second operand 'x is extended to 8'bxxxxxxxx.

```
1  module test (input clk, output reg out1);
2        reg [7:0] st1;
3        always@(posedge clk)
4        begin
5              st1 = 'x;
6              if (st1 === 'x)
7                out1 = '1;
8              else
9                out1 = '0;
10        end
11  endmodule
```

RTL7.32c

Message

```
Case equality/inequality comparison of 2-state variable and constants with X or Z
```

Default Severity

Warning

Description

The case equality (===) or case inequality (!==) operator is comparing 2-state variable(s) with constants containing 'x's or 'z's.

Example

In the example below, the comparison on line 4 will evaluate to 'false' since 'in' is a 2-state variable of type 'bit [7:0]' and each of its elements can only assume the value of '0' or '1'.

```
1   module test (input clk, input bit [7:0] in, output reg out1);
2
3       always@(posedge clk)
4       begin
5           if (in === '8’b0000001x)
6               out1 = '0;
7           else
8               out1 = in;
9       end
10  endmodule
```

RTL7.32d

Message

```
X or Z value in 2-state enum type declaration
```

Default Severity

Error

Description

The enumerated type declaration with a 2-state base type includes one or more names with x or z assignments.

Example

In the example below, enumerated type 'my_enum' is of a 2-state base type 'bit [1:0]'. The assignments containing 'x' or 'z' on line 4 and line 5 are therefore illegal.

```
1   typedef enum bit[1:0] {
2       A1 = 2'b00,
3       B1 = 2'b01,
4       X1 = 2'b0x,
5       Z1 = 2'bzx
6   } my_enum;
7
8   module test(input my_enum foo, output [1:0] bar);
9       assign bar = foo;
10  endmodule
```

RTL7.32e

Message

```
X or Z assignment to packed structure of mixed 2-state and 4-state
```

Default Severity

Warning

Description

The lhs of an assignment is a packed structure with mixed 2-state and 4-state members and the rhs of the assignment contains 'x's or 'z's.

Example

In the example below, packed structure 't_bundle_s' on line 5 has both 2-state member ('t_state_e control') and 4-state member ('logic [5:0] value'). The assignment to variable 'out' of type 't_bundle_s' with value of all 'x's on line 16 will trigger the RTL7.32e rule check message.

```
1 typedef enum bit [1:0] {
2     S0=2'b00, S1=2'b01, S2=2'b10, S3=2'b11
3 } t_state_e;
4
5 typedef struct packed {
6     t_state_e   control;
7     logic [5:0] value;
8 } t_bundle_s;
9
10 module test(input [1:0] sel, input [7:0] in0, in1, output t_bundle_s out);
11     always @*
12     begin
13         case (sel)
14            2'b00: out = in0;
15            2'b01: out = in1;
16            default: out = 'x;
17         endcase
18     end
19 endmodule
```

```
Note: In the example above, 'out' will be treated as a packed array and each
individual bit will be treated as having the value of 'x' (Don't Care or E-gate).
```

RTL7.32f

Message

`4-state 2-state variable to variable assignment`

Default Severity

Note

Description

There is an assignment of a 4-state variable or function to a 2-state variable

Note: `RTL7.32f` will only be issued if HDL option 'SV_2STATE_DASSIGN_CHK' is 'on'.

Example

In the example below on line 5, the lhs 'st1' is a 2-state variable (of type byte) but the rhs 'in1' of the assignment is a 4-state variable (of type wire [7:0]).

```
1  module test (input clk, input [7:0] in1, output byte out1);
2      byte st1; // 2 state
3      always@(posedge clk)
4      begin
5          st1 = in1;
6          out1 <= st1;
7      end
8  endmodule
```

```
Note: In System Verilog, a 2-state variable can only assume the value of '0' or
'1'. Hence, in simulation, if the resulting value contains 'x' or 'z' it will be
converted to '0' and this could result in mis-matches between simulation and
synthesis.
```

RTL7.33

Message

```
Signed 1 Bit Divisor found. Possible simulation mismatch
```

Default Severity

Warning

Description

Rhs expression has a divided operation with a signed 1 bit divisor. It is recommended not to use such constructs to avoid possible simulation or synthesis mismatches.

Example

The following is an example of a construct that can show simulation/synthesis mismatch with the construct on line 6:

```
1 module test(in_0, in_1, z_0);
2 input signed [1:0] in_0;
3 input signed in_1;
4 output z_0;
5 wire m_0;
6 assign m_0 = in_0/in_1;
7 assign z_0 = m_0;
8 endmodule
```

RTL8.1

Message

```
Multiple multipliers/dividers are in module/entity
```

Default Severity

Note

Description

One or more modules or entities have multiple multipliers or dividers.

Example

In the following example, the design uses two multipliers in `module test`. See line 4.

```
1 module test (a, b, c, q);
2 input [3:0] a, b, c;
3 output [11:0] q;
4 assign q = a * b * c;
5 endmodule
```

RTL8.2

Message

```
Latch(es) are inferred due to self assignment
```

Default Severity

Note

Description

Conformal has inferred one or more latches. Conformal infers a latch when it finds a self assignment.

Example

In the following example, both variable 'out1' and 'out2' has self assignment hence two latches inferred.

```
module test (in, clk, out1, out2);
 input  [7:0] in;
 input clk;
 output reg [7:0] out1, out2;
 always @(clk or in)
  begin
   if (clk)
    begin
     out1 <= in;
      out2 <= in;
     end
      else
       begin
        out2 <= out2;
       end
   end
 endmodule
```

RTL8.3

Message

```
Unreachable DFF/DLAT is removed
```

Default Severity

Warning

Description

The design includes one or more DFFs or DLATs that are unreachable. Unreachable registers are those registers that do not propagate to any observable point (for example, spare flops).

By default, Conformal removes all unreachable DFFs and DLATs that are local to a module.

If you want to keep the unreachable DFFs and DLATs, use the `READ DESIGN` command's `-keep_unreach` option. For example:

```
read design test.vhdl -vhdl -keep_unreach
```

Example

Example 1:

In the following example, the output of register `out1[1]` does not propagate to any observable point. See line 9.

```
1 module TEST (clk,in1,in2,out0);
2 input clk,in1,in2;
3 output out0;
4 reg    out0;
5 reg[1:0] out1;
6   always @ ( posedge clk )
7     begin
8       out0 <= in1;
9       out1[1] <= in2;
10     end
11 endmodule
```

Example 2:

In the following example, the variable 'tmp' is considered unreachable since it's value is not used other than in the always block where it is assigned to.

```
module test (in, clk, out);
 input  [7:0] in;
 input clk;
 output reg [7:0] out;
 reg tmp;
 always @(clk or in)
   begin
     if (clk)
      begin: latch
       tmp = in[7];
       out <= {in[6:0],tmp};
      end
    end
 endmodule
```

RTL8.4

Message

`Unreachable DFF/DLAT is kept`

Default Severity

Warning

Description

The design includes one or more DFFs or DLATs that are unreachable. Unreachable registers are those registers that do not propagate to any observable point (for example, spare flops).

By default, Conformal removes all unreachable DFFs and DLATs that are local to a module. You will receive this message if you use the `READ DESIGN` command's `-keep_unreach` option. For example:

```
read design test.v -verilog -keep_unreach
```

Example

In the following example, on line 9, DFF `out1[1]` is unreachable:

```
1 module TEST (clk,in1,in2,out0);
2 input clk,in1,in2;
3 output out0;
4 reg    out0;
5 reg[1:0] out1;
6   always @ ( posedge clk )
7     begin
8       out0 <= in1;
9       out1[1] <= in2;
10      end
11 endmodule
```

RTL8.5

Message

`Implicit signed multiplier is detected in module/entity`

Default Severity

Note

Description

Conformal detected (and recognized) the Verilog signed multiplier coding style that uses either sign-extension or subtraction.

Example

In the following example, on lines 5, a signed multiplier is created using sign-extension coding style:

```
1 module test( in1, in2, out);//using sign-extension
2   input [21:0] in1;
3   input [13:0] in2;
4   output [34:0] out;
5   assign out= {{21{in2[13]}}, in2[13:0]} * {{13{in1[21]}}, in1[21:0]};
6 endmodule
```

RTL8.6

Message

```
Detected and re-modeled tertiary expression in case item(s)
```

Default Severity

Note

Description

Conformal detected and re-modeled a tertiary expression coding construct of the form: *CONST1 < VAR <= CONST2* in a case item.

Such constructs will be re-modeled to be interpreted as follows: *(CONST1 < VAR) && (VAR <= CONST2)* to match user intent.

Example

In the following example, the case item expressions will be re-modeled as shown below:

```
module top(input [9:0] data_word, [3:0] word_len);
    unique case (1'b1)
      10'd0   < data_word <= 10'd16 : word_len = 4'd1;
      10'd16  < data_word <= 10'd32 : word_len = 4'd2;
      10'd32  < data_word <= 10'd48 : word_len = 4'd3;
      10'd48  < data_word <= 10'd64 : word_len = 4'd4;
      default : word_len = 4'b0;
    endcase
endmodule
```

will be interpreted as:

```
module top(input [9:0] data_word, [3:0] word_len);

    unique case (1'b1)
      10'd0   < data_word && data_word <= 10'd16 : word_len = 4'd1;
      10'd16  < data_word && data_word <= 10'd32 : word_len = 4'd2;
      10'd32  < data_word && data_word <= 10'd48 : word_len = 4'd3;
      10'd48  < data_word && data_word <= 10'd64 : word_len = 4'd4;
      default : word_len = 4'b0;
    endcase
endmodule
```

RTL8.6a

Message

```
Detected tertiary expression in case item(s)
```

Default Severity

Note

Description

Conformal detected a tertiary expression coding construct of the form: *CONST1 < VAR <= CONST2* in a case item.

Such constructs can be re-modeled to be interpreted as follows: *(CONST1 < VAR) && (VAR <= CONST2)* to match user intent by enabling a global flag.

Example

In the following example, the case item expressions:

```
module top(input [9:0] data_word, [3:0] word_len);
    unique case (1'b1)
      10'd0   < data_word <= 10'd16 : word_len = 4'd1;
      10'd16  < data_word <= 10'd32 : word_len = 4'd2;
      10'd32  < data_word <= 10'd48 : word_len = 4'd3;
      10'd48  < data_word <= 10'd64 : word_len = 4'd4;
      default : word_len = 4'b0;
    endcase
endmodule
```

can be re-modeled as:

```
module top(input [9:0] data_word, [3:0] word_len);

    unique case (1'b1)
      10'd0   < data_word && data_word <= 10'd16 : word_len = 4'd1;
      10'd16  < data_word && data_word <= 10'd32 : word_len = 4'd2;
      10'd32  < data_word && data_word <= 10'd48 : word_len = 4'd3;
      10'd48  < data_word && data_word <= 10'd64 : word_len = 4'd4;
      default : word_len = 4'b0;
    endcase
endmodule
```

RTL9.1

Message

```
Instance inout/output port has dynamic indexing. Treated as floating
```

Default Severity

Warning

Description

The design includes one or more instance inout/output ports with dynamic indexing. An inout or output port can only be connected to a variable or a net. In the case of dynamic indexing, the inout or output port is essentially connected to an expression and the connection can dynamically change depending on the index. As a result, Conformal creates floating signals for the inout/output connections.

Example

In the following example on line 10, outputs `out0` of module TEST is dynamically selected by input `in2`. Hence, out0 becomes floating.

```
1 module SUB (in2, out3);
2 input in2;
3 output out3;
4 assign out3 = in2;
5 endmodule

6 module TEST (in0,in1,in2,out0);
7 input [1:0] in0;
8 input in1,in2;
9 output [1:0] out0;
10   SUB inst1 (.in2(in0[in1]),.out3(out0[in2]));
11 endmodule
```

RTL9.2

Message

`Design has irregularly used inout/output expression`

Default Severity

Warning

Description

The design includes one or more irregularly used inout/output expressions.

Example

In the following example on line 11, inout port `inout3` is tied to `0` logic value

```
1 module SUB (in2, inout3);
2 input in2;
3 inout inout3;
4 wire inout3;
5   assign inout3 = in2;
6 endmodule
7
8 module TEST (in0,in1,in2);
9 input [1:0] in0;
10 input in1,in2;
11   SUB inst1 (.in2(in0[in1]),.inout3(1'b0));
12 endmodule
```

RTL9.3

Message

```
Supply0/supply1 is converted to wire
```

Default Severity

Warning

Description

The `supply0` or `supply1` net is converted into a wire.

Example

In the following example, `supply0 net in2` is converted to a wire. You need to specify the `-nosupply` option of the `read design` command.

```
module test (in, out);
input in;
output out;
supply0 in2;
assign out = in & in2;
endmodule
```

RTL9.4

Message

`Supply0/supply1 net is not a module port`

Default Severity

Warning

Description

The `supply0` or `supply1` net is not a module port.

Example

In the following example, `supply0 in2` is not a module port (see line 4). You need to specify the `-nosupply` option of the `read design` command.

```
module test (in, out);
input in;
output out;
supply0 in2;
assign out = in & in2;
endmodule
```

RTL9.5

Message

```
Power pin is converted to input pin
```

Default Severity

Warning

Description

A power pin is converted to an input pin.

Example

In the following example, when running the following Conformal commands:

```
set power pin -input vss vdd
read des test.v -fix_power_pin
```

the vss and vdd pins are changed from inout to input pins.

```
input in;
output out;
inout vss, vdd; // changed to input pins
supply0 vss;
supply1 vdd;

assign out = !in;
endmodule
```

RTL9.6a

Message

`Undriven inout port is converted to input port`

Default Severity

Warning

Description

This message warns you that when you use the `undriven_port_to_in` option of the `SET HDL OPTION` command, before running the `READ DESIGN` command, undriven inout ports are converted to input ports.

Example

For the following example, if you use the `SET HDL OPTION -undriven_port_to_in on` command before `READ DESIGN`, the undriven inout ports, port `a` and `b`, will be converted to input ports.

```
module top(a, b, out1);
inout a, b;
output out1;
assign out1 = 1'b0;
endmodule
```

RTL9.6b

Message

```
Undriven output port is converted to input port
```

Default Severity

Warning

Description

This message warns you that when you use the `undriven_port_to_in` option of the `SET HDL OPTION` command, before running the `READ DESIGN` command, undriven output ports are converted to input ports.

Example

For the following example, when you use the `SET HDL OPTION -undriven_port_to_in on` command before `READ DESIGN`, the undriven output port, port `out1`, will be converted to an input port.

```
module top(a, out1, out2);
 input a;
 output out1, out2;
 assign out2 = a;
endmodule
```

RTL9.7

Message

```
Cannot read value from OUT port
```

Default Severity

Error

Description

A feedback loop has been detected through the asynchronous set or resets of flip-flop(s) and the listed signals, wires, or expressions.

Example

In the following example, `dout` (line 5) is the outport. Attempting to read from `dout` will cause an error:

```
library ieee;
use ieee.std_logic_1164.all;
Entity dff is
port (clk: in std_logic;
       reset: in std_logic;
       din: in std_logic;
       dout: out std_logic);
end dff;
Architecture behave of dff is
signal reset_cond: std_logic;
begin
reset_cond <= dout or reset;     <b>
process (clk,din,reset_cond)
begin
if reset_cond = '1' then
              dout <= '0';
elsif clk'event and clk = '1' then
       dout <= din;
end if
end process;
end behave;
```

RTL9.8

Message

```
Set Liberty pin direction to output because it has an output function
```

Default Severity

Warning

Description

This rule check sets the pin direction from the Liberty library to an output because it has an output function.

Example

In the following example, in line 9, the direction of pin `Y` is set to an output because it has an output function:

```
library(test) {
cell (AND2X2) {
  pin(A) {
    direction : input;
  }
  pin(B) {
    direction : input;
  }
  pin(Y) {
    function : "(A B)";
  }
}
}
```

RTL9.9

Message

```
Extra ';' is detected in port list
```

Default Severity

Warning

Description

This rule check indicates that there is an extra semi-colon (`;`) after the last port in the port list.

Example

In the following example, in line 3, there is an extra semi-colon (`;`) after `bit`:

```
entity e is
  port(q : out bit;
    d : in bit;);
end;
architecture a of e is
  begin
    q <= d;
end;
```

RTL9.10

Message

```
Liberty cell does not use any input pin
```

Default Severity

Warning

Description

Indicates that the Liberty cell does not use any input pin.

Example

In the following example, in line 11, no input pin is used in the output function.

```
library(test) {
  cell (AND2X2) {
    pin(A) {
      direction : input;
    }
    pin(B) {
      direction : input;
    }
    pin(Y) {
      direction : output;
      function: "(1)";

    }
  }
  }
```

RTL9.10a

Message

```
Liberty cell modeled as a gray box
```

Default Severity

Warning

Description

The Liberty cell has outputs tied to constant values, but no inputs are connected to outputs. The tool models this cell as a gray box, where the name follows the following pattern:

```
\cdn_gray_box_^<cellname><index>^
```

Note: You can use gray box models to embed special ties/pins/feedthrus in a VHDL or Verilog design. With gray box models, if an incorrect connection is made to an output pin, the tool creates wired logic. This can help catch potential bugs in the design.

Example

In following example, both output pins OA and OB are tied to constant 1'b0 for library cell my_cell.

```
cell (my_cell) {
   ...
   pin (I1) {
    direction : input;
   ...
    }
    pin (I2) {
    direction : input;
    ...
    }
    pin (OA) {
    direction : output;
    function : "0";
    ...
    }
    pin (OB) {
    direction : output;
    function : "0";
    ...
       }
    }
```

After the Liberty library that contains the `my_cell` definition is read in, Conformal issues RTL9.10a and generates a gray-box model for `my_cell`.

RTL9.11

Message

```
Clock phase of Master-Slave DFF cell described in Liberty conflicts between ff group
     attribute and timing_type attribute
```

Default Severity

Error

Description

Indicates there is a conflict on the clock phase between Liberty attributes for a master-slave flip-flop. Liberty attribute ff group indicates this is a negedge triggered flip-flop with two attributes: `clocked_on` and `clocked_on_also`. However, the `timing_type` attribute specified for the output pin of this flop suggests it is a posedge triggered flip-flop.

You can prevent the error by doing one of the following:

- Fix the Liberty file by commenting out the `clocked_on_also` line within the ff group attribute for the corresponding cell.
- Use the `READ LIBRARY` command's `-respect_timing_type_for_flop` option to allow the software to derive the clock phase from `timing_type` attribute.

Example

In the following example, the master latch opens when clock pin `phi` is logic high and the slave latch opens when clock pin `phi` is logic low. This indicates this master slave flop is a negedge triggered flop. However, the `timing_type` for the output pin `q` for this flip-flop is specified as `rising_edge`, which indicates this is a posedge triggered flip-flop.

```
cell (ms_dff) {
  ...
  pin (resetb) {
    direction : input;
  }
  pin (q) {
    function : "IQ" ;
    direction : output;
   ...
    timing() {
      related_pin : "phi";
      timing_type : rising_edge ;
      ...
    }
  }
  ff (IQ,IQN) {
    clocked_on : phi ;
    clocked_on_also : !phi ;
    clear : !resetb ;
    next_state : "(!phi * dp) + (phi * dn)" ;
  }
}
```

RTL9.12

Message

```
Liberty cell with internal_node does not have correct statetable (blackboxed)
```

Default Severity

Warning

Description

Indicates that there is an `internal_node` in the liberty cell, but the corresponding statetable is missing or a corresponding pin of the `internal_node` name is not found in the cell.

Example

In the correct liberty file, if the `internal_node` attribute exists, there should be a corresponding statetable group as following:

```
statetable("in1", "ires"){
...
}
```

but there is no statetable in the cell in the following example:

```
cell(test){
  pin (in1) {
  direction: input;
  }
  pin (ires) {
  direction: internal;
  internal_node: "ires"
  }
  pin (out1) {
  direction: output;
  function: "in1 in2";
  }
}
```

RTL9.13

Message

`Set liberty cell to blackbox because some of its output pins have no function`

Default Severity

Warning

Description

Indicates that some of the liberty cell's output pin has no function attribute in it, but some timing table exist in the output pin. Then the liberty cell is set to be a blackbox.

Example

In the following example, if line 3 is removed, the cell would be marked as a blackbox.

```
pin (out2) {
  direction: output;
  function: "!in1";
  timing(){
  ...
  }
}
```

RTL9.14

Message

```
Liberty cell has duplicate signal (pin/member)
```

Default Severity

Error

Description

Indicates that there are duplicated signals in a bundle declaration of the Liberty cell.

Example

In the following example, the bundle (`z`) has duplicated members `z01`:

```
cell(zhdp_mpt2) {
      bundle(r) {
            members(r0, r1);
            direction : input;
            capacitance : 0.003750;
            fanout_load : 0.188;
            min_pulse_width_high : 0.056;
        }
        bundle(z) {
            members(z01, z01); /* ERROR: z01 is duplicated *
            direction : inout;
            capacitance : 0.005625000;
        }
}       /* cell(zhdp_mpt2) */
```

RTL9.14a

Message

`Liberty cell has duplicate signal of pin and pg_pin (pin/member)`

Default Severity

Note

Description

There are duplicated `pin` and `pg_pin` in the Liberty cell.

Example

In the following example, pin `VDDC` is duplicated:

```
cell (SVH_PGATDRV_G21_1) {
    ...
    pin (VDDC) {
    ...
    }
    pg_pin (VDDC) {
     ...
    }
    ...
```

RTL9.14b

Message

```
Liberty cell has duplicate output internal node used in the statetable
```

Default Severity

Error

Description

Indicates that there are duplicated internal nodes used in the state table declaration of the Liberty cell.

Example

In the following example, RTL9.14b will be issued, because the internal node IQN was used in two statetable declaration

```
statetable ("D1   CK SE SI ", "IQN ") {
table:      "-   ~R -  -    : - : N , \
            H/L R  L  -    : - : L/H , \
            -   R  H  H/L  : - : L/H " ;
}
statetable ("D2   CK SE SI ", "IQN") {
table:      "-   ~R -  -    : - : N, \
            H/L R  L  -    : - : L/H, \
            -   R  H  H/L  : - : H/L" ;
}
```

RTL9.15

Message

```
Size of the liberty cell's input map is different from the size specified in the
     state table
```

Default Severity

Warning

Description

Triggered when the size of the liberty cell's `input_map` is different from the size specified in the `statetable`.

Example

In the following example, `statetable` has three input entries ("`CK D RB`"), however, `input_map` only specifies two input entries("`CK D`").

```
cell(test) {
...
statetable("CK D RB", "IQ") {
 table: "- - L : - : L, \
  R L H : - : L, \
  R H H : - : H, \
  ~R - H : - : N" ;
}
pin(in) {
 input_map: "CK D" ;
 ...
}
...
}
```

RTL9.16

Message

```
Duplicated operating_condition, the second one is ignored
```

Default Severity

Warning

Description

Triggered when an `operating_condition` group has been redefined within the same library group. This can happen when you are reading two liberty files, and both files have the same named library group.

Example

```
library(test_lib){
...
operating_conditions("BEST"){
process:1;
temperature: 9;
voltage : 1.21;
tree_type : "balanced_tree";
}
...
operating_conditions("BEST"){
process :1;
temperature : 9;
voltage : 1.23;
tree_type : "balanced_tree";
}
...
}
```

RTL9.17

Message

```
Cell doesn't have any output pins or all output pins are undriven (blackboxed)
```

Default Severity

Warning

Description

Indicates that in the Liberty there are no output data pins defined for a cell or all of the output pins of a cell are undriven. Therefore, they will be blackboxed.

Example

In the following example, the output pins are blackboxed because all of the output pins are undriven:

```
library(test_lib){
...
 module (test_cell) {
  pin(D) {
 direction : input;
 }
 pin(CLK) {
 direction : input;
 }
 pin(Q){
 direction : output;
 }
 ff (IQ, IQN) {
 next_state : "D";
 clocked_on: "CLK"
 }
 }
 ...
}
```

RTL9.18

Message

```
Port declared as vector, then redeclared as scalar
```

Default Severity

Error

Description

This is triggered when a port is declared as a vector, and then redeclared as a scalar.

Example

In the following example, port `oo` is declared as vector on line 3, then redeclared as scalar on line 4.

```
1 module top(a, oo);
2 input [3:0] a;
3 output [3:0] oo;
4 reg oo;
5 assign oo = a;
6 endmodule
```

RTL9.20

Message

```
Model group of Liberty is unsupported
```

Default Severity

Warning

Description

The Liberty "model" group is not supported.

The model group can accept all the groups and attributes that a cell group accepts (plus `cell_name` simple attribute and short complex attribute).

Example

In the following example, the model group is ignored:

```
model(model_name) {
  area : float;
  ...
  cell_name : cellstring; /*optional*/
  ...
  pin(A, Y) { ...}
  short (A, Y);
}
```

RTL9.21

Message

```
Liberty cell with interface_timing attribute set to true was not an empty cell
```

Default Severity

Warning

Description

Indicates that a liberty cell where the interface_timing attribute is set to true also contains function attributes in the output pin or ff/latch group.

Example

The following triggers this message because the Liberty cell with the interface_timing attribute is set to true, but contains a function attribute in the output pin or ff/latch group used in the cell. Cells with interface_timing attribute set as true, should be empty.

```
cell(test) {

...

interface_timing : true ;
 pin(I1) {
 direction: input ;
 ...
        }
pin(I2) {
 direction: input ;
 ...
        }
pin(O1) {
 direction : output ;
 function: "I1 & I2";
 timing ()
 {
 ...
  }
}
}
```

RTL9.22

Message

```
Liberty cell removed by 'set hdl option -remove_cell' command
```

Default Severity

Note

Description

Notes that, due to the `SET HDL OPTIONS -remove_cell <space separated Liberty cell names>` command, Verilog and SystemVerilog parsers removed the instantiations of the specified Liberty cell names (wildcard names are supported). These cells are commented out and RTL9.22 is issued for each Liberty cell removed by the -`remove_cell` option.

Example

In the following example, at line 5, the instance `phy_inst` of Liberty cell phyc is removed by the "`set hdl option -remove_cell phyc`" command in the following dofile commands:

```
read library -liberty phyc.lib // phyc.lib contains Liberty cell phyc
set hdl option -remove_cell phyc
read design test.v
// test.v
1 module top_wrap (A, B, outx, V1, VVDD, VSS);
2  input A, B, V1, VVDD, VSS;
3  output outx;
4   assign outx = A && B;
5   phyc phy_inst(.VDD(V1), .VSS(VSS)); // phyc is a Liberty cell
6 endmodule
```

To display all the removed Liberty cells, use "`report rule check -verbose RTL9.22`" For example:

```
SETUP> rep ru c -verb RTL9.22
RTL9.22: Liberty cell removed by 'set hdl option -remove_cell ...' command
Type: Golden      Severity: Note       Occurrence: 1
1: top_wrap/phy_inst
on line 5 in file 'test.v'
```

RTL9.23

Message

```
Physical cell found in library
```

Default Severity

Note

Description

Notes that, due to the `SET_HDL_OPTIONS` -merge_physical_cell command, Verilog and SystemVerilog parsers merged the instantiations of Liberty cells that are being identified as a physical cell. Use "`SET_HDL_OPTIONS` -merge_verbose on" to enable `RTL9.23a` and `RTL9.23b` for more detail.

Example

In the following example, the Liberty cells, cell1, cell2, and cell3 will be identified as physical cells by default.

```
cell (cell1) {
  ...
  is_filler_cell : true;
  ...
}

cell (cell2) {
  ...
  is_decap_cell : true
  ...
}

cell (cell3) {
  ...
  is_tap_cell : true
  ...
}
```

With the command `SET HDL OPTIONS  -physical_cell_extract attribute_and_structure`, Conformal also recognizes more physical cells by using structure rules if a cell is black box that has only power/ground pins, and none of the power/ground pins with `pg_type`: internal_power/internal_ground attribute.

RTL9.23a

Message

`Physical cell referenced in design`

Default Severity

Note

Description

A physical cell is referenced in the design. This rule is checked only when `"set hdl option -merge_verbose on"` is enabled.

Example

In following example of Verilog netlist, cell FILLER1 is a kind of physical cell defined in the library.

```
1 module test(VDD, VSS);
2   inout VDD;
3   inout VSS;
4   FILLER1  u1(.VDDA(VDD), .VSSA(VSS));
5 endmodule
```

Conformal shows RTL9.23a for the instance u1 at line 4.

```
RTL9.23a: Physical cell referenced in design
Type: Golden      Severity: Note      Occurrence: 1
1: test/u1
on line 4 in file 'test.v'
```

RTL9.23b

Message

```
Physical cell merged in design
```

Default Severity

Note

Description

A physical cell is merged in the design. This rule is checked only when `"set hdl option -merge_verbose on"` is enabled.

Example

In following example of Verilog netlist, cell FILLER1 is a kind of physical cell defined in the library.

```
1 module test(VDD, VSS);
2   inout VDD;
3   inout VSS;
4   FILLER1  u1(.VDDA(VDD), .VSSA(VSS));
5   FILLER1  u2(.VDDA(VDD), .VSSA(VSS));
6 endmodule
```

Conformal shows RTL9.23b for cell removal at line 5 due to physical cell merging on the instance u2.

```
RTL9.23b: Physical cell merged in design
Type: Golden      Severity: Note       Occurrence: 1
1: test:Merge module FILLER1 instance u2 with u1 at line 4 on line 5 in file 'test.v'
```

RTL9.23c

Message

```
Physical cell(s) is not being merged based on current physical extraction and merge
     setting
```

Default Severity

Note

Description

A physical cell is detected in the design, but is not being merged based on current physical extraction and merge setting.

Example

In following example of Verilog netlist, cell FILLER1 `can be identified` as a kind of physical cell through Conformal structural checking method, but it has no physical cell Liberty attributes. With default setting of ’set_hdl_options -PHYSICAL_CELL_EXTRACT <LIBERTY_ATTRIBUTE>’, FILLER1 is not identified as physical cell and is not merged.

```
1 module test(VDD, VSS);
2   inout VDD;
3   inout VSS;
4   FILLER1  u1(.VDDA(VDD), .VSSA(VSS));
5   FILLER1  u2(.VDDA(VDD), .VSSA(VSS));
6 endmodule
```

Conformal shows RTL9.23c `for both u1 and u2 in line 4 and line 5 to` indicate these 2 possible physical cells are not merged.

RTL9.23d

Message

```
Physical cell excluded from the physical cell list
```

Default Severity

Note

Description

Notes that, due to the `SET HDL OPTION -MERGE_PHYsical_cell` and `-USER_DEFINED_PHYSICAL_CELL` commands would identify the physical cells to merge during Verilog and system Verilog parsing.

The `SET HDL OPTION -EXCLUDE_USER_DEFINED_PHYSICAL_CELL` command could be used to exclude the identified physical cells.

Example

In the following example, if the Liberty cells, cell1, cell2, and cell3 will be identified as physical cells.

If the command option `SET HDL OPTION -EXCLUDE_USER_DEFINED_PHYSICAL_CELL` cell3 is defined.

Then you could see the `RTL9.23d` reported on cell3

```
RTL9.23d: Physical cell excluded from the physical cell list
Type: Golden library     Severity: Note       Occurrence: 2
1: cell3
   on line 77 in file 'LIB/lib/std.lib'
```

RTL9.24

Message

```
Liberty cell has related_bus_pin/related_pin that cannot be found
```

Default Severity

Warning

Description

Indicates that the pin name specified in the `related_bus_pin`/`related_pin` attribute of the timing group does not exist.

Example

Conformal reports RTL9.24 with the following liberty format.

```
Pin (Y) {

...

timing () {
...
related_pin : "NA_PIN";
...
}
...
```

RTL9.25

Message

```
Liberty cell function has been overwritten due to multiple declarations
```

Default Severity

Warning

Description

Indicates that there are multiple function attributes exist in the same liberty pin.

Example

In the following example, pin 'out1' contains multiple function attributes:

```
cell(test) {
    statetable("A", "int OUT") {
    table: "H :  H H : H H, \
            L :  L L : L L";
    }
    pin (in1) {
        direction: input;
    }
    pin (ires) {
        direction: internal;
        internal_node: int;
        input_map: "in1";
    }
    pin (out1) {
        direction: output;
        function: in1;
        state_function: ires; // issue RTL9.25
        internal_node: OUT;   // issue RTL9.25
        input_map: "in1";
    }
}
```

RTL9.26

Message

```
Liberty pin with internal_node use default input_map
```

Default Severity

Note

Description

Indicates that the input_map attribute is missing from liberty pin with internal_node defined.

Example

In the following example, the 'input_map' attribute is missing for pin 'ires'.Conformal reports RTL9.26.

```
cell(test) {
    statetable("A", "int OUT") {
    table: "H :  H H : H H, \
            L :  L L : L L";
    }
    pin (in1) {
        direction: input;
    }
    pin (ires) {
        direction: internal;
        internal_node: int; // issue RTL9.26
    }
    pin (out1) {
        direction: output;
        internal_node: OUT;
        input_map: "in1";
    }
}
```

RTL9.27

Message

```
Liberty cell with clocked_on_also is detected
```

Default Severity

Warning

Description

Indicates the occurrence of a liberty cell using both **clocked_on** and **clocked_on_also** attributes.

Example

In the following two cells would have the same modeling:

```
cell(sampling_transfer_srff) {
    ff( IQ, IQN) {
        next_state: "D";
        clocked_on: "CLK";
        clocked_on_also: "!CLK";
    }   // issue RTL9.27
    ...
}

cell(sampling_transfer_latchs) {
    latch( INT, INTN) {
        data_in: "D";
        enable: "CLK";
    }
    latch( IQ, IQN) {
        data_in: "INT";
        enable: "!CLK";
    }
    ...
}
```

RTL9.27a

Message

```
Model inverted clocked_on_also Liberty cell as DFFs
```

Default Severity

Warning

Description

Indicates that the liberty cell with **clocked_on** and **clocked_on_also** attributes, which are inversion related, is modeled as DFF.

Example

In the following two cells would have the same modeling:

```
cell(sampling_transfer_jkff) {
    ff( IQ, IQN) {
        next_state: "D";
        clocked_on: "CLK";
        clocked_on_also: "!CLK";
    }   // issue RTL9.27a because inverted clocked_on and clocked_on_also
    ...
}

cell(sampling_transfer_jkff) {
    latch( INT, INTN) {
        data_in: "D";
        clocked_on: "CLK";
    }
    latch( IQ, IQN) {
        data_in: "INT";
        clocked_on: "!CLK";
    }
    ...
}
```

RTL9.27b

Message

```
Model clocked_on_also Liberty cell as DFFs
```

Default Severity

Warning

Description

Indicates that liberty cell with **clocked_on** and **clocked_on_also** attributes are modeled as DFF.

Example

The following two cells would have the same modeling if hdl options -read_translate_msff is off:

```
cell(sampling_transfer_jkff) {
    ff( IQ, IQN) {
        next_state: "D";
        clocked_on: "CK1";
        clocked_on_also: "CK2";
    }   // issue RTL9.27b if hdl option -read_translate_msff off
    ...
}

cell(sampling_transfer_jkff) {
    latch( INT, INTN) {
        data_in: "D";
        clocked_on: "CK1";
    }
    latch( IQ, IQN) {
        data_in: "INT";
        clocked_on: "CK2";
    }
    ...
}
```

RTL9.28

Message

```
Attribute specified by 'add notranslate attribute' is no translated
```

Default Severity

Note

Description

Attribute specified by 'add notranslate attribute' is no translated

Example

In the following example, the attribute is_macro_cell would be ignored and rule RTL9.28 would be reported if the command add_notranslate_attribute is applied.

```
cell(macro_and) {
    is_macro_cell: true;// ignore this attribute with command
                        // "add_notranslate_attribute is_macro_cell macro_and"
    pin (A) {
        direction: input;
    }
    pin (B) {
        direction: input;
    }
    pin (O) {
        direction: output;
        function: A&B;
    }
}
```

Verbose example:

    modA: The attribute 'is_macro_cell' is not translated in the module

RTL9.28a

Message

Cell specified by the 'add notranslate attribute' command cannot be found

Default Severity

Warning

Description

Warns that a cell that was specified as a target of the ADD NOTRANSLATE ATTRIBUTE command cannot be found. The ADD NOTRANSLATE ATTRIBUTE command specifies library cell attribute that are parsed, but will not be translated when running the READ LIBRARY command.

Example

RTL9.28b

Message

Notranslate 'is_macro_cell' attribute is not applied

Default Severity

Warning

Description

Indicates that add_notranslate_attribute is_macro_cell does not perform at the reported module. The is_macro_cell attribute can only be removed at a cell with single power/ground pin.

Example

Verbose example:

'is_macro_cell' attribute is kept at cell 'macro_cell' because this cell has multiple power/ground pins

RTL9.29

Message

```
Create EEQ variant from EEQ master
```

Default Severity

Warning

Description

This rule is reported when the following rules are true:
1. No EEQ variant cell definition is found in library space
2. EEQ master cell is defined in library space
3. EEQ variant is defined in LEF or in EEQ master cell attribue "`physical_variant_cell`" in libery.
In LEF, EEQ variant must have same pin names, pin numbers and the pin orders as EEQ master, and add an information EEQ `<EEQ master name>`.

Example

1. LEF example

```
<EEQ master cell>
MACRO CELLA
  CLASS CORE ;
  ORIGIN 0 0 ;
  FOREIGN CELLA 0 0 ;
  SIZE 1.152 BY 0.3 ; SYMMETRY X Y ; SITE mainSite ;
  PIN VDD [...]
  END VDD
  PIN VSS [...]
  END VSS
  PIN A [...]
  END A
  END
END CELLA
```

```
<EEQ Variant cell>
MACRO CELLA_M2
  EEQ CELLA;                 //this means it is EEQ variant
  CLASS CORE ;
  ORIGIN 0 0 ;
  FOREIGN CELLA 0 0 ;
  SIZE 1.152 BY 0.3 ; SYMMETRY X Y ; SITE mainSite ;
  PIN VDD [...]
  END VDD
  PIN VSS [...]
  END VSS
  PIN A [...]
  END A
  END
END CELLA
```

2. Liberty Example

```
cell (NOR) {
…….
   cell_leakage_power : 0.0067;
   physical_variant_cell:"NOR_shl NOR_shr”   //NOR_shl and NOR_shr are EEQ variant
…….
}
```

RTL9.30

Message

```
The mapping relation between EEQ master and variant is not created due to difference
     in their pins
```

Default Severity

Warning

Description

This rule is reported for EEQ variant and EEQ master having differences in pin arrangement, e.g. number of pins or order of pins.

Example

Example in LEF

```
<EEQ master cell>
MACRO CELLA
CLASS CORE ;
ORIGIN 0 0 ;
FOREIGN CELLA 0 0 ;
SIZE 1.152 BY 0.3 ; SYMMETRY X Y ; SITE mainSite ;
PIN VDD [...]
END VDD
PIN VSS [...]
END VSS
PIN A [...]
END A
END
END CELLA
```

```
<EEQ Variant cell>
MACRO CELLA_M2
EEQ CELLA;
CLASS CORE ;
ORIGIN 0 0 ;
FOREIGN CELLA 0 0 ;
SIZE 1.152 BY 0.3 ; SYMMETRY X Y ; SITE mainSite ;
PIN VSS [...]     // CELLA has pin VDD in front and then VSS. Wrong order
END VSS
PIN VDD [...]
END VDD
PIN A [...]
END A
```

```
END
END CELLA
```

RTL10

Message

```
Both posedge and negedge are used in different always/process
```

Default Severity

Warning

Description

Both the posedge and negedge of a clock are used in different *always/process* blocks.

Example

On line 5 of the following example, the design uses the posedge clock in an `always` block. And on line 9, the design uses the negedge clock in another `always` block.

```
module SEN (clk,rst,in0,out0,out1);
input clk,rst,in0;
output out0,out1;
reg    out0,out1;
  always @ ( posedge clk )
    begin
      out0 <= in0;
    end
  always @ ( negedge clk )
    begin
      out1 <= !in0;
    end
endmodule
```

RTL10.1

Message

```
Unrecognized D-flop coding style. No D-flops can be inferred (blackboxed)
```

Default Severity

Warning

Description

There is an unrecognizable D-flop coding style in the `always` block and no D-flops can be inferred from the RTL code.

Example

In the following example, signals '`clk`' and '`rst`' are interpreted as a clock signals because they are not used as data signals inside the `always` block.

No D-flops can be inferred due to unrecognized coding of the single clock control signal.

```
1 module test(clk, rst, data_out);
2 input clk, rst;
3 output data_out;
4 reg data_out;
5     always @(posedge clk or posedge rst)
6             data_out = 1'b0;
7 endmodule
```

To fix this, you can add an '`if`' statement (line 6 below):

```
1 module test(clk, rst, data_out);
2 input clk, rst;
3 output data_out;
4 reg data_out;
5     always @(posedge clk or posedge rst)
6          if (rst == 1)
7             data_out = 1'b0;
8 endmodule
```

RTL11

Message

```
Incomplete condition in a function/procedure/task block may cause mismatch between
     RTL and gate-level
```

Default Severity

Warning

Description

The design includes one or more incomplete conditional statements within functions, procedures, or task blocks.

Examples

- In the following example, the `if` statement is missing its `else` branch. See line 8.

  ```
  module TER (clk,rst,in0,out0);
  input clk,rst,in0;
  output out0;
  wire    out0;
  assign out0 = func1 (in0,rst);
  function func1;
  input in0, rst;
    if (rst)
      func1 = in0;
  endfunction
  endmodule
  ```

- Incomplete conditional statements might lead to false positive result under certain circumstance and should be carefully examined.

  In the following example, the variable parameter type output in the subprogram might get an ambiguous value because of the incomplete conditional statements. In this testcase, `nxt_state` is associated with a variable parameter of mode OUT in procedure `next_state`. Inside the procedure, when x = 3, there is no assignment to z. In this case, `nxt_state` does not hold its original value of 3 after calling procedure `next_state`. Instead, it gets a value `0`.

  ```
  library ieee;
  use ieee.std_logic_1164.all;
  ```

```
entity test is
    procedure next_state(x: in integer range 0 to 3;
                         z: out integer range 0 to 3) is
    begin
      case x is
        when 1 => z := 1;
        when 2 => z := 2;
        when others => NULL;
      end case;
  end next_state;
end test;
architecture rtl of test is
begin
    process
      variable cur_state: integer range 0 to 3;
      variable nxt_state : integer range 0 to 3;
    begin
      cur_state := 3;
      nxt_state := 3;
      next_state(cur_state, nxt_state);
    end process;
end rtl;
```

In the following example, 'var2' is not assigned in the 'then' branch of the'if' statement in function 'foo'. Conformal will report RTL11 rule violation. Note that simulation may initialize 'var2' to 1'b0 and may cause simulation to synthesis mismatches.

```
module test(input s, in1, in2, output reg out);

function bit foo(a,b,c);
   bit var1, var2;
   if (a)
      var1 = b;
   else begin
      var1 = c;
      var2 = c;
   end
   foo = var1 & var2;
endfunction

always_comb
   out = foo(s, in1, in2);
endmodule
```

RTL11a

Message

```
Incomplete condition in an automatic function/procedure/task block may cause
     mismatch between RTL and gate-level
```

Default Severity

Warning

Description

The design includes one or more incomplete conditional statements within SV automatic functions blocks.

Examples

In the following example, 'var2' is not assigned in the 'then' branch of the'if' statement in function 'foo'. Conformal will report RTL11a rule violation. Note that simulation may initialize 'var2' to 1'b0 and may cause simulation to synthesis mismatches.

```
module test(input s, in1, in2, output reg out);

function automatic bit foo(a,b,c);
   bit var1, var2;
   if (a)
      var1 = b;
   else begin
      var1 = c;
      var2 = c;
   end
   foo = var1 & var2;
endfunction

always_comb
   out = foo(s, in1, in2);
endmodule
```

RTL11b

Message

`Incomplete condition when evaluating constant function`

Default Severity

Warning

Description

One or more incomplete conditional statements are encountered when evaluating the function in a constant context, such as when evaluating variable range or generate statement condition.

Examples

In the following example, function 'func1' is called with ’sel == 2’, which is not handled by any of the branches. The return value is therefore uninitialized. Note that by default LEC set the uninitialized return value to '0'.

```
   module test
      #(parameter int VAL1 = 2, VAL2 =2, SEL = 2)
       (output reg [31:0] q);
       function 280

integer func1(int sel, int val1, val2);
           if (sel == 0)
               func1 = val1;
           else if (sel == 1)
               func1 = val2;
       endfunction
        localparam int TMP_NBITS = func1(SEL, VAL1, VAL2);
        localparam int NUM_NBITS = (TMP_NBITS > 1)? TMP_NBITS-1 : 1;

        function automatic bit [NUM_NBITS-1:0] f_mask();
            f_mask = '1;
        endfunction
        assign q = f_mask();
    endmodule
```

RTL11c

Message

`Incomplete condition when evaluating constant automatic function`

Default Severity

Warning

Description

One or more incomplete conditional statements are encountered when evaluating the automatic function in a constant context, such as when evaluating variable range or generate statement condition.

Examples

In the following example, function 'func1' is called with ’sel == 2’, which is not handled by any of the branches. The return value is therefore uninitialized. Note that by default LEC set the uninitialized return value to '0'.

```
module test
   #(parameter int VAL1 = 2, VAL2 =2, SEL = 2)
    (output reg [31:0] q);
    function automatic integer func1(int sel, int val1, val2);
        if (sel == 0)
            func1 = val1;
        else if (sel == 1)
            func1 = val2;
    endfunction
    localparam int TMP_NBITS = func1(SEL, VAL1, VAL2);
    localparam int NUM_NBITS = (TMP_NBITS > 1)? TMP_NBITS-1 : 1;

    function automatic bit [NUM_NBITS-1:0] f_mask();
        f_mask = '1;
    endfunction
    assign q = f_mask();
endmodule
```

RTL12

Message

```
Referenced variable(s)/signal(s) are not in sensitivity list
```

Default Severity

Warning

Description

The design includes one or more variables or signals that are referenced but not included in sensitivity lists.

Example

On line 7 of the following example, the design references variable `in2`, which is not included in the sensitivity list on line 5.

```
module SEN (in0,in1,in2,out0);
input in0,in1,in2;
output out0;
reg    out0;
  always @ ( in0 or in1 )
    begin
      out0 = in0 | in1 | in2;
    end
endmodule
```

RTL12.1

Message

`Constant object is referenced in sensitivity list`

Default Severity

Warning

Description

There is a constant object in the sensitivity list.

Example

In the following example, `val` is a constant (see line 6):

```
module SEN (in0,out0);
input in0;
output out0;
parameter val = 1;
reg    out0;
  always @ ( in0 or val )
    begin
      out0 = in0 ;
    end
endmodule
```

RTL12.2

Message

```
Variable(s) referenced in an always_comb block also used as target of assignment(s)
     within the block
```

Default Severity

Warning

Description

A variable is both written to and read from the same SystemVerilog always_comb block, and is not added to the sensitivity list. This can cause a synthesis versus simulation mismatch.

Example

This rule is issued for the following RTL design for line 9:

```
1 module test
2   (
3    input e_comb,
4    output logic b_comb, a_comb
5    );
6
7   always_comb
8     begin
9       a_comb = b_comb;
10       b_comb = e_comb;
11     end
12
13 endmodule
```

RTL13

Message

`The for loop is never entered (is always false)`

Default Severity

Warning

Description

The design includes one or more `for` loops that will never hold true.

Example

On line 7 of the following example, the variable `i` is initialized with a value of 2 (`i=2`). Thus, this `for` loop will never hold true, since `i` will never be less than a value of 1 (`i<1`).

```
module test(out,in);
output [3:0] out;
reg [3:0] out;
input [3:0] in;
integer i;
always begin
  for (i=2; i<1; i=i+1) begin
     out = i;
  end
  out = in;

end
endmodule
```

RTL13.1

Message

`For-loop index or condition variable(s) assigned within the loop`

Default Severity

Warning

Description

The FOR-LOOP index cannot be assigned within the loop.

Example

In the following example, `i` is the FOR-LOOP index and it is assigned within the loop (see line 8):

```
module top(input [1:0] in, output [4:0] out);
sum sum1(in,out);
endmodule
module sum(input [1:0] a, output [4:0]result);
integer i;
  always  begin
  for (i=0; i<3; i=i+1) begin
    i = a;
  end
  end
endmodule
```

RTL13.2

Message

```
The evaluation of for-loop condition is not constant
```

Default Severity

Warning

Description

The Conformal software detected a non-static loop. As per synthesis semantics, a loop in an HDL design must be statically unrollable—that is, the number of loop iterations should be statically known.

Example

In the following example, `N` is not a constant (see line 9):

```
module neg_NSLOOP (a, b, c);
input [1000:0] a, b;
output [1000:0] c;
reg [1000:0] c;
integer N;
always @(a or b)
begin: P
  integer I;
  for (I = 0; I < N; I = I + 1)
    c[I] = a[I] & b[I];
end
endmodule
```

RTL13.3

Message

```
For-generate control variable should be declared using genvar statement
```

Default Severity

Warning

Description

The index loop variable used in a generate for-loop shall be declared as a genvar. This rule indicates that a non-genvar variable is used where a genvar is required.

Example

In the following example, the for-generate is not a 'genvar' variable on line 6:

```
1 module t1(aa, oo);
2 input [7:0] aa;
3 output [7:0] oo;
4
5 integer gv; // should be genvar gv;
6 for (gv = 7; gv >= 0; gv = gv -1)
7   assign oo[gv] = !aa[gv];
8 endmodule
```

RTL13.3a

Message

```
For-generate control variable is not declared or not an integer
```

Default Severity

Error

Description

In the following example, the index loop variable ’n’ used in a generate for-loop is not declared. Conformal will report RTL13.3a with default severity ’error’ and blackbox the enclosing module.

Example

In the following example, the for-generate loop index ’n’ is not declared. Conformal will report RTL13.3a with default severity error and blackbox the enclosing module.

```
module top(input [3:0] x, y, output [3:0] z);
    generate
    for (n=0; n<3; n=n+1) begin: sumit
      submod submod (.x(x[n]), .y(y[n]), .z(z[n]));
    end
    endgenerate
endmodule
```

Note: To resolve this error, simply declare ’n’ as a genvar variable:l

```
module test(input [3:0] x, y, output [3:0] z);
    generate
    genvar n;
    for (n=0; n<3; n=n+1) begin: sumit
      submod submod (.x(x[n]), .y(y[n]), .z(z[n]));
    end
    endgenerate
endmodule
```

RTL13.4

Message

`For-generate control variable assigned a non-constant value`

Default Severity

Error

Description

The index loop variable used in a generate for-loop must be assigned a constant value. This rule is triggered when the generate for-loop is assigned a non-constant value.

Example

In the following example, the `genvar` variable is assigned with a non-constant value in line 6:

```
1 module t1(aa, oo);
2 input [7:0] aa;
3 output [7:0] oo;
4 integer i;
5 genvar gv;
6 for(gv = i; gv >= 0; gv = gv -1)
7 assign oo[gv] = !aa[gv];
8 endmodule
```

RTL13.5

Message

```
The for-loop condition specifies a null range
```

Default Severity

Warning

Description

The for-loop condition specifies a null range.

Example

In the following example, the range `INPUT'left` to `INPUT'right` in line 15 is a null range. To avoid the rule violation, change the range to `INPUT'left downto INPUT'right`.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  entity top is
4  port (
5       A : in  std_logic_vector( 7 downto 0 );
6       B : out integer
7  );
8  end top;
9  architecture rtl of top is
10        function foo ( INPUT : std_logic_vector )
11               return integer is
12        variable sum : integer;
13        begin
14                sum := 0;
15                for i in INPUT'left to INPUT'right loop
16                        if INPUT(i) = '1' then
17                                sum := sum + 1;
18                        end if;
19                end loop;
20                return sum;
21        end;
22 begin
23        B <= foo(A);
24 end rtl;
```

RTL13.6

Message

`Standalone generated blocks are not supported in the IEEE standard`

Default Severity

Warning

Description

In the IEEE standard, you can create generated statements using one of three methods: a generate-loop, a generate-conditional, or a generate-case. Standalone generated blocks are not supported.

Example

The following example triggers this message because the sequential block 'name1' in line 6 is a standalone block. To avoid this warning, change '`begin : name1`' to '`if (1) begin : name1`'.

Before fix (RTL13.6 is reported due at line 6):

```
1 module top (Z,A);
2 output [1:0] Z;
3 input  [1:0] A;
4 generate
5 genvar m;
6 begin : name1
7    for (m = 0; m < 2; m = m+1) begin : name2
8    assign Z[m] = A[m];
9    end
10 end
11 endgenerate
12 endmodule
```

After fix (RTL13.6 no longer reported):

```
1 module top (Z,A);
2 output [1:0] Z;
3 input  [1:0] A;
4 generate
5 genvar m;
6 if (1) begin : name1
```

```
  for (m = 0; m < 2; m = m+1) begin : name2
  assign Z[m] = A[m];
  end
end
endgenerate
endmodule
```

RTL13.7

Message

`Dead branch detected`

Default Severity

Note

Description

Indicates that there is a dead branch in a ternary expression or in an if-then-else statement.

Example

For following RTL design

```
1 module top(input [3:0] sel, output o);
2
3 assign o = (sel > 5 ) ? 1'b0 :
4                         (sel > 3) ? 1'b1 :
5                                     (sel > 7) ? 1'b0 :
6                                                 1'b1;
7
8 endmodule
RTL13.7: Detect dead branches
Type: Golden      Severity: Note       Occurrence: 1
1: top:test.v:5:49 Detect dead branches (ternary expression)
on line 5 in file 'test.v'
```

If previous condition expressions are false - (self > 5) is false and (self > 3) is also false. (sel >7) will never be true and flag it is a dead branch

RTL13.8

Message

```
For-loop has a non-terminating or potentially non-terminating condition
```

Default Severity

Warning

Description

The condition expression of the for-loop is potentially always true and the for-loop never terminates. The loop is most probably marked as non-synthesizable by synthesis tools.

Example

In the example below, if the value of 'cond' is '3'd7', the for-loop will not terminate.

```
module test(out, in, cond);
output reg out;
input [7:0] in;
input [2:0] cond;

reg [2:0] ii;
always @* begin
out = 1'b0;
for (ii = 0; ii <= cond; ii = ii+1) begin
  out = out || in[ii];
end

end
endmodule
```

RTL13.8a

Message

For-loop has a non-terminating condition (always true)

Default Severity

Warning

Description

The condition expression of the for-loop is always true and the for-loop never terminates. The loop is most probably marked as non-synthesizable by synthesis tools.

Example

In the example below, 'ii' is always less than 15, hence the loop will never terminate.

```
module test(out, in, cond);
output reg out;
input [7:0] in;
input [2:0] cond;

reg [2:0] ii;
always @* begin
out = 1'b0;
for (ii = 0; ii <= 15; ii = ii+1) begin
  out = out || in[ii];
end

end

endmodule
```

RTL13.9

Message

Size of for-loop non-constant condition is greater than size of the index

Default Severity

Warning

Description

The size of the loop limit expression is greater than the size of the loop index variable. The loop may never terminate.

Example

In the example below, 'cond' is of size 4-bit and can hold the value of '4'd0' to '4'd15', but 'ii' is 3-bit and can only hold the value between '3'd0' to '3'd7'. For any 'cond' value greater than '4'd8', the loop never terminates.

```
module test(out, in, cond);
output reg out;
input [7:0] in;
input [3:0] cond;

reg [2:0] ii;
always @* begin
out = 1'b0;
for (ii = 0; ii < cond; ii = ii+1) begin
  out = out || in[ii];
end

end

endmodule
```

RTL13.10

Message

`Irregular for-loop transformed to while-loop`

Default Severity

Note

Description

This message informs the user that a for-loop has an irregular structure.

Example

In the following example, a variable in the loop condition is different from the variable in the loop next statement.

```
module test(input [3:0] in, output int out);

     function automatic int func1(input logic [3:0] index);
           for (func1=1; index > 1; func1=func1+1) begin
                 index = index >> 1;
           end
     endfunction

     always @(in)
            out = func1(in);

endmodule
```

Note: Certain irregular for-loops are not supported by all synthesis or synthesis-oriented tools. Recode the for-loop if possible to ensure uniform support across all tools.

RTL13.11

Message

`Bit-select or part-select expression of a genvar variable`

Default Severity

Warning

Description

A bit-select or part-select expression is referencing a genvar variable.

Example

In the following example on line 13, part-select references are made to genvar variable 'idx'. Conformal will issue RTL13.11.

```
 1    module test
 2        #(parameter SZ  = 8,
 3          parameter MIN = 256
 4         )
 5         (input [SZ-1:0] in,
 6          input [15:0] sel,
 7          output [SZ:0] out);
 8
 9        wire [SZ-1:0] x[(MIN*2-1):0];
10        generate
11        genvar idx;
12        for (idx=0;idx<(MIN*2);idx=idx+1) begin: BLK1
13            assign x[idx] = (idx < MIN)? idx[SZ-1:0] : idx[(SZ*2)-1:SZ];
14        end
15        endgenerate
16
17        assign out = x[sel] + in;
18    endmodule
```

RTL14

Message

`Signal has input but it has no output`

Default Severity

Warning

Description

Conformal has removed one or more floating signals. This situation occurs when the output of a logic cone has no fan-out loads. In other words, the signal is driven by a gate or input port (has drives), but its output does not drive any gate or output port (has no loads).

Example

On line 5 of the following example, `flt` does not have any fan-out loads, so Conformal issues the RTL14 warning.

```
module test(aa, bb, o1, o2);
input aa, bb;
output o1, o2;
wire flt;
assign flt = aa != bb;
assign o1 = aa == bb;
endmodule
```

RTL14.1

Message

`Fanout load of the signal is removed`

Default Severity

Warning

Description

Indicates that the fanout load of a signal is removed, and therefore the signal becomes a floating signal, which is also removed.

Example

In the following example, `bb` (on line 5) has fanout load at line 4, but this fanout is removed because `aa` has RTL14 violation. The RTL14.1 rule checker reports that the fanout load of `bb` is removed, and therefore `bb` becomes a floating signal, which is also removed.

```
module test(oo);
output oo;
wire aa, bb, cc;
  assign aa = !bb;   // RTL14 rule violation
  assign bb = !cc;   // RTL14.1 rule violation
endmodule
```

RTL15

Message

`Clock and asynchronous set/reset expression must be one bit wide`

Default Severity

Error

Description

A clock variable or an asynchronous set/reset variable in the sensitivity list is more than one-bit wide.

Example

On line 2 of the following example, `clk` is declared as two-bit wide. On line 6, an RTL15 error is reported because `clk` is not a single-bit and it is used as an asynchronous set/reset variable on the sensitivity list.

```
1  module test(data_in, data_out, clk);
2  input[1:0] clk;
3  input data_in;
4  output data_out;
5  reg data_out;
6  always @(posedge clk) begin
7      data_out <= data_in;
8  end
9  endmodule
```

RTL15.1

Message

```
Else branch of event controlled if statement is not supported
```

Default Severity

Warning

Description

For modeling a flip-flop with asynchronous forces, the clock edge should be the fulfilling condition of the last conditional block. In addition, there should be assignment to output under only one clock edge. Both edges of clock signal are used in a sequential process.

Example

In the following example, both `HIGH` and `LOW` edges of the signal `clk` are used in a sequential process (see lines 19 and 21, respectively):

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity ClockEdge is
  port (sort    :       out unsigned (3 downto 0);
        clk     :       in  std_logic;
        rst     :       in  std_logic;
        ena1    :       in  unsigned (3 downto 0));
end ClockEdge;
architecture rtl_clkedge of ClockEdge is
constant ZERO  : unsigned(3 downto 0) := "0000";
constant HIGH  : std_logic := '1';
constant LOW  : std_logic := '0';
begin
        p0_OK:process (clk,rst)
        begin
                if (rst =HIGH) then
                                sort   <= ZERO;
                elsif ( clk'event and clk=HIGH) then
                                sort <= ena1;
                elsif ( clk'event and clk=LOW) then
                                sort <= ena1 + ena1;
                end if;
        end process p0_OK;
end rtl_clkedge;
```

RTL15.2

Message

```
Sensitivity/clock style is unsupported
```

Default Severity

Warning

Description

The always block is sensitive to both the edges and levels of some signals. Synthesis tools do not support both level and edge sensitive nodes in the sensitivity list.

Example

In the following example, line 5 has both level and edge sensitivity nodes:

```
module mult_clks_in_always2 (clk, q, d, rst1, rst2);
input clk, d, rst1, rst2;
output q;
reg q;
always @(clk or posedge rst1)
begin
    if (rst2)
      q <= d;
end
endmodule
```

RTL15.2a

Message

`Unconventional reset style`

Default severity

Warning

Description

The always/process block has unconventional reset style and may cause modeling differences (NonEquivalent) with synthesis tools.

Example

In the example below, the always block reset expression on line 4 depends on multiple signals from the sensitivity list.

```
1  module test(input clk, rst, gbl_rst, input [3:0] d, output reg [3:0] q);
2      always @(posedge clk or negedge rst or negedge gbl_rst)
3      begin
4         if (!rst && !gbl_rst)
5             q <= 0;
6      end
7  endmodule
```

RTL15.3

Message

```
Clock interferes with data in sequential block
```

Default Severity

Warning

Description

In a sequential block, clock signals could be referenced in the data logic. By default, the Conformal software might interpret the clock reference in data logic as `1'b1` (for posedge clock) or `1'b0` (for negedge clock). This interpretation might not always match the designers expectation.

Example

The following example shows where the Conformal software would issue this message, where `clk` is the clock:

```
always @(posedge clk) q1 <= func(clk, others);
always @(negedge clk) q2 <= func(clk, others);
```

which might be considered equivalent to:

```
always @(posedge clk) q1 <= func(1'b1, others);
always @(negedge clk) q2 <= func(1'b0, others);
```

RTL15.4

Message

```
Multiple statements in clocked always block not supported
```

Default Severity

Warning

Description

This message will be printed when encountering statement(s) outside of the primary block (set, reset, or clocked block) of a clocked always block. The module will be blackboxed.

Example

In the following example, the assignment statement on line-9 is outside of the reset and clocked block of an always flop.

```
1  module test(input clk, rst, d, output reg q);
2      reg tmpq;
3      always @(posedge clk or negedge rst)
4      begin
5          if (!rst)
6              tmpq <= 1'b0;
7          else
8              tmpq <= d;
9          q <= tmpq;         // RTL15.4
10      end
11  endmodule
```

Note: The coding style with statements outside of the primary blocks of a clocked always block is not generally supported by synthesis tools.

RTL15.4a

Message

```
Multiple statements in clocked vhdl process. May cause simulation/synthesis
     mismatch
```

Default Severity

Warning

Description

This message will be printed when encountering statement(s) outside of the primary block (set, reset, or clocked block) of a vhdl clocked process block.

Example

In the following example, the assignment on line-25 is outside of the vhdl clocked process block. This statement is sensitive to any changes on 'clk' or 'rst', hence causing possible mismatch with post-synthesis netlist.

```
14  architecture rtl of my_flop is
15  begin
16
17      process (rst, clk)
18      variable reg_var : std_ulogic_vector(3 downto 0);
19      begin
20          if rst = '0' then
21              reg_var := (others => '0');
22          elsif clk'event and clk = '1' then
23              reg_var := d;
24          end if;
25          reg_var(2) := '0';
26          q <= reg_var;
27      end process;
28  end architecture rtl;
```

Note: The following coding style is recognized by Conformal and will not generate RTL15.4a violation:

```
14  architecture rtl of flop is
15  begin
16      process (rst, clk)
17      variable reg_var : std_ulogic_vector(3 downto 0);
18      begin
19          if rst = '0' then
20              reg_var := (others => '0');
```

```
        elsif clk'event and clk = '1' then
            reg_var := d;
        end if;
        q <= reg_var;
    end process;
end architecture rtl;
```

RTL16.1

Message

`Non-local variable is read in a function body`

Default Severity

Warning

Description

A non-local variable is read in a function body.

Example

On line 4 of the following example, `glob_1` is declared outside of function `f1()`. On line 10, an RTL16.1 warning is reported because `glob_1` is read inside the function body. RTL16.1 warns of a potential mismatch between synthesis and simulation semantics.

```
1  module foo(state, data);
2  input state;
3  output data;
4  reg glob_1; // glob_1 is declared outside of function f1
5  function f1;
6  input state;
7  reg local_1;
8  begin
9       case (state)
10         1'b0: local_1 = glob_1;
11         1'b1: local_1 = 1'b0;
12       endcase
13       f1 = local_1;
14    end
15    endfunction
16    assign data = f1(state);
17  endmodule
```

RTL16.2

Message

`Non-local variable is assigned in a function body`

Default Severity

Warning

Description

A non-local variable is assigned in a function body.

Example

On line 1 of the following example, `glob_1` is declared outside of function `f1()`. On line 7, an RTL16.2 warning is reported because `glob_1` is assigned inside the function body. RTL16.2 warns about the function body contains a side effect due to this external variable assignment.

```
reg glob_1; // glob_1 is declared outside of function f1
  ...
  function f1 ...
    reg local_1;
    ...
  begin
    ...
    glob_1 = local_1; // RTL16.2: glob_1 is assigned inside the function body
    ...
  end
```

RTL16.3

Message

```
Continuous assignment calls a function with non-local variable reference(s)
```

Default Severity

Warning

Description

A continuous assignment statement calls a function that refer to non-local variable(s).

Example

In the following example, the continuous assignment to 'sum' (on line 5) calls the function 'f_sum' which makes a reference to non local variable 'valx' (on line 8).

```
module test (val1, val2, valx, sum);
 input [7:0] val1, val2, valx;
 output [9:0] sum;

 assign sum = f_sum(val1, val2);

 function [9:0] f_sum(input [7:0] v1, input [7:0] v2);
  f_sum = v1 + v2 + valx;
endfunction

endmodule
```

RTL17

Message

`Variable size exceeds the maximum limit`

Default Severity

Error

Description

The variable size exceeds the maximum limit. By default the maximum size of a variable is 4194304. You can use the ’`set hdl option -varsize_limit`’ option to change the maximum size for a variable.

Example

In the following example, the variable `tmp` size exceeds the limit of 8192*8192 (see line 3)

```
module test (out);
output [31:0] out;
wire [8192*8192:0] tmp;
assign out = tmp[31:0];
endmodule
```

RTL17.1

Message

```
Expression size exceeds the maximum limit
```

Default Severity

Error

Description

The expression size exceeds the maximum limit.

Example

The following example triggers the RTL17.1 message because the expression size exceeds limit 67108864. The objects maximum bit size of 67108864 is predefined by all Conformal tools.

```
module test(output r1);
  wire [2147483648:0] r1;  // ERROR: RTL17.1 violation
  assign r1 = 0;
endmodule
```

RTL17.2

Message

```
Port size exceeds the maximum limit(module removed)
```

Default Severity

Error

Description

This is triggered when the port size exceeds the maximum limit.

Example

In the following example, the port size of output port `o1` is 524289 bits, which exceeds the default port size `524288` bits (see line 2). Use the `set hdl option -PORT_SIZE_limit <size>` command to adjust the port size limit.

```
1 module test(o1)
2 output [0: 524288] o1;
3 endmodule
```

RTL17.3

Message

`Array dimension size is required to be greater than zero`

Default Severity

Error

Description

Fixed-size dimensions are represented by an address range, or a single positive number, which specifies the size of a fixed-size unpacked array. Array dimension sizes must be greater than zero. When an array dimension size is not a positive number, `[size]` will become the same as `[0: size-1]`.

Example

In the following example, this message is triggered because the array dimension size is 0, which is not a positive number (see line 2).

```
1 module top;
2 int array[8][0];
3 endmodule
```

RTL17.4

Message

`Instantiation depth exceeds maximum limit (blackboxed)`

Default Severity

Warning

Description

The message indicates that the instantiation depth exceeds the maximum instantiation depth limit specified by the `SET INSTANTIATION DEPTH` command. The default maximum instantiation depth limit is 100.

Example

In the following example, the instantiation depth limit is specified as 2 in the dofile. However, the instantiation depth of module L2 in `top.v` is 3 and exceeds the specified limit.

Conformal reports RTL17.4 and module L2 is blackboxed.

```
// top.v
module L3(ii, oo);
input ii;
output oo;
  L1 inst(ii, oo);
endmodule

module L2(ii, oo);
input ii;
output oo;
  assign oo = ii;
endmodule

module L1(ii, oo);
input ii;
output oo;
  L2 inst(ii, oo);
endmodule

module top(ii, oo);
input ii;
output oo;
  L3 inst(ii, oo);
endmodule
```

```
// dofile
set instantiation depth 2
read design top.v -root top
==========================================================================
```

RTL17.5

Message

```
Variable with very long name encountered
```

Default severity

Note

Description

Conformal encounters variable with very long name (of more than 4095 characters) when processing the design.

Note that this rule is not enabled by default. To enable this rule, use the `set_rule_handling` command:

```
set_rule_handling RTL17.5 -ENABLE_RUle
```

RTL18.1

Message

```
Package is not an IEEE standard
```

Default Severity

Ignore

Description

The IEEE package is not a standard package.

Example

In the following example, `std_logic_arith` is not a standard IEEE package (see line 2):

```
library ieee;
use ieee.std_logic_arith.all;
entity test is
  port (out1    :       out signed (1 downto 0);
    in1    :       in signed (1 downto 0);
    in2    :       in  signed (1 downto 0));
end test;
architecture rtl of test is
begin
  p1:process( in1,in2)
    begin
      out1 <= in1 + conv_integer(in2);
    end process p1;
end rtl;
```

RTL18.2

Message

`Function definition has empty body`

Default Severity

Warning

Description

The function definition has empty body.

Example

In the following example, the function `f1` in line 4 is an empty function:

```
1  module foo(state, data);
2  input state;
3  output data;
4  function f1;
5  input state;
6  reg local_1;
7  begin
8  end
9  endfunction
10    assign data = f1(state);
11  endmodule
```

RTL18.2a

Message

`Task/procedure definition has empty body`

Default Severity

Warning

Description

The task/procedure call refers to an empty task/procedure body.

Example

In the following example, the task enable t1 at line 9 refers to an empty task t1 at line 4:

```
1  module foo(state, data);
2  input state;
3  output data;
4  task t1;
5  input state;
6  output data;
7  endtask
8    always @* begin
9       t1(state, data);
10   end
11  endmodule
```

RTL18.3

Message

```
Function call does not refer to a function definition (blackboxed)
```

Default Severity

Error

Description

The function call does not refer to a function definition.

Example

In the following example, the function func in line 6 is not defined:

```
module test(in,out);
input in;
output out;
reg out;
always @(in)
  out = func(in);
endmodule
```

RTL18.4

Message

```
Ignoring resolution function. This might cause mismatches between simulation and
     synthesis
```

Default Severity

Warning

Description

The 'resolved' function defined in package IEEE STD_LOGIC_1164 is the only supported resolution function. Other resolution functions are ignored.

Example

In the following example, resolution function `RESOLVE_VOLTAGE` is ignored:

```
PACKAGE test IS
      subtype t_simple_integer is integer range -32768 to 32767;
      type t_simple_integer_vector is array (natural range <>) of t_simple_integer;
      type t_drive is ('Z', 'V');
      type UVOLTAGE is
              record
                      V : t_simple_integer;
                      VDRIVE : t_drive;
              end record;
       type UVOLTAGE_VECTOR is array (natural range <>) of UVOLTAGE;
       function RESOLVE_VOLTAGE ( V: UVOLTAGE_VECTOR ) return UVOLTAGE;
       subtype VOLTAGE is RESOLVE_VOLTAGE UVOLTAGE;
       function std_logic_2_voltage (B, VDD: t_simple_integer) return VOLTAGE;
 END test;
```

RTL18.5

Message

`Return value is ignored for a function call`

Default Severity

Warning

Description

This is triggered when the return value of a function call does not assign to a register.

Example

In the following example, the return value of the function call `foo` is ignored on line 6.

```
1 module test;
2 reg addr;
3 function foo;
4 input a;
5 begin
6 foo = a;
7 end
8 endfunction
9 always @(addr)
10 foo(addr);
11 endmodule
```

RTL18.6a

Message

`Extra actual parameters to subprogram call`

Default Severity

Error

Description

This is triggered when there are too many arguments to a subprogram.

Example

In the following example, there are too many arguments to subprogram `add_one` on line 15:

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4 entity top is
5   port(ii: in integer;
6        oo: out integer);
7 end top;
8  architecture rtl of top is
9    function add_one (val : integer)
10     return integer is
11   begin
12     return val+1;
13   end add_one;
14 begin
15  oo <= add_one(ii, 1);
16 end rtl;
```

RTL18.6b

Message

`Subprogram call has too few association elements`

Default Severity

Error

Description

This is triggered when there are too few arguments to a subprogram.

Example

In the following example, there are too few arguments to subprogram `add_x` on line 15:

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4 entity top is
5   port(ii: in integer;
6        oo: out integer);
7 end top;
8 architecture rtl of top is
9   function add_x (val : integer; x: integer)
10     return integer is
11   begin
12     return val+1;
13   end add_x;
14 begin
15   oo <= add_x(ii);
16 end rtl;
```

RTL18.7

Message

`Logical library is not declared`

Default Severity

Warning

Description

This message indicates that a logical library is not declared. You can use the `-map` and `-mapfile` options of the `read_design` or `read_library` command to declare a logical library.

Example

In the following example, the logical library "`my_lib`" on line 3 in `top.vhd` is not declared using `dofile1`. To avoid this message, use `dofile2` to declare the logical library.

```
-- top.vhd
1 library ieee;
2 use ieee.std_logic_1164.all;
3 library my_lib;
4 entity top is
5   port(ii: in std_logic;
6        oo: out std_logic);
7 end top;

// dofile1
read design test.vhd -vhdl

// dofile2
read design -mapfile my_lib test.vhd -vhdl
```

RTL18.8a

Message

```
Unit not found in either the specified library or the default work library
```

Default Severity

Warning

Description

This message indicates that a unit (package or component in VHDL) is not found in the specified library or the default work library.

Example

In the following example, the package "`my_pkg`" is mapped to a logical library "`pkg_lib`" using the dofile. However, the tool cannot find the unit "`my_pkg`", used on line 4 in `top.vhd`, in the "`my_lib`" or the default work library.

```
-- my_pkg.vhd
package my_pkg is
end my_pkg;

-- top.vhd
1 library ieee;
2 use ieee.std_logic_1164.all;
3 library my_lib;
4 use my_lib.my_pkg.all;
5 entity top is
6   port(ii: in std_logic;
7        oo: out std_logic);
8 end top;

// dofile
read design -mapfile pkg_lib my_pkg.vhd -mapfile my_lib top.vhd -vhdl
```

RTL18.8b

Message

```
Unit not found in specified library and will use the one found in the work library
```

Default Severity

Warning

Description

This message indicates that the tool cannot find a unit (package or component in VHDL) in the specified library, but found one in the default work library and will use that one instead.

Example

In the following example, the files `my_pkg.vhd` and `top.vhd` are mapped to the default work library using the dofile. The tool cannot find "`my_pkg`", used on line 4 of `top.vhd`, in the library "`my_lib`".If there is one in the default work library, the tool will use that one.

```
-- my_pkg.vhd
package my_pkg is
end my_pkg;

-- top.vhd
1 library ieee;
2 use ieee.std_logic_1164.all;
3 library my_lib;
4 use my_lib.my_pkg.all;
5 entity top is
6   port(ii: in std_logic;
7          oo: out std_logic);
8 end top;

// dofile
read design my_pkg.vhd top.vhd -vhdl
```

RTL18.9

Message

```
Only wire instance or instances exist in the library module
```

Default Severity

Note

Description

This rule is reported when a library module contains only wire type instance or instances.

Example

In the following example, the library module 'LH' contains only a wire instance.

```
module LH;
  input in;
  output out;
  assign out = in;
endmodule
```

RTL19.1

Message

`Identifier is a reserved keyword and might conflict in designs with mixed languages`

Default Severity

Warning

Description

The identifier is a reserved keyword for Verilog or VHDL.

Example

In the following example, `logic` is a System Verilog keyword (see line 2):

```
module test (input din, output logic);
assign logic = din;
endmodule ;
```

RTL19.3

Message

`Identifier is declared more than once`

Default Severity

Error

Description

This is triggered when an identifier is declared more than once.

Example

In the following example, the identifier `tmp` is redeclared on line 5.

```
1 module test(a, b, o)
2 input a, b;
3 output o;
4 wire tmp;
5 wire tmp;
6 assign tmp = a & b;
7 assign o = tmp;
8 endmodule
```

RTL19.4

Message

```
Identifier is not declared
```

Default Severity

Error

Description

This is triggered when an identifier is not declared.

Example

In the following example, the identifier `abc` on line 3 is not declared.

```
1 module top(oo);
2 output oo;
3 assign oo = abc;
4 endmodule
```

RTL19.5

Message

```
VHDL identifier cannot contain double underscores
```

Default Severity

Error

Description

In VHDL, double underscores are not allowed in identifiers and will result in a lexical error.

Example

In line 1 of the following example, the identifier sub__0 contains a double underscore:

```
1 entity sub__0 is
2   port (oo: out bit);
3 end sub__0;
```

RTL20.1

Message

`Pre-defined attribute is not supported`

Default Severity

Warning

Description

The specified pre-defined attribute in the design is not supported. Remodel the design to avoid this error.

Example

In the following example, the `last_attribute` in line 21 is not supported:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity lat is
port ( din: in bit;
        clk :in bit;
        din1 : in bit_vector(1 downto 0);
        dout : out bit_vector(2 downto 0);
        dout1 : out bit_vector(1 downto 0));
end lat;
architecture behave of lat is
  signal clk1 : bit;
  component lat_bit is
  port ( d: in bit;
      clk :in bit;
        q : out bit);
  end component;
begin
  dout(0) <= din;
  process(clk1,din)
    begin
    if clk1'last_value = '0' and clk1 = '1' then
      dout(2 downto 1) <= din1;
    end if;
  end process;
  I1 : lat_bit port map(din, clk, dout1(1));
  I0 : lat_bit port map(din, clk, dout1(0));
end behave;
```

RTL20.2

Message

`Function is not supported`

Default Severity

Warning

Description

The referenced function is not supported (`1'bx` will be returned).

Example

In the following example, the function `'>'` on line 13 is not supported for type ’time’:

```
1  library ieee;
2  use ieee.math_real.all;
3  ENTITY test_constant_reals IS
4     PORT (
5           sample_out : OUT integer);
6  END test_constant_reals;
7  ARCHITECTURE rtl OF test_constant_reals IS
8     signal    t1, t2 : time;
9     BEGIN
10
11    p0 : process(t1, t2) begin
12
13        if  (t1 > t2) then
14
15        end if;
16    end process p0;
17
END rtl;
```

RTL20.3

Message

`Could not find configuration`

Default Severity

Warning

Description

The referred configuration could not be found.

Example

In the following example, if the configuration `cfg1` is not defined, this message will be issued.

```
entity test is
  port(a, b: in boolean;
          c: out boolean);
end test;
architecture rtl of test is
  component gate
    port(a, b: in boolean;
            c: out boolean);
  end component;
begin
  U1: gate port map(a=>a, b=>b, c=>c);
end rtl;
configuration cfg of test is
    for rtl
        for U1: gate use configuration work.cfg1;
        end for;
    end for;
end cfg;
```

RTL20.3a

Message

`Conflicting instance configuration. Instance is reconfigured`

Default Severity

Warning

Description

This message indicates that an instance was configured more than once, and that the instance was reconfigured and bound to new modules.

Example

In the following example, the instance `inst2` is configured and bound to architecture `arch2` of entity `sub` on line 3 in `cfg.vhd`. However, instance `inst2` is re-configured on line 5. The instance `inst2` will be bound to architecture `arch1` of entity `sub`.

```
-- sub.vhd
library ieee;
use ieee.std_logic_1164.all;

entity sub is
 port(ii:in std_logic;
  oo:out std_logic);
end sub;

architecture arch1 of sub is
begin
 oo <= ii;
end arch1;

architecture arch2 of sub is
begin
 oo <= not ii;
end arch2;

-- top.vhd
library ieee;
use ieee.std_logic_1164.all;

entity top is
 port(ii:in std_logic;
  oo1, oo2:out std_logic);
end top;
```

```
architecture rtl of top is
 component sub is
  port (ii: in std_logic;
   oo: out std_logic);
 end component;
begin
  inst1: sub port map(ii=>ii, oo=>oo1);
  inst2: sub port map(ii=>ii, oo=>oo2);
end rtl;

-- cfg.vhd
1 configuration cfg of top is
2   for rtl
3     for all: sub use entity work.sub(arch2);
4     end for;
5     for inst2: sub use entity work.sub(arch1);
6     end for;
7   end for;
8 end cfg;
```

RTL20.3b

Message

```
Binding error in VHDL configuration
```

Default Severity

Error

Description

VHDL configuration fails to bind the instance to an entity.

Example

In the following example, the instances of '`my_and`' are configured and bound to architecture '`rtl`' of entity '`my_and`' on line 31. However, the binding fails to find the entity/architecture pair. Error is reported as follows:

```
// Error: RTL20.3b: Binding error in VHDL configuration
//  Could not find entity work.my_and(rtl)
//  on line 31 in file 'test.vhd'
1 LIBRARY ieee;
2 USE ieee.std_logic_1164.all;
3
4 entity top is
5   port (
6     A            : in  std_logic_vector(1 downto 0);
7     Y            : out std_logic;
8     Z            : out std_logic);
9 end top;
10
11 architecture rtl of top is
12   component my_and is
13     port (
14       A  : in std_logic_vector(1 downto 0);
15       Y  : out std_logic);
16   end component;
17 begin  -- rtl
18     and_2_wrapper: my_and
19       port map (
20         A  => A,
21         Y  => Y);
22     and_3_wrapper: my_and
23       port map (
24         A  => A,
25         Y  => Z);
26 end rtl;
```

```

configuration config_and of top is
  for rtl
    for all : my_and
      use entity work.my_and(rtl);
      end for;
    end for;

  end for;
end config_and;
```

RTL20.3c

Message

`Could not find root configuration specified design`

Default Severity

Error

Description

The specified root configuration design cannot be found and causes the design elaboration process to stop.

Example

In the following example, in file 'test.sv' on line 14, the design specified in configuration 'cfg1' is the module 'top' from library 'rtlLib'

```
1 module dut1(input in, output out);
2 assign out = in;
3 endmodule
4
5 module dut2(input in, output out);
6 assign out = 1'b0;
7 endmodule
8
9 module top(input in, output out);
10 dut d1 (.in(in), .out(out));
11 endmodule
12
13 config cfg1;
14   design rtlLib.top;
15   instance top.d1 use dut2;
16 endconfig
```

when file 'test.sv' is compiled without '-mapfile' option, or with '-mapfile' option but the library name is not "rtlLib", then design elaboration will fail due to the specified design cannot be found. Conformal will stop the process and throw out error to notify user to review the configuration file setting and the 'read design' command option(s).

RTL20.3d

Message

```
Could not find configuration specified design
```

Default Severity

Warning

Description

The specified configuration design cannot be found.

Example

In the following example, in file 'test.sv' on line 14, the design specified in configuration 'cfg1' is the module 'top' from library 'rtlLib; and on line 19, the design specified in 'cfg2' is the module 'top' from library 'extraLib'.

```
1 module dut1(input in, output out);
2   assign out = in;
3 endmodule
4
5 module dut2(input in, output out);
6   assign out = 1'b0;
7 endmodule
8
9 module top(input in, output out);
10  dut d1 (.in(in), .out(out));
11 endmodule
12
13 config cfg1;
14   design rtlLib.top;
15   instance top.d1 use dut2;
16 endconfig
17
18 config cfg2;
19   design extraLib.top;
20   instance top.d1 use dut1;
21 endconfig
```

when the file 'test.sv' is compiled with "-mapfile rtlLib" option:

```
read design -sv -mapfile rtlLib test.sv -noelaborate
```

and the design is elaborated with "-rootconfig" option:

```
elaborate design -root top -rootonly -rootconfig cfg1
```

Conformal will perform root module 'top' instance binding based on configuration 'cfg1' setting. For configuration 'cfg2', specified design 'extraLib.top' cannot be found but the setting will not impact the design elaboration. Conformal will throw out RTL20.3d rule violation to notify user to review the design setting issue.

RTL20.4

Message

`Integer constant value is not within the integer subtype's valid range`

Default Severity

Warning

Description

This is triggered when the value of the integer constant is not within the specified bounds.

Example

In the following example, the value of integer constant `c1` is 9, which is outside the specified range of 0 to 8 (see line 6).

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity top is
4 end top;
5 architecture rtl of top is
6 constant c1: integer range 0 to 8 := 9;
7 begin
8 end rtl;
```

RTL20.5

Message

```
Found unsupported coding style
```

Default Severity

Error

Description

Indicates that the tool has encountered a coding style that it does not support.

Example

The following example uses a non-synthesizable coding style). To fix this, recode lines 18-20 such that they adhere to the `if (reset)/else(clocking)'` coding style.

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity top is
4   port(reset_n_i: in  std_ulogic;
5          clk_i    : in  std_ulogic);
6 end top;
7 architecture rtl of top is
8   signal reg_sig: std_ulogic_vector(31 downto 0);
9 begin
10   process (reset_n_i, clk_i)
11   variable reg_var : std_ulogic_vector(31 downto 0);
12  begin
13    if reset_n_i = '0' then  -- do asynchronous rest
14      reg_var := (others => '0');
15    elsif clk_i'event and clk_i = '1' then  -- do synchronous clocking
16      reg_var  := reg_sig;
17    end if;
18    reg_var(26) := '0';
19    -- assign register variable to register signal
20    reg_sig <= reg_var;
21  end process;
22 end;
```

RTL20.6

Message

`Multiple default library lists are not allowed`

Default Severity

Error

Description

The `default liblist` clause specifies the default, ordered set of libraries to be searched to find the current instance. You cannot specify more than one `default liblist` clause. The tool uses the first clause, and ignores any other `default liblist` clauses.

Example

The following example defines two `default liblist` clauses. The clause on line 4 is ignored.

```
1 config cfg;
2   design libtop.top;
3   default liblist lib2 lib1;
4   default liblist lib3;
5   instance top.device1 liblist lib1;
6   instance top.device2 liblist lib2;
7 endconfig
```

RTL20.7

Message

`Multiple instance clauses are ignored`

Default Severity

Warning

Description

The instance clause specifies the instance to which an expansion clause should apply. You cannot specify more than one `instance` clause for a particular instance. The tool uses the first clause and ignores any other `instance` clauses.

Example

The following example tries to rebind the instance `top.device1` with modules defined in `lib2`. The clause on line 5 is ignored.

```
1 config cfg;
2   design libtop.top;
3   default liblist lib2 lib1;
4   instance top.device1 liblist lib1;
5   instance top.device1 liblist lib2;
6   instance top.device2 liblist lib2;
7 endconfig
```

RTL20.8

Message

```
Module cannot be found in specified library list
```

Default Severity

Warning

Description

Module cannot be found in specified library list.

Example

In the following example, instance ‘top.ins_c’ is instantiated module ‘c’ on line 5 in file ‘top.v’. In configuration file, user has specified the liblist_claus for instance ‘top.ins_c’ with library list ‘lib_d’ and ‘lib_e’, however, there is no module with name ‘c’ inside these two library list.

```
// top.v
3   a ins_a (.x(x), .y(w1));
4   b ins_b (.x(x), .y(w2));
5   c ins_c (.x(x), .y(w3));
6   d ins_d (.x(x), .y(w4));


// cfg.v
1 config test_cfg;
2 design top_lib.top;
3 default liblist work;
4 instance top.ins_a liblist lib_a;
5 instance top.ins_b liblist lib_b;
6 instance top.ins_c liblist lib_d;
7 instance top.ins_c liblist lib_e;
8 endconfig;
```

RTL20.9

Message

```
Found unresolved instance based on configuration rule
```

Default Severity

Warning

Description

The instantiated module/cell cannot be found in user specified liblist or default liblist.

Example

In the following example, instance ‘top.ins_c’ is instantiated module ‘c’ on line 5 in file ‘top.v’. In configuration file, user has specified the liblist_claus for instance ‘top.ins_c’ with library list ‘lib_d’ and ‘lib_e’. Also user has specified the default library list ‘work’, however, there is no module with name ‘c’ inside these two library list. Also it cannot be found in default library ‘work’. For those instances which has been specified in configuration but cannot be resolved, CFM will report RTL20.9 rule violation.

```
// top.v
3   a ins_a (.x(x), .y(w1));
4   b ins_b (.x(x), .y(w2));
5   c ins_c (.x(x), .y(w3));
6   d ins_d (.x(x), .y(w4));

// cfg.v
1 config test_cfg;
2 design top_lib.top;
3 default liblist work;
4 instance top.ins_a liblist lib_a;
5 instance top.ins_b liblist lib_b;
6 instance top.ins_c liblist lib_d;
7 instance top.ins_c liblist lib_e;
8 endconfig;
```

RTL20.10

Message

```
Instance specified in configuration cannot be found
```

Default Severity

Warning

Description

Instance cannot be found in design tree.

Example

In the following example, module ‘top’ has four instances with name ‘ins_a’, ‘ins_b’, ‘ins_c’ and ‘ins_d’. In configuration file, user has specified the liblist_claus for instance ‘top.c’ with library list ‘lib_d’ and ‘lib_e’, however, there is no instance with name ‘c’ inside module ‘top’.

```
// top.v
  a ins_a (.x(x), .y(w1));
  b ins_b (.x(x), .y(w2));
  c ins_c (.x(x), .y(w3));
  d ins_d (.x(x), .y(w4));

// cfg.v
config test_cfg;
design top_lib.top;
default liblist work;
instance top.c liblist lib_d;
instance top.c liblist lib_e;
endconfig;
```

RTL20.11

Message

No -ROOTConfig specified for design elaboration

Default Severity

Warning

Description

Design contains configuration file while no -ROOTConfig is used in READ DESIGN or ELABORATE DESIGN commands

Example

In the following example, cfg.v contains a configuration for the design hierarchy.

```
SETUP> read design -noelab -sv -mapfile rtl_lib top.v
SETUP> read design -noelab -sv -mapfile cell_lib sub.v
SETUP> read design -noelab -sv cfg.v
SETUP> elaborate design -root top -rootonly
```

Although Conformal will automatically apply the configuration, Conformal issues this rule violation to indicate no explicit -ROOTConfig is used in the command.

```
RTL20.11: No -ROOTConfig specified for design elaboration
    Type: Golden      Severity: Warning    Occurrence: 1
    1: top
        on line 1 in file 'top.v'
```

RTL20.12

Message

Configuration file is used in design elaboration

Default Severity

Note

Description

This rule indicates which design configurations are used and applied to the design elaboration

Example

In the following example, cfg.v and cfg_sub.v contain configurations for the the top and sub module respectively.

```
SETUP> read design -noelab -sv -mapfile rtl_lib top.v
SETUP> read design -noelab -sv -mapfile cell_lib sub.v
SETUP> read design -noelab -sv cfg.v cfg_sub.v
SETUP> elaborate design -root top -rootonly
```

Conformal will issue this rule to indicate the configurations in both cfg.v and cfg_sub.v are being used in the elaboration.

```
RTL20.12: Configuration file is used in design elaboration
    Type: Golden    Severity: Note    Occurrence: 2
    1: sub_cfg
        on line 1 in file 'cfg_sub.v'
    2: top_cfg
        on line 1 in file 'cfg.v'
```

RTL20.13

Message

Multiple configurations for a module

Default Severity

Warning

Description

A module has been specified in design clauses of two or more different configuration files

Example

In the following example, both cfg.v and cfg2.v have specified the top module in the configuration.

```
// cfg.v
config top_cfg;
design rtl_lib.top;
endconfig

// cfg2.v
config top_cfg_beh;
design rtl_lib.top;
endconfig
```

```
SETUP> read design -noelab -sv -mapfile rtl_lib top.v
SETUP> read design -noelab -sv -mapfile cell_lib sub.v
SETUP> read design -noelab -sv cfg.v cfg2.v
SETUP> elaborate design -root top -rootonly
```

Conformal will issue a rule violation to indicate that the module top has been specified in top_cfg_rtl in cfg1.v and in top_cfg_beh in cfg2.v

```
RTL20.13: No -ROOTConfig specified for design elaboration
    Type: Golden      Severity: Warning    Occurrence: 1
    1: top:Module top has multiple configuration in top_cfg_beh and top_cfg
        (top_cfg) on line 1 in file 'cfg.v'
        (top_cfg_beh) on line 1 in file 'cfg2.v'
```

RTL20.14

Message

Instance is not specified in the configuration

Default Severity

Warning

Description

Instance in the design is not specified in the configuration file

Example

In the following example, ’cfg.v’ does not specify library for the hierarchical instance ’top.ins_q.ins_a’ in the configuration file cfg.v. Module 'a' is also not in the default library "work", too

```
1    // cfg.v
2    config gold_config;
3        design top_lib.top;
4        default liblist work;
5        instance top.ins_p liblist lib_p;
6        instance top.ins_q liblist lib_q;
7        instance top.ins_p.ins_a liblist lib_a_1;
8        instance top.ins_p.ins_b liblist lib_b;
9        // instance top.ins_q.ins_a liblist lib_a; // removed
10       instance top.ins_q.ins_b liblist lib_b;
11   endconfig;
```

Conformal will issue a rule violation to indicate that the hierarchy instance top.ins_q.ins_a is not specified and not bound by the configuration file

```
RTL20.14: Instance is not specified in the configuration
    Type: Golden      Severity: Warning    Occurrence: 1
    1: lib_q_q_1:no config found for top/ins_q/ins_a. Using module 'a'
        definition found in library lib_a_1 instead.
        on line 2 in file 'cfg.v'
```

RTL21

Message

`Module blackboxed because memory address space exceeds limit`

Default Severity

Warning

Description

When a memory address space exceeds the specified limit, the module is blackboxed.

Example

For example, you run `SET HDL OPTION -MAXMEM_address_space 4` before `READ DESIGN`, the valid size of memory address space is four. Therefore, a memory declaration like "`reg [7:0] mem [3:0];`" (on line 7) would trigger this rule check.

```
1 module test(din, clk, rw, dout, addr);
2 input [7:0] din;
3 input clk, rw;
4 input [3:0] addr;
5 output [7:0] dout;
6 reg [7:0] dout;
7 reg [7:0] mem [3:0];
8 always @(posedge clk) begin
9 if (rw)
10 dout <= mem[addr];
11 else
12 mem[addr] <= din;
13 end
14 endmodule
```

RTL22

Message

`Incorrect parameter for CW module instance`

Default Severity

Error

Description

An instantiated CW module has invalid parameter data. Check the valid range defined for the parameter within the CW module.

Example

In line 10 of the following example, `a_width` with value `-8` is passed to `CW_mult` at line 10. This error is reported, because the valid range is `>=1`.

```
1 module test ( a, b, tc, out0) ;
2   parameter a_width = -8, b_width = 8, out_width = 16;
3   // inputs
4   input [(a_width - 1) : 0] a;
5   input [(b_width - 1) : 0] b;
6   input tc;
7   // outputs
8   output [(out_width - 1 ) :0] out0;
9
10   CW_mult #(a_width, b_width) U1 (.A(a), .B(b), .TC(1'b1), .Z(out0));
11 endmodule
An error rule violation RTL22 is reported below.
RTL22: Incorrect parameter for CW module instance
Type: Golden      Severity: Warning    Occurrence: 1
1: test:wA=-8. Valid range of wA is >=1.
on line 10 in file 'test.v'
```

RTL23

Message

```
Flop with asynchronous hold might cause non-equivalence with synthesized netlist
```

Default Severity

Warning

Description

Indicates that the design includes a flop with an asynchronous hold condition. By default, Conformal transforms the asynchronous hold condition to its equivalent synchronous hold condition.

In some cases, this transformation can cause nonequivalence with the synthesized netlist. To disable this transformation, use the following command:

```
set hdl options -dff_async_hold on
```

Example

```
always @(posedge clk or posedge hold or negedge reset_n) begin
  if(!reset_n) begin
   shift_reg <= 1'b0;
  end
  else if (hold) begin
   shift_reg <= shift_reg; // Asynchronous Hold
  end
  else begin
   shift_reg <= input_reg;
  end
end
```

RTL24

Message

```
Limited support for external name in VHDL 2008
```

Default Severity

Warning

Description

External name constructs have limited support in VHDL 2008. External variable/constant names are not supported in VHDL 2008. External signal names are treated as local signal names.

Example

In the following example, external name <<signal sub1.sub_sub.test_signala : std_ulogic_vector (0 to 1)>> has limited support:

```
library ieee;
use ieee.std_logic_1164.all;

entity top_module is
port (
        input_single    : in std_ulogic;
        input_vector    : in std_ulogic_vector(0 to 1);
        output_single   : out std_ulogic;
        output_vector   : out std_ulogic_vector(0 to 1);
        test_input      : in std_ulogic_vector(0 to 1);
        test_output     : out std_ulogic_vector(0 to 1)
);
end top_module;

architecture top_arc of top_module is
begin
sub1 : entity work.sub_module
port map (
input_single => input_single,
input_vector => input_vector,
output_single => output_single,
output_vector => output_vector
);

<<signal sub1.sub_sub.test_signala : std_ulogic_vector (0 to 1)>>(0) <=
test_input(0);
<<signal sub1.sub_sub.test_signala : std_ulogic_vector (0 to 1)>>(1) <=
test_input(1);
27 test_output(0) <= <<signal sub1.sub_sub.test_signala : std_ulogic_vector (0 to
1)>>(0);
28 test_output(1) <= <<signal sub1.sub_sub.test_signala : std_ulogic_vector (0 to
1)>>(1);
29 end top_arc;
RTL24: Limited support for external name in VHDL 2008
Type: Golden      Severity: Warning    Occurrence: 1
1: top_module
on line 27 in file 'top.vhdl'
```

RTL25

Message

`Non-automatic task/function`

Default Severity

Warning

Description

Indicates that the task or function is not declared as automatic. This can potentially cause simulation/synthesis mismatch.

Example

For example, the following causes this warning because the function is not declared as automatic:

```
module test(input en, input [7:0] a, b, output reg [8:0] sc);
  function reg [8:0] add1(input en, input [7:0] a, b);
   bit [8:0] tmp = 0;
    if (en) tmp = a + b;
    add1 = tmp;
  endfunction

  always @(a or b)
   sc = add1(en, a, b);
endmodule
```

To avoid this warning, declare the task/function as automatic:

```
module test(input en, input [7:0] a, b, output reg [8:0] sc);
  function automatic reg [8:0] add1(input en, input [7:0] a, b);
   bit [8:0] tmp = 0;
    if (en) tmp = a + b;
    add1 = tmp;
  endfunction
  ...
endmodule
```

RTL26

Message

`Non-constant generate condition`

Default Severity

Error

Description

This message will be printed when the condition of a conditional `generate` statement does not resolve to a constant.

Example

In this example, the variable 'c' in function 'check_connect()' is uninitialized, causing the value of localparam `CHECK_CONNECT` to be undefined ('X). The condition of the if-generate statement on line-6 is therefore considered as a non-constant.

```
1  module test (input [7:0] in, output [7:0] out);
2      parameter int N1 = 0, N2 = 0;
3      localparam int CHECK_CONNECT = check_connect(N1, N2);
4
5      generate
6          if (CHECK_CONNECT > 0)
7              assign out = in;
8          else
9              assign out = 'z;
10      endgenerate
11
12      function automatic int check_connect(int N1, int N2);
13          int c;
14          begin
15              if (N1 > N2) c = N1;
16              check_connect = c;
17          end
18      endfunction
19  endmodule
20
21  module top(input [7:0] in, output [7:0] out);
22      test #(.N1(1), .N2(2)) test(.*);
23  endmodule
```

Note: To correct the violation, use an assignment statement to initialized the value of 'c' prior to it being used.

RTL27

Message

```
Defparam in a generate scope shall not change parameter value outside the scope
```

Default Severity

Error

Description

The defparam statement that under generate block should only change parameter value at downstream scope.

Examples

In the following example, line 10 is allowed because it is changing a parameter of inst_blk1, which lies in the downstream of defparam's scope, namely blk1. On the contrary, line 11 is not allowed because it is changing a parameter of inst, and inst lies outside blk1.

```
1 module top(out1, out2);
2 parameter PAR = 2;
3 output out1, out2;
4
5 sub inst(out1);
6
7 generate
8   if (PAR > 1) begin : blk1
9     sub inst_blk1(out2);
10    defparam inst_blk1.VAL = 1;  // allowed
11    defparam inst.VAL = 1;  // not allowed
12   end
13 endgenerate
14 endmodule
15
16 module sub(out);
17 parameter VAL = 0;
18 output out;
19 assign out = VAL;
20 endmodule
```

RTL29

Message

`Operator is shared`

Default Severity

Note

Description

Indicates that operators (multiplier/divider/adder/subtractor/modulus) were shared in the design during elaboration.

To enable this rule message, use `SET_HDL_OPTIONS -RTL_RPT_EXTRACT_CSE ON` command before the design is read in.

Example

In the following example the multiplier operation '`ain*bin`' are identical in line 5 and 6 so Conformal RTL elaboration will attempt to share the implementation for these with a single multiplier datapath element.

```
1 module top(input [1:0] ain, bin, cin, din, output reg [1:0] zout1, zo ut2);
2
3   always_comb
4   begin
5     zout1 <= ain * bin * din;
6     zout2 <= ain * bin * cin;
7   end
8 endmodule
```

RTL30

Message

`Common sub-expression detected`

Default Severity

Note

Description

The same sub-expression occurs in multiple places and may be combined. This rule check can be used to improve verification and/or implementation result.

Note that RTL30 is not enabled by default. To enable this rule, use the following command:

```
set_rule_handling RTL30 -ENABLE_RUle
```

Example

In the example below, Conformal will report RTL30 on the common sub-expression '`(aa - bb)`' from the continuous assignments on line 5 and line 6.

```
1 module test(input [15:0] a, b, input sel, output [15:0] z0, z1);
2   wire [15:0] aa, bb;
3   assign aa = a;
4   assign bb = b;
5   assign z0 = sel? (aa + bb) : (aa - bb);
6   assign z1 = sel? (aa - bb) : '0;
7 endmodule
```

RTL31

Message

`Possible combinational loop on variable`

Default Severity

Warning

Description

The message indicates that an RTL construct has the potential to create a combinational loop in the design.

Example

In the following example, the element "`mem_elem[mem_idx]`" is referred to both in left hand side and right side element "assign" statement. Such a construct can potentially cause a combinational loop in the design by accident.

```
module top
(
  input [2:0] iidx,
  input en1,
  input [2:0] mem_idx,
  input logic datain1,
  input logic datain2,
  output logic [7:0] mem_elem
);

assign mem_elem[mem_idx] =
       (iidx == mem_idx) ?
        (en1 ?  datain1 : datain2) : mem_elem[mem_idx];

endmodule
```

Report Example:

`REPORT RULE CHECK -v RTL31` will produce the message as follows for the above example:

```
RTL31: Possible combinational loop on variable
    Type: Golden      Severity: Warning    Occurrence: 1
    1: top/mem_elem
       on line 13 in file 'test3.sv'
```

SPICE Netlist Format

This category of rules applies to SPICE netlists used for Conformal Custom. The following lists the SPICE (SPI) rule checks.

SPI1.1

Message

```
Diode instance is ignored
```

Default Severity

Warning

Description

Conformal is ignoring one or more diode instances in the SPICE netlist.

Example

Conformal reports the following:

```
SPI1.1: Ignoring diode instance
Type: Golden design Severity: Warning Occurrence: 1
1: ant/D1
on line 3 in file 'test.sp'
```

That is, Conformal ignores a diode instance called `D1` in the SPICE netlist.

```
.SUBCKT ant y
D1 y vss
.ENDS
```

SPI1.2

Message

```
Incorrect diode instance is ignored
```

Default Severity

Warning

Description

Conformal is ignoring one or more incorrectly instantiated diode instances in the SPICE netlist.

Conformal ignores diodes in all cases. See SPI1.1 on page 533.

Example

In the following example, Conformal generates the SPI1.2 message for line 3 because `D` should be followed by some alphanumeric characters. Also, because Conformal ignores all diodes, line 2 triggers the SPI1.1 message.

```
.subckt foo a b
    D1 a b
    D a b
  .ends
```

SPI1.4

Message

```
Subckt name duplicated and previous ones are ignored
```

Default Severity

Warning

Description

Conformal encountered a subcircuit name that was already used in the SPICE netlist. When this duplication happens, Conformal retains the last definition encountered and ignores earlier ones.

Example

```
.SUBCKT weak out in
mp1 out vss vdd vdd pch w=3 l=2
mn1 out in vss vss nch w=100 l=1
.ENDS
```

```
.SUBCKT weak out in
mp2 out vss vdd vdd pch w=3 l=2
mn2 out in vss vss nch w=100 l=1
.ENDS
```

```
// Command: read des weak.spi -spi
// Parsing file 'weak.spi'
// Golden root module is set to 'weak'
// Warning: (SPI1.4) Subckt name duplicated and previous ones are ignored
(occurrence:1)
// Note: Read SPICE design successfully
// Command: exit -f
```

SPI3.2

Message

```
Model redefined. Using last definition and ignoring earlier ones
```

Default Severity

Warning

Description

Conformal encountered a redefined model in the SPICE netlist. When this redefinition occurs, Conformal uses the last definition encountered and ignores earlier ones.

Example

In the following SPICE netlist example, `P1` is defined as `PMOS` and redefined as `NMOS`. In this case, Conformal recognizes `NMOS` as the definition for `P1`.

```
.model P1 PMOS
.model P1 NMOS
```

SPI4.1

Message

```
The .GLOBAL statement for pin is redefined. Previous definition is ignored
```

Default Severity

Warning

Description

Conformal encountered a redefined `.GLOBAL` statement in the SPICE netlist. When this redefinition occurs, Conformal uses the last definition encountered and ignores earlier ones.

Example

In the following SPICE netlist example, pin `pr` is defined as power and redefined as ground. In this case, Conformal recognizes ground as the definition for pin `pr`.

```
.global pr:P
.global pr:G
```

SPI4.2

Message

```
The .GLOBAL statement for pin with type 'C' is not supported and is ignored
```

Default Severity

Warning

Description

Conformal encountered a `.GLOBAL` statement with pin type 'C' in the SPICE netlist. When this occurs, Conformal ignores that pin type.

Example

In the following SPICE netlist example, pin type 'C' of the global net 'a' will be ignored:

```
.global vdd:P a:C;
```

SPI5.1

Message

`Implicit bulk VDD Net reset to GND due to .GLOBAL declaration`

Default Severity

Warning

Description

The SPICE netlist includes a `.global` declaration that overrides an implicit `VDD` assignment.

By default, Conformal assumes that the bulk node of `PMOS` is `VDD`. The net connected to the bulk node is thus implicitly assigned as `VDD`. However, if there is a `.global` declaration for a `PMOS` net, the global declaration takes precedence over the implicit assignment.

Example

In the following example, net `a` retains its definition as assigned by the `.global` statement on line 1. Conformal ignores the implicit meaning of this net based on the bulk connection (see line 5). Also refer to SPI5.2 on page 540.

```
.global a:G
.global b:P
.subckt inv aa bb out
M1 aa bb VDD a PMOS
M2 aa bb VSS b NMOS
.ends
```

SPI5.2

Message

```
Implicit bulk GND Net reset to VDD due to .GLOBAL declaration
```

Default Severity

Warning

Description

The SPICE netlist includes a `.global` declaration that overrides an implicit `GND` assignment.

By default, Conformal assumes that the bulk node of `NMOS` is `GND`. The net connected to the bulk node is thus implicitly assigned as `GND`. However, if there is a `.global` declaration for an `NMOS` net, the global declaration takes precedence over the implicit assignment.

Example

In the following example, net `b` retains its definition as assigned by the `.global` statement on line 2. Conformal ignores the implicit meaning of this net based on the bulk connection (see line 6). Also refer to SPI5.1 on page 539.

```
.global a:G
.global b:P

.subckt inv aa bb out
M1 aa bb VDD a PMOS
M2 aa bb VSS b NMOS
.ends
```

SPI7.1

Message

`Duplicated pin in subckt. Creating new pin`

Default Severity

Warning

Description

Conformal encountered a duplicate pin name in the subckt definition while reading a SPICE netlist. When this duplication occurs, Conformal creates a new name for the second pin with the following format: `<duplicate_pin_name>_vplx_redundant_<number>`.

Example

In the following SPICE netlist example, pin `a` is defined on line 2 and repeated in the subckt definition on line 3. Conformal creates a new pin for the second `a` it encountered. The new pin is named `a_vplx_redundant_0`.

```
.subckt inv a b a out
M1 a b VDD VDD PMOS
M2 a b VSS VSS NMOS
.ends
```

SPI7.2

Message

`Duplicate pin of subckt with different pin connections in instance`

Default Severity

Warning

Description

Conformal is detecting an instance with duplicated pins in the SPICE netlist:

Example

In the following SPICE netlist example, the second pin of instance `x1` of `subckt inv` is redundant and will be ignored:

```
.global vdd gnd;
.subckt inv in  in;
m1 vdd  in  out vdd p;
m2 out  in  gnd gnd n;
.ends inv;
.subckt top o i;
x1 o i  inv;
.ends top;
```

SPI8.1

Message

```
Undefined Pin in '*.PININFO' is ignored
```

Default Severity

Warning

Description

A SPICE `*.PININFO` statement includes an undefined pin. `*.PININFO` can be used to define pins as input (`I`), output (`O`) or bidirectional (`B`). If `I`, `O`, or `B` is missing for the pin defined in a PININFO statement, Conformal ignores the undefined pin and issues a warning.

Example

In the following PININFO example, pin `a` is defined as `I`. However, on line 2 the definition for pin `b` is missing. When Conformal encounters line 2, it generates the SPI8.1 message.

```
*.pininfo a:I
*.pininfo b:
```

SystemVerilog

This category of rules applies to SystemVerilog designs. The following lists the System Verilog (SV) rule checks.

- SV5.3 on page 599
- SV6.1a on page 600
- SV6.1b on page 601
- SV6.1c on page 602
- SV6.1d on page 603
- SV7.1 on page 604
- SV8.1 on page 605
- SV8.2 on page 606
- SV8.3 on page 607

SV1.1

Message

`covergroup usage is unsupported and ignored`

Default Severity

Warning

Description

The `covergroup` usage is not supported and is ignored.

Example

In the following example, `g2` (line 2) is unsupported:

```
module test( input logic [7:0] in1, reg clk, output logic [7:0] out1);
covergroup g2 @(posedge clk);
endgroup
always@( posedge clk)
begin
  out1 = in1;
end
endmodule
```

SV1.2

Message

```
The 'assert property' is unsupported and ignored
```

Default Severity

Warning

Description

The assert property is not supported and is ignored.

Example

In the following example, assert property `p1` (line 3) is ignored

```
module prop1 (input clk,output logic [7:0] count1);
logic [7:0] counter1 = '0;
  assert property (p1) count1 = counter1;
endmodule
```

SV1.2a

Message

```
The 'expect property' is unsupported and ignored
```

Default Severity

Warning

Description

The expect property is not supported and is ignored.

Example

In the following example, the ’expect’ statement on line 5 including it’s ’else’ clause is ignored. The ’always’ statement on line 1 resolved to an empty block.

```
1  module test(input clk, a, en);
2
3  always @(en) begin
4    if (en)
5    expect ( @ (posedge clk) a) begin
6       $display("OK!!");
7       $finish;
8    end else
9       $error ("Failed");
10 end
11
12 endmodule
```

SV1.3

Message

```
The 'assume property' is unsupported and ignored
```

Default Severity

Warning

Description

The assume property is not supported and is ignored.

Example

In the following example, assume property `p1` is ignored:

```
module prop1 (input clk,output logic [7:0] count1);
logic [7:0] counter1 = '0;
  assume property (p1) count1 = counter1;
endmodule
```

SV1.4

Message

```
The 'cover property' is unsupported and ignored
```

Default Severity

Warning

Description

The cover property is not supported and is ignored.

Example

In the following example, cover property `p1` is ignored:

```
module prop1 (input clk,output logic [7:0] count1);
logic [7:0] counter1 = '0;
  cover property (p1) count1 = counter1;
endmodule
```

SV1.5

Message

```
The 'void' type is not allowed in object declarations
```

Default Severity

Error

Description

The 'void' type is not allowed in the object declaration.

Example

In the following example, it is illegal to declare a `void` type variable for `var1`:

```
module prop1 (input clk,output logic [7:0] count1);
void var1;
logic [7:0] counter1 = '0;
  assume property (p1) count1 = counter1;
endmodule
```

SV1.6

Message

```
Missing argument list for function call
```

Default Severity

Error

Description

Indicates that the function call does not have an argument list.

Example

In the following example, in line 11, the argument list is missing:

```
module test(in, out);
input in;
output out;
function func;
input din;
begin
  func = din;
end
endfunction
always @(in)
  out = func;
endmodule
```

SV1.7

Message

```
Unbounded range parameter '$' is not supported
```

Default Severity

Error

Description

Indicates that the unbounded range parameter '$' is not supported. This is for System Verilog only.

Example

In the following example, in line 6, '$' is used as parameter.

```
module test #(parameter p1 = 1)(input int i, output int out);
always @(i)
  out = i+p1;
endmodule
module top(input int in, output int dout);
test #($) sub(in,dout);
endmodule
```

SV1.8

Message

```
Procedural assertion is unsupported and ignored
```

Default Severity

Warning

Description

Indicates that the procedural assertion is unsupported and is ignored. This is for System Verilog only.

Example

In the following example, the procedural assertion in line 3 is unsupported and ignored.

```
module test (input logic clk,input logic in);
always @(posedge clk)
    if (in == '1) assert (req1 || req2);
    else begin
    $error("assert failed" );
    end
endmodule
```

SV1.9

Message

```
The use of 'distribution weight' is unsupported and ignored
```

Default Severity

Warning

Description

The usage of distribution weight is described in SystemVerilog LRM 13.4.4 Distribution.

The usage of distribution weight is unsupported and is ignored by all Conformal software tools.

Example

In the following example, `x` is equal to `100`, `200`, or `300` with weighted ratio of 1-2-5. The usage of the distribution weight will trigger this warning.

```
x dist {100 := 1, 200 := 2, 300 := 5}
```

SV1.10

Message

`Sequence operation is unsupported and ignored`

Default Severity

Warning

Description

The usage of sequence is described in SystemVerilog LRM 17.5 Sequences.

The usage of sequence is unsupported and is ignored by all Conformal software tools.

Example

In the following example, the usage of sequence operation (`first_match` in line 5) will trigger this warning.

```
sequence t1;
    te1 ## [2:5] te2;
endsequence
sequence ts1;
    first_match(te1 ## [2:5] te2);
endsequence
```

SV1.12

Message

```
number_of_ticks is not a constant value
```

Default Severity

Error

Description

The usage of `$past()` is described in SystemVerilog LRM 17.7.3 The sampled value is:

```
$past( expression1 [, number_of_ticks] [, expression2] [, clocking_event])
```

where the `number_of_ticks` must be 1 or greater. If `number_of_ticks` is not specified, it defaults to 1. The SV1.12 error is reported if `number_of_ticks` is not evaluated to be a constant value.

Example

The following example will trigger this error if `input_port_N` is not a constant expression:

```
$past( bet1, input_port_N)
```

SV1.13

Message

```
number_of_ticks must be 1 or greater
```

Default Severity

Error

Description

The usage of `$past()` is described in SystemVerilog LRM 17.7.3 The sampled value is:

```
$past( expression1 [, number_of_ticks] [, expression2] [, clocking_event])
```

where `number_of_ticks` must be 1 or greater. If number_of_ticks is not specified, then it defaults to 1. The SV1.13 error is reported if `number_of_ticks` is not evaluated to be 1 or greater.

Example

The following example will trigger this error because -1 is not 1 or greater than 1:

```
$past( bet1, -1)
```

SV1.14

Message

```
Sequence method is unsupported and ignored
```

Default Severity

Warning

Description

The sequence usage is described in SystemVerilog LRM 17.5 Sequences.

The sequence methods are described in SystemVerilog LRM 17.12.6.

The SV1.14 warning is reported if sequence methods are used.

Example

The following example will trigger this warning because an ended method is used (line 5):

```
sequence e1;
    @(posedge sysclk) $rose(ready) ##1 proc1 ##1 proc2 ;
endsequence
sequence rule;
    @(posedge sysclk) reset ##1 inst ##1 e1.ended ##1 branch_back;
endsequence
```

SV1.15

Message

```
Recursive properties are unsupported and ignored
```

Default Severity

Warning

Description

The sequence usage is described in SystemVerilog LRM 17.11.4 Recursive property.

The SV1.15 warning is reported if recursive properties are used.

Example

- The following example shows a simple recursive property case, where a property instantiation `prop_always` is inside the `prop_always` property declaration. This causes a recursive property problem.

```
property prop_always(p);
    p and (1'b1 |=> prop_always(p));
endproperty
```

- The following example shows a mutually recursive case, where in property declaration `check_phase1`, there is an instantiation `check_phase2`, and in property declaration `check_phase2`, there is an instantiation `check_phase1`. This causes a mutually recursive property problem.

```
property check_phase1;
    s1 |-> (phase1_prop and (1'b1 |=> check_phase2));
endproperty
property check_phase2;
    s2 |-> (phase2_prop and (1'b1 |=> check_phase1));
endproperty
```

SV1.17

Message

```
Only one default clocking can be specified in a program, module, or interface
```

Default Severity

Error

Description

Indicates that there is default clocking specified more than once in the same program or module that results in a compiler error.

Example

In the following example, line 3 declares `endclocking` as the second default clocking:

```
1 module top( input clk1, clk2, input [3:0] in1, output [3:0] out1);
2    default clocking CLK1 @(posedge clk1) ; endclocking
3    default clocking CLK2 @(posedge clk2) ; endclocking
4 assign out1 = $sampled(in1);
5 endmodule
```

SV1.18

Message

```
Only posedge/negedge expressions are supported on global clocking events
```

Default Severity

Warning

Description

In Verilog/SystemVerilog, an event can be edge-sensitive or level-sensitive. Only posedge and negedge expressions are supported in a clocking event.

Example

In the following example, @(clk) is not supported:

```
module test( input logic [7:0] in1, reg clk, output logic [7:0] out1);
  global clocking
   sys1 @(clk);
  endclocking : sys1
  assign out1 = $sampled(in1, @$global_clock);
endmodule
```

SV1.19

Message

```
Only one global clocking can be specified in an entire elaborated SystemVerilog
     model
```

Default Severity

Error

Description

According to the SystemVerilog IEEE 1800-2009 LRM, there can be no more than one global clocking declaration in an elaborated SystemVerilog model.

Example

In the following example, there are two global clocking declaration.

```
module test( input logic [7:0] in1, reg clk, output logic [7:0] out1);
  global clocking
   sys1 @(negedge clk);
  endclocking : sys1
  global clocking
   sys2 @(posedge clk);
  endclocking : sys2
  assign out1 = $sampled(in1, @$global_clock);
endmodule
```

SV1.20

Message

```
Global clocking declaration does not exist
```

Default Severity

Error

Description

You cannot invoke the `$global_clock` system function when the elaborated SystemVerilog model does not have a global clocking declaration.

Example

The following example is missing a global clocking declaration:

```
module test( input logic [7:0] in1, reg clk, output logic [7:0] out1);
  assign out1 = $sampled(in1, @$global_clock);
endmodule
```

SV1.21a

Message

`Non-constant cycle delay range is unsupported`

Default Severity

Error

Description

After it is computed, the `constant_primary` expression must be an integer value.

Example

In the following example, cycle delay range 'range' is not a constant data

```
module test( input in1, in2, clk, input integer range);
  assert property (@(posedge clk) in1 ##range in2);
endmodule
```

SV1.21b

Message

```
Cycle delay range must be 0 or greater
```

Default Severity

Error

Description

After it is computed, the `constant_primary` expression must be an integer value within `{0, maxint}`. Non-constant data and negative values are not supported.

Example

```
module test #(parameter PA = 5, PB = 8)(input in1, in2, clk);
  assert property (@(posedge clk) in1 ##(PA-PB) in2);
endmodule
```

SV1.21c

Message

```
Cycle delay range exceeds the maximum limit
```

Default Severity

Error

Description

The default upper bound of the `constant_primary` value is set to 1024. If the computed constant primary value is more than upper bound, LEC will error out it and stop elaboration.

Example

```
module test #(parameter PA = 1027, PB = 1)(input in1, in2, clk);
  assert property (@(posedge clk) in1 ##(PA-PB) in2);
endmodule
```

SV1.22

Message

`Checker declaration is unsupported and ignored`

Default Severity

Warning

Description

Checker declarations are not supported and are ignored.

Example

In the following example, checker declaration `my_check1` is ignored:

```
checker my_check1 (logic test_sig, event clock);
default clocking @clock; endclocking
property p(logic sig);
...
endproperty

a1: assert property (p (test_sig));
c1: cover property (!test_sig ##1 test_sig);
endchecker : my_check1
```

SV1.23

Message

Class and related constructs are not supported

Default Severity

Warning

Description

In general, classes and related constructs are non-synthesizable and are ignored.

This message is printed when a class or class-related construct, such as a definition of a class, instantiation of a class object, and access to a class member or a class method is encountered.

Example

In the following example, SV1.23 will be issued on class related constructs on line 1, 7, 10, 11, 12.

```
1  class mycls; // class definition
2     int aa;
3     int bb;
4  endclass
5
6  module top(input [31:0] aa, input [31:0] bb, output reg cc);
7      mycls obj = new; // class instantiation
8      always @(aa or bb)
9      begin
10          obj.aa = aa; // class member access - lhs
11          obj.bb = bb; // class member access - lhs
12          cc = obj.aa + obj.bb; // class member access - rhs
13       end
14  endmodule
```

Note: Classes and related constructs are non-synthesizable. To prevent excessive warning messages, exclude the constructs by using conditional compilation directives such as `ifdef, `else, `elsif, or `ifndef.

SV1.23a

Message

```
Dynamic Array/Queue related constructs are not supported
```

Default Severity

Warning

Description

Indicates the presence of a queue or queue-related usage, such as a reference to a queue object or to a queue element(s), or a function or task call to a queue method.

Example

In the following example, SV1.23a will be issued on queue related usage on line 8 and line 10.

```
1  module test(input clk, input en, input int data,
2              output int out);
3     int values[$];
4     always @(posedge clk)
5     begin
6        out = 0;
7        if (en)
8          values.push_back(data);
9        else
10          out = values.pop_front();
11     end
12  endmodule
```

Task calls and lhs assignment to queue objects or elements will be ignored (the statement become null statement). Function calls and rhs references to queue objects or elements will be replaced with the value 'z'.

SV2.1

Message

`Packed type cannot contain real/shortreal/unpacked type`

Default Severity

Error

Description

Noninteger data types, such as real and shortreal, are not allowed in packed structures or unions. Neither are unpacked arrays.

Example

In the following packed struct type example, the errors are shown by comments:

```
typedef struct packed {
  bit [3:0] GFC;
  real VPI;         // real type is not allowed in a packed structure
  bit PT [3:0] ;    // unpacked type is not allowed a packed structure
  bit [7:0] HEC;
} myPackedType;
```

SV2.2

Message

```
All elements must have same size in a packed union type
```

Default Severity

Error

Description

A packed union shall contain members that must be packed structures, or packed arrays or integer data types all of the same size (in contrast to an unpacked union, where the members can be different sizes). This ensures that a union member that was written as another member can be read back.

Example

In the following packed union type example, the errors are shown by comments:

```
typedef union packed {
  bit [7:0] v1;
  bit [3:0][1:0] v2;
  bit [8:0] v3; // ERROR: v3 has 9 bits, but both v1 and v2 have 8 bits
} badPkUnionType;
```

SV2.2a

Message

```
Elements with different sizes in an unpacked union type
```

Default Severity

Warning

Description

An unpacked union type with elements of different sizes can cause verficiation difference across tools.

This is due to implementation differences between tools for how the elements in the union are packed.

This message is to issue a warning for such cases when encountered in a design.

Example

In the following packed union type example, the errors are shown by comments:

```
typedef union {
    // mismatch sizes here
    shortint sload;  // size 16
    byte bload[4]; // size 32
    } Packet;

module test(output [31:0] out);
    Packet p2;
    assign p2.sload = 16'hcccc;
    assign out = {<<  {p2} };
endmodule
```

SV2.3

Message

```
An assignment pattern ('{...}) must be used instead when the target is an unpacked
     array or struct type
```

Default Severity

Error

Description

A concatenation is being used in a context where an assignment pattern is required. A concatenation can only be assigned to a bit or integer type. An assignment pattern ('{...}) must be used instead when the target is an unpacked array or struct type.

Example

In the following example, the right hand side expression on line 6 should use assignment pattern " '{ 0:d1, 1:d2, 2:1'b0, 3:1'b1 } " instead:

```
1  module test_09(q, d1, d2);
2      output bit q [3:0];
3      input d1;
4      input d2;
5
6      assign q = { 0:d1, 1:d2, 2:1'b0, 3:1'b1 };
7  endmodule
8
```

SV2.4

Message

```
Illegal assignment pattern context
```

Default Severity

Error

Description

An assignment pattern type must constrain to an array or struct type. Using assignment patterns in any other context is not allowed.

Example

In line 8 of the following example, the context in which the assignment pattern `{in0,a}` is used is not supported:

```
1 typedef struct { bit[7:0] opcode ; bit  r; }mytagged;
2
3 module test( in0,in1, out0,out1,a ,b ) ;
4 mytagged c [1:0] ;
5 input [7:0] in0,in1;
6 output [8:0] out0,out1  ;
7 input a,b;
8 assign c = {'{in0,a},{in1,b}};
9 assign out0 = {c[1].opcode,c[1].r};
10 endmodule
```

SV2.5

Message

```
Multiple specifications for a data type occur within an assignment pattern
```

Default Severity

Warning

Description

The assignment pattern specifies a data type (implicitly or explicitly) more than once.

Only the last specified value will be applied.

Example

In line 7, the assignment pattern contains matching types (int and myint) therefore specifying the values of out.addr and out.data more than once. Only the last value, 32'd0 will be applied.

In line 8, the data int type is used more than once in the assignment pattern. Only the last value, 32'd1 will be applied.

```
1  typedef int myint;
2  typedef struct {
3    int    addr;
4    myint  data;
5  } S;
6  module test(output S out, out2);
7    assign out = '{int:32'd1, myint:32'd0};
8    assign out2 = '{int:32'd0, addr:32'd0, int:32'd1};
9  endmodule
```

SV2.5a

Message

```
Ambiguous type(s) or specification(s) in assignment pattern
```

Default Severity

Warning

Description

Tools type or specification can cause ambiguous interpretation in an assignment pattern.

Example

In the following example, the target of an assignment pattern is of type union. Though some tools may allow it, assignment pattern is specified for assignment to structure fields and array elements, and not for assignment to union members.

```
module test( input int in0, output int out0);

  typedef union {
    bit isfloat;
    union { int i; byte f; } n;
  } mtagged;

  mtagged a;
  assign a.n = '{in0, 1}; // a.n is a union
   assign out0 = a.n.i;
endmodule
```

SV2.5b

Message

```
Illegal type(s) or specification(s) in assignment pattern
```

Default Severity

Error

Description

The assignment pattern contains types or specifications that are illegal. Most probably, this is caused by type-mismatch with the target of the assignment pattern.

Example

In the example below on line 11, the target of the sub-pattern ‘{default : ‘{default: 1’b}} is a union. This is illegal and rule check `SV2.5b` will be issued.

```
1  package pack;
2      typedef union {
3          logic f; reg sel;
4      } union_t0;
5
6      typedef struct {
7          union_t0 e1;
8      } union_t2;
9
10      typedef union_t2 union_t3[1:0];
11      localparam union_t3 P1 = '{2{'{default : '{default: 1'b1}}}};
12
13  endpackage
14
15  import pack::*;
16  module test (output union_t3 out1);
17      assign out1 = P1;
18  endmodule
```

SV2.5c

Message

```
An assignment pattern may only have an array or structure as target
```

Default Severity

Error

Description

Assignment patterns can only be used as assignments to structure fields and array elements. Any other data types are illegal.

Example

In the following example, the target of an assignment pattern is of type union. Since union is not a legal left-hand side object for assignment pattern, SV2.5c will be reported on line 9.

```
1    module test( input int in0, output int out0);
2
3    typedef union {
4      bit isfloat;
5      union { int i; byte f; } n;
6    } mtagged;
7
8    mtagged a;
9    assign a.n = '{in0, 1}; // a.n is a union
10 assign out0 = a.n.i;
11
12 endmodule
```

SV2.6

Message

Non-blocking assignment to automatic variables is illegal

Default Severity

Error

Description

The automatic keyword cannot be used in nonblocking assignments. Variables declared in automatic tasks/functions are deallocated at the end of the task/function invocation, and are not assigned values using nonblocking assignments or procedural continuous assignments.

Example

In line 3 and 4, the variable out2, S1, and S2 are declared in an automatic task, the non-blocking assignment in line 5, 6, and 7 for S1, S2 and out2 are illegal.

```
1 module top(input clk,input [7:0] in1,output reg [7:0] out1);
2
3 task automatic task1(input [7:0] in2, output reg [7:0] out2);
4 struct packed {reg [7:0] r1;} S1,S2;
5 S1 <= in2;
6 S2 <= S1;
7 {out1[7:3], out2[3:0]} <= S2;
8 endtask
9
10 always @(posedge clk)
11 begin
12   task1(in1, out1);
13 end
14 endmodule
```

SV2.7

Message

```
Automatic variable used in a non-procedural context
```

Default Severity

Error

Description

The `automatic` keyword cannot be used in a data declaration that is outside of a procedural context.

Example

In line 3, the variable `reg2` is declared as an automatic variable within a module scope. You cannot use the automatic keyword in a non-procedural context.

```
1 module variable1 (input bit [1:0] in1, output bit out1);
2
3 automatic bit reg1;
4
5 always @(in1)
6 begin
7    reg1 = in1[0] ^ in1[1];
8    out1 = ~reg1;
9 end
10 endmodule
```

SV2.8

Message

Undeclared member reference of a struct/union type

Default Severity

Error

Description

A reference is made to a non-existing member of a structure or union.

Example

The following triggers this rule message because the assignment in line 13 refers to f5, which does not exist in the structure variable 'tp' (of type 'struct t1').

```
1  typedef struct packed {
2    bit  [2:0] f1;
3    bit  [2:0] f2;
4    bit  [2:0] f3;
5  } t1;
6
7  module test(in, out);
8  input [2:0] in;
9  output [8:0] out;
10
11  t1 tp;
12
13  assign tp.f5[0] = in; // error tp.f5 undefined
14  assign out = tp;
15
16  endmodule
```

SV2.9

Message

Identifier is referenced before declaration

Default Severity

Warning

Description

A variable reference is made before it is declared.

Example

The following triggers this rule message because the hierarchical reference on line 10 refers to var1 with range and sub-element, which does not be declared before it is used.

```
1  typedef struct packed {
2    logic m1;
3    logic [3:0] m2;
4  } udt;
5
6  module top(input logic ena, input [3:9] data_in, output logic [3:0] data_out);
7
8  always @(ena or data_in) begin
9    data_out = 4'b0000;
10    if (var1[1].m1)
11       data_out = var1[1].m2[3:0];
12 end
13
14 udt [1:0] var1;
15 assign var1[1].m1 = ena;
16 assign var1[1].m2[3:0] = data_in;
17
18 endmodule
```

SV3.1

Message

```
Modules can neither be declared nor instantiated in interfaces
```

Default Severity

Error

Description

Interfaces can be declared and instantiated in modules (either flat or hierarchical), but modules can neither be declared nor instantiated in interfaces.

Example

In the following example, the module instantiation is not allowed in the interface declaration:

```
module mod1 (input aa, bb, output zz);
  assign zz = aa && bb;
endmodule

interface simple_bus; // Define the interface
  logic req, gnt;
  logic [7:0] addr, data;
  logic [1:0] mode;
    logic start, rdy;
    mod1 u0(req, gnt, rdy); // ERROR: SV3.1 violation.
endinterface: simple_bus
```

SV3.2

Message

`Unresolved SystemVerilog interface port`

Default Severity

Error

Description

An interface port type was specified, but does not exist.

Example

In the following example, a port called slav was specified, but does not exist.

```
module memMod (simple_bus.slav a); // typo; modport name is "slave".
  logic avail;
  always @(posedge a.clk)
    a.gnt <= a.req & avail;
endmodule
module cpuMod (simple_bus.master b);
  always @(posedge b.clk)
    b.req <= b.gnt & b.rdy;
endmodule

interface simple_bus (input bit clk, inout logic gnt, req, rdy);
  logic [7:0] addr, data;
  logic [1:0] mode;
  logic start ;
  modport slave (input req, addr, mode, start, clk,
                 output gnt, rdy, ref data);
  modport master(input gnt, rdy, clk,
                 output req, addr, mode, start, ref data);
endinterface: simple_bus
```

SV3.3

Message

```
Accessing an object that is not fully declared in the modport declaration
```

Default Severity

Warning

Description

Accessing one or more objects that are declared in the interface, but not fully declared in the corresponding modport declaration.

Example

In the following example, module "`CPU`" has a generic interface port bound to the interface modport called "`master`" and accesses the interface object "x" through the task "`master_write`" (line 27) and accesses the less object through task "`is_ge`".

This triggers SV3.3 because the declaration of "`master`"(line 5) does not declare "`x`" or "`less`" in its input/output port declarations.

```
1  interface bus_if (input logic rst, clk);
2      logic   [1:0]     x, y;
3      logic             less;
5      modport master(input rst,clk, import master_write, is_ge);
6      modport slave(input x, output y, less);
7
8      function void master_write(input logic [1:0] a);
9          x = a;
10     endfunction
12     function logic is_ge();
13         return !less;
14     endfunction
15 endinterface : bus_if
17 module CPU (interface i_bus_if);
18     logic   [1:0]   op1;
19     always@(negedge i_bus_if.rst or posedge i_bus_if.clk) begin
20         if (!i_bus_if.rst)
21             op1 <= 2'b0;
22         else
23             op1 <= i_bus_if.is_ge() ? 2'b0 : (op1+1'b1);
24     end
26     always @(op1)
27         i_bus_if.master_write(.a(op1));
```

```
endmodule : CPU
module ALU (interface i);
    always@(i.x) begin
        i.y = i.x ^ 2'b11;
        i.less = (i.x < i.y);
    end
endmodule : ALU

module top(input clk, rst, output [1:0] x, y, output less);
    bus_if      i(.*);
    CPU         cpu(.i_bus_if(i.master));
    ALU         alu(.i(i.slave));
    assign  less = i.less;
    assign  x = i.x;
    assign  y = i.y;
endmodule : top
```

SV3.3a

Message

```
SystemVerilog interface modport port of type output or inout may introduce multiple
     drivers
```

Default Severity

Warning

Description

The modport declaration of an interface contains a port expression with port which is also specified in the port list of the interface. This may introduce multiple drivers if the modport is used in a submodule(s).

Example

In the example below on line 4, the modport 'mp' contains port 'q' which is an output port of the interface 'myifc'

```
1 interface myifc(input en, output [3:0] q);
2      reg [3:0] a, b, c;
3      assgn q = en? a : b;
4      modort mp(input a, b, en, output c, q); // <= SV3.3a
5  endinteface
6
7  module mid(myifc.mp ifcp);
8      assign ifcp.c = ifcp.a;
9      assign ifcp.q = ifcp.en? ifcp.a : ifcp.b;
10 endmodule
11
12 module test(input en, input [3:0] a, b, output [3:0] q1, q2);
13     wire [3:0] q;
14     myifc myifc(.en(en), .q(q));
15     assign myifc.a = a;
16     assign myifc.b = b;
17     assign q2 = myifc.c;
18
19     assign q1 = q;
20     mid mid(myifc.mp);
21 endmodule
```

SV3.3b

Message

```
SystemVerilog interface port of type input or inout may introduce multiple drivers
```

Default Severity

Warning

Description

The input or inout port of an interface is driven or assigned from within the interface.

Example

In the following example, 'en' is an input port in interface 'myifc'. The assignment to 'en' on line 4 can potentially create multiple drivers. Conformal will report rule-check message SV3.3b.

```
1 interface myifc(input en, output [3:0] q);
2      reg [3:0] a, b, c, q;
3      task write();
4           en= 1'b0;
5      endtask
6    modport mp(input a, b, en, output c, q);
7 endinteface
```

SV3.3c

Message

```
Reference to port not declared in interface modport is illegal
```

Default Severity

Warning

Description

The accessed interface variable is not specified as a port in the modport declaration.

Example

In the example below, access to 'ifcp.q' on line 8 is illegal since 'q' is not listed in the port list of modport 'mp' of interface 'myifc' (on line 3)

```
1  interface myifc(input reg en);
2       reg [3:0] a, b, c, q;
3       modport mp(input a, b, en, output c);
4  endinterface
5
6  module mid(myifc.mp ifcp);
7       assign ifcp.c = ifcp.a;
8       assign ifcp.q = ifcp.en? ifcp.a : ifcp.b; // <= SV3.3c
9  endmodule
10
11  module test(input en, input [3:0] a, b, output [3:0] q);
12       wire [3:0] q;
13       myifc myifc(.en(en));
14       assign myifc.a = a;
15       assign myifc.b = b;
16       assign q  = myifc.q;
17       mid mid(myifc.mp);
18  endmodule
```

SV3.3d

Message

```
Connections to interface port with no modport specified; connections are of type
     inout
```

Default Severity

Note

Description

An instantiated module has system-verilog interface port with no modport specified in either the interface port declaration, or in the connection expression.

Example

In the example below, in instantiation of module '`sub`' on line 12, no modport is specified either in the instantiation statement or in the port declaration of sv-interface port '`my_if`' on line 6.

```
 1  interface my_if;
 2     logic [7:0] var1, var2, var3;
 3     modport my_mp ( input  var1, input  var2, output var3);
 4  endinterface
 5
 6  module sub (my_if my_if);
 7     assign my_if.var3 = my_if.var1 & my_if.var2;
 8  endmodule
 9
10  module test (input [7:0]  in1, input [7:0]  in2, output out);
11     my_if my_if();
12     sub sub (my_if);
13     assign my_if.var1 = in1;
14     assign my_if.var2 = in2;
15     assign out = |my_if.var3;
16  endmodule
```

SV3.4

Message

```
Zero replication in concatenation is considered to have a size of zero and is
     ignored (SystemVerilog 2009 or later)
```

Default Severity

Note

Description

A replication multiplier is a constant 0 but the entire concatenation remains valid. The replicated concatenation portion is treated as NULL as if the portion is removed from the entire concatenation.

Note that for Verilog or SystemVerilog 2005, VLG3.4 is issued instead of SV3.4.

Example

In the following example on line 6, replication `2'b0` cause the expression `'{2'b0{2'b1}}'` to evaluate to NULL and be ignored, Hence, the concat expression `'{{2'b0{2'b1}}, 1'b0}'` evaluates to `1'b0`.

```
1 module test (a, q);
2   input [3:0] a;
3   output [4:0] q;
4   reg [4:0] q;
5    always @ (a ) begin
6     q = {{2'b0{2'b1}}, 1'b0};
7    end
8 endmodule
```

SV3.5

Message

```
Replication operator is not allowed in unpacked array concatenation
```

Default Severity

Warning

Description

The unpacked array concatenation contains replication operator. Typically the issue can be resolved by replacing unpacked array concatenation with assignment pattern.

Example

In the following example, the right hand side expression on line 2 consists of replications. The standard coding is to use "{8'b1111_1111, 8'b0000_0000}" or "'{{8{1'b1}}, {8{1'b0}}}" instead:

```
1    module test(output byte out [2]);
2        assign out = {{8{1'b1}}, {8{1'b0}}};
3    endmodule
```

SV4.1

Message

`SystemVerilog action block of assertion ignored`

Default Severity

Warning

Description

IEEE 1800-2005 LRM has defined the `assert_property_statement` syntax as `assert_property_statement::=`

```
assert property ( property_spec ) action_block
```

The Conformal software reads in the statement and only converts the following portion:

```
assert property ( property_spec)
```

The `action_block` information is read-in and ignored.

Example

In the following example, action blocks are ignored on lines 6 and 8:

```
1 module test(input clk, req, ack, reset_n, output reg err);
2   property pa1;
3      @(posedge clk) !reset_n || !req |-> !ack;
4   endproperty
5   assert_ack1:assert property (pa1)
6                          err=1'b0;
7                       else
8                          err=1'b1;
9 endmodule
```

SV4.2a

Message

`Event expression in property/sequence instantiation argument list is unsupported`

Default Severity

Warning

Description

The event expression cannot be used as argument to a property or sequence instance.

Example

In the following example, on line 7, the event expression `posedge clk` cannot be used as `expression` which is required by the `list_of_argument` to the property instance `never` :

```
1 property never(prop, clk, rst);
2    @(clk) disable iff(rst) not(prop);
3 endproperty : never
4
5 module test(input a, clk, rst);
6 ...
7    assert property (never(~a, posedge clk, rst));
8 ...
9 endmodule
```

SV4.2b

Message

`Property expression in property/sequence instantiation argument list is unsupported`

Default Severity

Warning

Description

The property expression cannot be used as argument to a property or sequence instance.

Example

In the following example, on line 11, the property expression `seq_expr` cannot be used as `expression` which is required by the `list_of_argument` to the property instance `never`:

```
1 module test(input a, b, clk, rst);
2
3 property never(prop, clk, rst);
4    @(clk) disable iff(rst) not(prop);
5 endproperty : never
6
7 sequence seq_expr;
8     a ##1 b;
9 endsequence
10 ...
11    assert property(never(seq_expr, clk, rst));
12 ...
13 endmodule
```

SV5.1

Message

```
Undefined module is instantiated with implicit .* port connection. Its instance
     ports are treated as unconnected.
```

Default Severity

Warning

Description

There is an undefined module instantiation with an implicit port connection (.*) style. The tool treats the instance port connection as unconnected.

Example

In the following example, on line 2, there is an instance statement with implicit port connection.

```
1 module aa(input in1, input in2, output out1, output out2    );
2    cc u3 ( .*);
3 endmodule
```

If the referenced module '`cc`' is undefined and "`set undefined cell black_box`" is used, the tool generates the netlist as follows:

```
module cc;
// module is bboxed.
endmodule

module aa(in1, in2, out1, out2);
input  in1, in2;
output out1, out2;
wire  out2, out1, in2, in1;
  cc u3();
endmodule
```

SV5.3

Message

```
Bind statement target instance or scope not found and is ignored
```

Default Severity

Warning

Description

The bind statement specifies a bind target (instance or scope) that does not exist in the module hierarchy.

Example

In the example below, module 'ptest2' specified as target of the bind statement on line 2 does not exist in the hierarchy rooted at module 'test'. Conformal will issue SV5.3 rule check message and the bind statement will be ignored.

```
1 // File: bind.sv
2 bind ptest2 ptest_bind u_ptest_bind(clk, set, rst);
3
4 module ptest_bind(input clk, set, rst);
5   always @(set or rst)
6     assert (!((set != 1’b0) && (rst != 1’b0)));
7 endmodule
```

```
1 // File: test.sv
2 module test(input clk, set, rst, d, output q);
3   ptest1 u1(.*);
4 endmodule
5
6 module ptest1(input clk, set, rst, d, output q);
7   asrff asrff(.*);
8 endmodule
```

SV6.1a

Message

```
SystemVerilog elaboration system task '$info' is used and active
```

Default Severity

Note

Description

The '$info' system task is used and processed during elaboration.

Example

```
1 module m();
2 parameter FOO=-1;
3 if (FOO<0 || FOO>10) $info("parameter FOO has an illegal value");
4 endmodule
SV6.1a: SystemVerilog elaboration system task '$info' is used and active
Type: Golden      Severity: Note       Occurrence: 1
1: m
on line 3 in file 'test.sv'
```

SV6.1b

Message

`SystemVerilog elaboration system task '$warning' is used and active`

Default Severity

Warning

Description

The '$warning' system task is used and processed during elaboration.

Example

```
1 module m();
2 parameter FOO=-1;
3 if (FOO<0 || FOO>10) $warning("parameter FOO has an illegal value");
4 endmodule
SV6.1b: SystemVerilog elaboration system task '$warning' is used and active
Type: Golden      Severity: Warning       Occurrence: 1
1: m
on line 3 in file 'test.sv'
```

SV6.1c

Message

```
SystemVerilog elaboration system task '$error' is used and active
```

Default Severity

Error

Description

The '$error' system task is used and processed during elaboration.

Example

```
1 module m();
2 parameter FOO=-1;
3 if (FOO<0 || FOO>10) $error("parameter FOO has an illegal value");
4 endmodule
SV6.1c: SystemVerilog elaboration system task '$error' is used and active
Type: Golden      Severity: Error       Occurrence: 1
1: m
on line 3 in file 'test.sv'
```

SV6.1d

Message

`SystemVerilog elaboration system task '$fatal' is used and active`

Default Severity

Error

Description

The '$fatal' system task is used and processed during elaboration.

Example

```
1 module m();
2 parameter FOO=-1;
3 if (FOO<0 || FOO>10) $fatal(2, "parameter FOO has an illegal value");
4 endmodule
SV6.1d: SystemVerilog elaboration system task '$fatal' is used and active
Type: Golden      Severity: Error       Occurrence: 1
1: m
on line 3 in file 'test.sv'
```

SV7.1

Message

`SystemVerilog inside expression range with X/Z value`

Default Severity

Error

Description

Indicates that the resulting range of the inside expression is empty. The expression is always false.

Example

In the following example, the range expression on line 4 has x/z value in it.

```
module test(input signed [3:0] in, output reg [1:0] out);
  always @*
    begin
      if (in inside {[4'hz:4'h3]})
      out = 2'b00;
      else
      out = 2'b11;
    end
endmodule
```

SV8.1

Message

`Package imported`

Default Severity

Note

Description

The SystemVerilog package is imported to current scope

Example

In the following example, Conformal issues a note to indicate the package is imported.

```
package pkg;
  parameter int W = 1;
endpackage

module top();
  import pkg::*;
endmodule
```

SV8.2

Message

```
Package with multiple definitions imported
```

Default Severity

Note

Description

The rule indicates there are multiple definitions of a SystemVerilog package, and which library is used for the package importing.

Example

In the following example, the package file pkg.v is read to lib_1 and lib_2 respectively, and the design module top has import pkg::*. Conformal issues the rule to indicate which library package is used, and what library has the multiple package definition.

```
// pkg.v
package pkg;
  parameter int W = 1;
endpackage

// top.v
module top();
  import pkg::*;
endmodule


TCL_SETUP> read_design -noelab -sv -mapfile lib_1 pkg.v
TCL_SETUP> read_design -noelab -sv -mapfile lib_2 pkg.v
TCL_SETUP> read_design -sv top.v
```

```
SV8.2: Package with multiple definitions imported
       Type: Golden      Severity: Note       Occurrence: 1
       1: top: import package pkg from library lib_2. pkg
       also defined in lib_1
          on line 3 in file 'top.v'
```

SV8.3

Message

```
Package definition not parsed into current design library
```

Default Severity

Note

Description

The rule indicates the imported package is not parsed into current design library.

Example

In the following example, the package file pkg.v is read to lib_1 and lib_2 respectively, and the design module top has import pkg::*. Conformal issues the rule to indicate the package "pkg" is not parsed into "rtl" library, and uses the package found in lib_2.

```
// pkg.v
package pkg;
  parameter int W = 1;
endpackage

// top.v
module top();
  import pkg::*;
endmodule


TCL_SETUP> read_design -noelab -sv -mapfile lib_1 pkg.v
TCL_SETUP> read_design -noelab -sv -mapfile lib_2 pkg.v
TCL_SETUP> read_design -sv -mapfile rtl top.v
```

```
SV8.3: Package definition not parsed into current design library
       Type: Golden      Severity: Note       Occurrence: 1
       1: package pkg not parsed into library rtl. Used alternative
       package from library lib_2
          on line 2 in file 'top.v'
```

User-Defined Primitive

This category of rules applies to designs that include user-defined primitives. The following lists the User-Defined Primitive (UDP) rule checks.

UDP1.1

Message

`Swapped set and reset of DFF/DLAT`

Default Severity

Note

Description

Conformal swapped a set and reset for a user-defined DFF or DLAT primitive. Automatic swapping of DFF and DLAT set and reset occurs only when there is a need to accommodate a conversion (for example, merging DFFs).

Example

Two warnings occurred for the following example:

1. On lines 5 through 8, instance `U1` is merged into instance `U2` because they have the same connection (except `q` and `qb` for outputs).
2. Conformal swaps set and reset of instance `U2` to accommodate merging instance `U1`.

```
module RSLTA (Q,QB,S,R,notifier);
        input   S,R;
        output  Q,QB;
        input     notifier;
        rsltaq  U1(.q(Q),.s(S),.r(R),
        .notify(notifier));
        rsltaqb U2(.qb(QB),.s(S),.r(R),
        .notify(notifier));
endmodule
```

UDP1.2

Message

`Removed set-domination logic of DFF/DLAT`

Default Severity

Note

Description

One or more user-defined DFF or DLAT primitives have multiple sets of set-dominant logic. Conformal removed the redundant set of set-dominant logic.

Example

On line 11 of the following example, the design includes an OR logic to ensure a set-dominant DFF, but the `DESFQ` primitive was already defined in line 1 as a set-dominant DFF. Thus, the design includes two set-dominant logics (shown in lines 1 and 11).

```
primitive DESFQ = a set-dominant DFF
module test ( N01, H01, H02, H03, H04);
    input H01;
    input H02;
    input H03;
    input H04;
    output N01;
    not ( _G005, H03 );
    not ( _G002, H04 );
    or  ( N01, _G008, H04 );
    DESFQ ( .Q(_G008), .D(H01), .CP(H02),
       .RB(_G005), .SB(_G002), .notifier(notifier) );
endmodule
```

UDP1.3

Message

```
Unknown set and reset domination. Implemented as reset domination
```

Default Severity

Warning

Description

A UDP is interpreted as a sequential DLAT or DFF logic, and:

- DLAT or DFF has both asynchronous set and reset logic
- set and reset logic might result in 1 at the same time
- no domination on set or reset

Example

```
primitive srq0 ( Q, S, RX );
output Q; reg Q;
input S;
input RX;
  table
    //S   RX   :   Q-  :   Q+
    1   1    :   ?   :   1   ;
    0   0    :   ?   :   0   ;
    ?   1    :   1   :   1   ;
    0   ?    :   0   :   0   ;
    0   1    :   ?   :   -   ;
endtable
endprimitive
```

The Conformal software will generate a DLAT circuit with asynchronous set and reset:

- set = S
- reset = RX'
- d = 0
- clock = 0

The UDP does not specify the behavior when S = 1 and RX = 0. The Conformal software implements the circuit as a reset dominated logic, which implies that when S = 1 and RX = 0, Q = 0. This is usually caused by UDP table incompletion.

UDP2.1

Message

```
Inverter on data input of DFF/DLAT is moved to output
```

Default Severity

Warning

Description

One or more user-defined DFF or DLAT primitives (UDP) include an inverter on the data port. By default, Conformal relocates inverters from data port to output port.

Example

In the following example, primitive `dff_simple` has an inverter at the data port of the DFF, but Conformal moves it to the output port of the DFF.

```
primitive dff_simple(Q, S, R, CK, D);
output Q;
input  S, R, CK, D;
reg    Q;

  table
    1  0   ?    ?  :  ?  :  1;
    ?  1   ?    ?  :  ?  :  0;
    0  0   r    0  :  ?  :  1;
    0  0   r    1  :  ?  :  0;
    0  0   (?0) ?  :  ?  :  -;
  endtable
endprimitive
```

UDP2.2

Message

```
Inverter on data input of DFF/DLAT is not moved to output
```

Default Severity

Warning

Description

One or more user-defined DFF or DLAT primitives (UDP) include an inverter on the data port. By default, Conformal relocates inverters from data port to output port. However, in this case, you specified that Conformal will not move the DFF or DLAT when you used the following command:

```
add udp model dff_simple -nomove_inverter
```

Example

In this example, primitive `dff_simple` has an inverter at the data port of the DFF, but Conformal did not move it since the user applied the `-nomove_inverter` option.

```
primitive dff_simple(Q, S, R, CK, D);
output Q;
input  S, R, CK, D;
reg    Q;
  table
    1  0   ?    ?  :  ?  :  1;
    ?  1   ?    ?  :  ?  :  0;
    0  0   r    0  :  ?  :  1;
    0  0   r    1  :  ?  :  0;
    0  0  (?0)  ?  :  ?  :  -;
  endtable
endprimitive
```

Input:

```
SETUP>add udp model dff_simple -nomove_inverter
SETUP>read design test.v -verilog
```

UDP3

Message

```
Merged redundant user-defined DFF/DLAT primitive(s)
```

Default Severity

Note

Description

The design includes one or more redundant user-defined DFF or DLAT primitives. Conformal merges all redundant user-defined DFF and DLAT primitives.

Example

In the following example, DFF `UDP_A` is redundant and merged into DFF `UDP_B`. See lines 7 through 10.

```
module test ( N01, N02, H01, H02, notifier );
  input H01;
  input H02;
  input notifier;
  output N01;
  output N02;
  DESFQ UDP_A(.Q(N01), .D(H01), .CP(H02), .RB(1'b1),
   .SB(1'b1), .notifier(notifier));
  DESFQ UDP_B(.Q(N02), .D(H01), .CP(H02),
   .RB(1'b1), .SB(1'b1), .notifier(notifier));
endmodule
```

UDP3.1

Message

```
Conflicting entries are detected in the outputs
```

Default Severity

Warning

Description

There are conflict entries in the user-defined primitive outputs.

Example

In the following example, the two lines have the same input values but different output values:

```
//      D      CK     RB      SB      FLAG :    Qt :     Qt+1
        ?      ?       0       0         ? :     ? :        1;//
        ?      ?       0       0         ? :     ? :        0;//
```

UDP3.2

Message

`Primitive has unspecified term(s)`

Default Severity

Warning

Description

There are unspecified term(s) in the UDP table.

Example

In the following example, term '010' and '011' is unspecified:

```
primitive udp_inv_clr0 (qn, clr, pre, inp);
        output qn;
        input  clr, pre, inp;
        table
        //    clr    pre   inp   : qn
              0      0     ?     : 0;
              1      ?     0     : 1;
              1      ?     1     : 0;
              // ?   1     0     : 1;
              // ?   1     1     : 0;
              x      x     1     : 0;
              x      x     0     : 1;
        endtable
endprimitive
```

UDP4.1

Message

```
Primitive contains invalid 'z' symbol
```

Default Severity

Warning

Description

The UDP contains an entry that has an invalid symbol 'z'.

Example

In the following example, because the entry in line 11 has symbol 'z', `mydff` will be blackboxed:

```
primitive mydff (qq, clk, dd, ss, rr);
output qq; reg qq;
input clk, dd, ss, rr;
table
// clk dd ss rr : qq : qq+
    r   0  0  0 :  ? : 0 ;
    r   1  0  0 :  ? : 1 ;
    f   ?  0  0 :  ? : - ;
    ?   ?  1  0 :  ? : 1 ;
    ?   ?  0  1 :  ? : 0 ;
    ?   ?  1  1 :  ? : z ;
endtable
endprimitive
```

UDP4.2

Message

```
Primitive contains an 'x' output without an 'x' input
```

Default Severity

Warning

Description

The UDP contains an entry that has an 'x' value output without an 'x' value input.

Example

In the following example, the last line of the table has 'x' as an output, and its input does not contain 'x':

```
primitive mydff (qq, clk, dd, ss, rr);
output qq; reg qq;
input clk, dd, ss, rr;

table
// clk  dd  ss  rr  :   qq  :   qq+
    r   0   0   0   :   ?   :   0   ;
    r   1   0   0   :   ?   :   1   ;
    f   ?   0   0   :   ?   :   -   ;
    ?   ?   1   0   :   ?   :   1   ;
    ?   ?   0   1   :   ?   :   0   ;
    ?   ?   1   1   :   ?   :   x   ;
endtable
endprimitive
```

Verilog

This category of rules applies to designs that are written in Verilog language. RTL rules apply to Verilog designs *and* VHDL designs (see “Register Transfer Level” on page 219).

The following lists the Verilog (VLG) rule checks.

VLG1.1

Message

`Case inequality operators are treated as inequality operators`

Default Severity

Warning

Description

The design includes one or more case inequality operators that Conformal treats as logical inequality operators.

Inequality operators syntax:

```
!== is a case inequality operator.
!=  is a logical inequality operator.
```

Example

In the following example, Conformal treats the case inequality operator as a logical inequality operator. See line 7.

```
1 module VLGT (clk,cond,in0,in1,out0);
2 input clk,in0,in1;
3 input [1:0] cond;
4 output out0;
5 reg out0;
6 always @(posedge clk) begin
7   if (cond[0] !== cond[1])
8     out0 <= in0;
9   else
10    out0 <= in1;
11 end
12 endmodule
```

VLG1.2

Message

`Case equality operators are treated as equality operators`

Default Severity

Warning

Description

The design includes one or more case equality operators that Conformal treats as logical equality operators.

Equality operators syntax:

```
=== is a case equality operator.
==  is a logical equality operator.
```

Example

In the following example, Conformal treats the case equality operator as a logical equality operator. See line 7.

```
1 module VLGT (clk,cond,in0,in1,out0);
2 input clk,in0,in1;
3 input [1:0] cond;
4 output out0;
5 reg out0;
6 always @(posedge clk) begin
7   if (cond[0] === cond[1])
8     out0 <= in0;
9   else
10    out0 <= in1;
11 end
12 endmodule
```

VLG1.3

Message

`Wild inequality is treated as inequality`

Default Severity

Warning

Description

Conformal does not support the `!=?` SystemVerilog operator with the exact semantics supported in simulation. Conformal treats `!=?` as `!=`.

Example

In the following example, Conformal encounters `!?=` on line 4 and treats it as `!=`. Similarly, on line 5, Conformal treats `!==` (case inequality) as `!=`. (Refer to VLG1.1 on page 624.)

```
1 module test(aa, bb, o1, o2);
2 input aa, bb;
3 output o1, o2;
4   assign o1 = aa !=? bb;
5   assign o2 = aa !== bb;
6 endmodule
```

VLG1.4

Message

`Wild equality is treated as equality`

Default Severity

Warning

Description

Conformal does not support the `==?` SystemVerilog operator with the exact semantics supported in simulation. Conformal treats `==?` as `==`.

Example

In the following example, Conformal encounters `==?` on line 4 and treats it as `==`. Similarly, on line 5, Conformal treats `===` (case equality) as `==`. (Refer to VLG1.2 on page 625.)

```
1 module test(aa, bb, o1, o2);
2 input aa, bb;
3 output o1, o2;
4   assign o1 = aa ==? bb;
5   assign o2 = aa === bb;
6 endmodule
```

VLG1.5

Message

`Arithmetic shift operator is not supported in Verilog 1995`

Default Severity

Error

Description

The arithmetic shift operators `<<<` (shift left) and `>>>` (shift right) are not part of the Verilog 1995 standard. The software supports Verilog 1995 as default design input language.

Use the `READ DESIGN` command's `-verilog2k` or `-systemverilog` options to enable the software to support the `<<<` and `>>>` operators.

Example

In the following example, the arithmetic shift operator `>>>` (shift right) is not supported in Verilog 1995 on line 5:

```
1 module test(aa, cnt, oo);
2 input [7:0] aa;
3 input [2:0] cnt;
4 output [7:0] oo;
5 assign oo = aa >>> cnt;
6 endmodule
```

VLG2.2

Message

`Non-binary case item is used in casex/casez statement`

Default Severity

Note

Description

The design includes one or more case_items that use a non-binary value in a casex or casez statement.

Note: this rule is obsoleted and is being replace with VLG2.2a for casex and VLG2.2b for casez statement.

Example `casex` statement syntax:

```
case  (case_expression)
       case_item1 : case_item_statement1;
       case_item2 : case_item_statement2;
       default    : case_item_statement3;
endcase
```

Example

In the following example, the design uses an `X` value for the second case_item. See line 9.

```
1 module test ( a, b, c, sel, out0);
2 input  a, b, c;
3 input  sel;
4 output out0;
5 reg out0;
6   always @(sel or a or b or c) begin
7     casex(sel)
8       1'b1   : out0 = a;
9       1'bx   : out0 = b;
10      default: out0 = c;
11    endcase
12   end
13 endmodule
```

VLG2.2a

Message

`Non-binary case item is used in casex statement`

Default Severity

Note

Description

The design includes one or more case_items that use a non-binary value in a casex or casez statement.

Example `casex` statement syntax:

```
casex  (case_expression)
       case_item1 : case_item_statement1;
       case_item2 : case_item_statement2;
       default    : case_item_statement3;
endcase
```

Example

In the following example, the design uses an `X` value for the second case_item. See line 9.

```
1 module test ( a, b, c, sel, out0);
2 input  a, b, c;
3 input  sel;
4 output out0;
5 reg out0;
6   always @(sel or a or b or c) begin
7     casex(sel)
8       1'b1   : out0 = a;
9       1'bx   : out0 = b;
10      default: out0 = c;
11    endcase
12   end
13 endmodule
```

VLG2.2b

Message

```
Non-binary case item is used in casez statement
```

Default Severity

Note

Description

The design includes one or more case_items that use a non-binary value in a casex or casez statement.

Example `casex` statement syntax:

```
casez  (case_expression)
       case_item1 : case_item_statement1;
       case_item2 : case_item_statement2;
       default    : case_item_statement3;
endcase
```

Example

In the following example, the design uses a `z` value for the second case_item. See line 9.

```
1 module test ( a, b, c, sel, out0);
2 input  a, b, c;
3 input  sel;
4 output out0;
5 reg out0;
6   always @(sel or a or b or c) begin
7     casez(sel)
8       1'b1   : out0 = a;
9       1'bz   : out0 = b;
10      default: out0 = c;
11    endcase
12   end
13 endmodule
```

VLG2.3

Message

`default keyword is not the last item of case statement(s)`

Default Severity

Warning

Description

The `default` keyword is not the last case_item in one or more case statements.

Sample `case` statement syntax:

```
case  (case_expression)
    case_item1 :  case_item_statement1;
    case_item2 :  case_item_statement2;
    default    :  case_item_statement3;
endcase
```

Example

In the following example, the keyword `default` is not the last case_item. See line 9.

```
1 module test ( a, b, c, sel, out0);
2 input  a, b, c;
3 input  sel;
4 output out0;
5 reg out0;
6   always @(sel or a or b or c) begin
7     casex(sel)
8       1'b1   :  out0 = a;
9       default:  out0 = c;
10      1'b0   :  out0 = b;
11    endcase
12  end
13 endmodule
```

VLG2.4

Message

```
Casex/casez statement uses a non-xz case expression and supported case_items
     expression
```

Default Severity

Note

Description

The design includes one or more `case_items` that use non-xz case expressions in a `casex` or `casez` statement.

Example

In the following example, the design uses non-xz case expression in a casex statement (see line 11).

```
1 module test ( a, b, c, sel, out0);
2 input a, b, c;
3 input sel;
4 output out0;
5 reg out0;
6 wire w;
7 assign w = 1'bx;
8 always @(sel or a or b or c) begin
9 casex(sel)
10 1'b1 : out0 = a;
11 a|w : out0 = b;
12 default: out0 = c;
13 endcase
14 end
15 endmodule
```

VLG3.1

Message

`Unsized constant(s) with leading X/Z value is extended beyond 32 bits`

Default Severity

Warning

Description

One or more unsized constants with a leading X/Z value are extend beyond 32 bits.

Example

In the following example, the design assigns an unsized constant with leading `X` value `'bx` to a 34-bit output `out0` (see line 5).

```
1 module test(out0,y,in);
2 output [33:0] out0;
3 output y;
4 input in;
5 assign out0 = 'bx;
6 assign y = in;
7 endmodule
```

VLG3.2

Message

`Verilog event expression in always block(s) are complex`

Default Severity

Warning

Description

The Verilog event expressions used in one or more *always* blocks are complex.

Example

In the following example, event expression `a|b` is considered complex (see line 5).

```
1 module test (a, b, q);
2 input [3:0] a, b;
3 output [3:0] q;
4 reg [3:0] q;
5 always @ (a | b ) begin
6   q = a & b;
7 end
8 endmodule
```

VLG3.3

Message

`Replication is not a positive constant or it contains X or Z values`

Default Severity

Warning

Description

A replication multiplier is not a constant or it contains X or Z values.

Note: This check has a warning default severity level, but the module is blackboxed.

Example

In the following example, replication `2'bx` is illegal value (see line 6).

```
1 module test (a, q);
2 input [3:0] a;
3 output [4:0] q;
4 reg [4:0] q;
5 always @ (a ) begin
6   q = {{2'b1x{1'b1}}, 1'b0};
7 end
8 endmodule
```

VLG3.4

Message

`Zero replication is treated as NULL in concatenation`

Default Severity

Warning

Description

A replication multiplier is a constant 0 but the entire concatenation remains valid. The replicated concatenation portion is treated as NULL as if the portion is removed from the entire concatenation.

Note that for SystemVerilog 2009 and later, SV3.4 is issued instead of VLG3.4.

Example

In the following example, replication `2'b0` is illegal value in Verilog or SystemVerilog 2004 (see line 6).

```
1 module test (a, q);
2 input [3:0] a;
3 output [4:0] q;
4 reg [4:0] q;
5 always @ (a ) begin
6   q = {{2'b0{2'b1}}, 1'b0};
7 end
8 endmodule
```

VLG3.5

Message

`Null expression is not allowed (caused by zero replication)`

Default Severity

Warning

Description

A replication multiplier is a constant 0 and the entire concatenation becomes NULL. The replicated concatenation portion is treated as NULL as if the portion is removed from the entire concatenation.

Note: This check has a warning default severity level, but the module is blackboxed.

Example

In the following example, on line 6 the replication multiplier is a constant 0 and the entire concatenation becomes NULL, which is not allowed.

```
1 module test (a, q);
2 input [3:0] a;
3 output [4:0] q;
4 reg [4:0] q;
5 always @ (a ) begin
6   q = {2'b0{2'b1}};
7 end
8 endmodule
```

VLG3.5a

Message

`Null operand is used`

Default Severity

Warning

Description

Indicates that a zero replication expression (where the replication multiplier is a constant zero) will be treated as a null expression.

Example

In line 6 of the following example, LEC treats {1'b0{1'b1}} as a NULL expression:

```
1    module test (a, q);
2     input [3:0] a;
3     output [4:0] q;
4     reg [4:0] q;
5     always @ (a ) begin
6       q = {{1'b0{1'b1}}+1'b1};
7     end
8     endmodule
```

VLG3.6

Message

`Negative replication is not allowed`

Default Severity

Warning

Description

A replication multiplier is a negative integer constant.

Note: This check has a warning default severity level, but the module is blackboxed. With `set rule handling -ignore VLG3.6`, this rule will be ignored, and the expression of negative replication will be replaced by 1-bit wide zero.

Example

In the following example, replication `-1` is not allowed (see line 6).

```
1 module test (a, q);
2 input [3:0] a;
3 output [4:0] q;
4 reg [4:0] q;
5 always @ (a ) begin
6   q = {{-1{2'b1}}, 4'b1};
7 end
8 endmodule
```

VLG3.7

Message

`Unsized constant value is not allowed in concatenation`

Default Severity

Error

Description

Unsized constant numbers are not allowed since the size of each operand is required to calculate the complete size of the concatenation.

Example

The following example triggers the VLG3.7 message because `1` is an unsized constant value. See line 6.

```
1 module test (a, q);
2 input [3:0] a;
3 output [4:0] q;
4 reg [4:0] q;
5 always @ (a) begin
6     q = {1, 4'b1};
7 end
8 endmodule
```

VLG4.1

Message

`Wire type wand is used`

Default Severity

Warning

Description

The design includes a user-specified wire-AND resolution for one or more multi-driven nets. The wand construct is not generally synthesizable and might lead to simulation and synthesis mismatch. Use '`set hdl option -wand_gate true`' to have LEC implement wand constructs as boolean AND gates.

Example

In the following example, the design declares wire `wand`, which is used for multi-driven output `q`. See line 4.

```
1 module test (a,b,q);
2 input a, b;
3 output q;
4 wand q;
5 assign q = a;
6 assign q = b;
7 endmodule
```

VLG4.2

Message

```
Wire type wor is used
```

Default Severity

Warning

Description

The design includes a user-specified wire-OR resolution for one or more multi-driven nets. The wor construct is not generally synthesizable and might lead to simulation and synthesis mismatch. Use the 'set hdl option -wor_gate true' to have LEC implement wor constructs are boolean OR gates.

Example

In the following example, the design declares wire `wor`, which is used for multi-driven `out0`. See line 5.

```
1 module test ( a, b, c, sel, out0);
2 input  a, b, c;
3 input  sel;
4 output out0;
5 wor out0;
6 assign out0 = a;
7 assign out0 = b;
8 endmodule
```

VLG4.3

Message

`Implicit declared net is generated`

Default Severity

Error

Description

An implicit declared net is generated.

Example

In the following example, `in_top` is an implicit declared net (see line 13):

```
1 `default_nettype none
2 module test(out,in);
3 input in;
4 output out;
5 reg out;
6   always @(in)
7   begin
8     if (in ) out = in; else out = ~in;
9   end
10 endmodule
11 module top (out_top);
12 output out_top;
13 test s1(out_top,in_top);
14 endmodule
```

VLG4.3a

Message

`Implicit declared net in port declaration was converted to wire type`

Default Severity

Warning

Description

When Verilog directive `` `default_nettype `` is set to none, all port types should be explicitly declared. Implicit nets will trigger this message.

Example

In the following example, `` `default_nettype `` is set to none, but there is an implicit net called `din`:

```
01  `default_nettype none
02  module test(input logic din, output logic dout);
03    assign dout = din;
04  endmodule
```

```
LEC gives the warning message as follows.
// Warning: (VLG4.3a) Implicit declared net in port declaration
// was converted to wire type.
// Warning: Implicit net 'din' not allowed due to directive
// (`default_nettype none) on line 2 at column 27 in file 'a.v'
```

VLG4.4

Message

`Cutpoint not found`

Default Severity

Warning

Description

Cannot find the specified cutpoint.

Example

In the following example, the pragma cutpoint `ram2` cannot be found in the design (see line 7):

```
1 module test(clk, addr, din, dout);
2 input clk;
3 input [1:0]  addr;
4 input [0:0]  din;
5 output [0:0] dout;
6 reg   [0:0]  ram [3:0];
7   // pragma cutpoint "ram2"
8 always @(clk or addr or din) begin
9   if (clk && ram[0] ) ram[addr[1:0]] = din;
10 end
11 assign dout = ram[addr[1:0]];
12 endmodule
```

VLG4.5

Message

```
Wire type tri0/tri1 used
```

Default Severity

Warning

Description

The design includes a user-specified tri0/tri1 resolution that models nets with resistive pull-down/pull-up devices.

Example

In the following example, the design declares wire tri0, which is used for output q. See line 4.

```
1 module test (a, q);
2   input a;
3   output q;
4   tri0 w;
5   assign w = a;
6   assign q = w;
7 endmodule
```

VLG4.6

Message

`Net and variable declaration must appear after the port declaration`

Default Severity

Warning

Description

The net or variable declaration of a non-ANSII port occurs before its port declaration.

Example

In the following example, 'z' is first declared as 'reg' on line 2, and later declared as 'output' on line 4.

```
1  module test(in, clk, z);
2    reg z;
3    input in, clk;
4    output z;
5    always @(posedge clk)
6        z <= in;
7  endmodule
```

VLG5.1

Message

```
Primitive output port has multiple bits
```

Default Severity

Error

Description

The design includes one or more primitive output ports with multiple bits. Conformal ignores all but the least significant bit (LSB) of the primary vector output.

Example

In the following example, output `o` has three bits. Conformal only keeps the LSB. See line 4.

```
1 module test(o,a,b);
2 output [2:0] o;
3 input a,b;
4 and(o,a,b);
5 endmodule
```

VLG5.2

Message

```
Primitive input port has multiple bits. Ignore all but LSB bit
```

Default Severity

Warning

Description

One or more primitive input ports have multiple bits. Conformal changes all primitive vector input ports to scalar input ports by applying OR logic reduction.

Example

In the following example, Conformal reduces the two-bit input `a` to a single-bit input `a` by OR-ing `a[0]` and `a[1]` together. See line 5.

```
1 module test ( a, out0);
2 input[1:0]  a;
3 output out0;
4 wire out0;
5 not (out0,a);
6 endmodule
```

VLG5.2a

Message

```
Primitive input port has multiple bits. Convert to 1-bit by reduction-or
```

Default Severity

Warning

Description

The Conformal software assumes all Verilog primitives use a 1-bit expression to connect to their input terminals. If a multiple-bit expression is connected to an input terminal of a Verilog primitive, this multiple-bit expression is converted to 1-bit expression by one of the following methods:

1. Takes the LSB (least significant bit) of the multiple-bit expression. This method is the default conversion and will report the VLG5.2 rule message.
2. Treats the entire multiple-bit expression as a logical true/false value. This is the same as taking a reduction-OR evaluation of the multiple-bit expression. Use the `SET HDL OPTION -primitive_input_conversion logic` command and the software will report the VLG5.2a rule message.

Example

For file `t1.v`:

```
module t1(aa, bb, oo);
input aa;
input [2:0] bb;
output oo;
and u0(oo, aa, bb); // bb is 3-bit expression
endmodule
```

Use the following commands to see the `VLG5.2a` rule violation:

```
set hdl option -primitive_input_conversion logic
read design t1.v
report rule check -verbose
```

VLG5.3

Message

`Wire and port size declaration(s) do not match`

Default Severity

Warning

Description

In one or more cases, the design's wire and port size declarations do not match.

Example

In the following example, the declared size of input ports `a` and `b` is `[3:0]`, but the declared size of wires `a` and `b` is `[4:0]`. See lines 2 and 4.

```
1 module test (a, b, q);
2 input [3:0] a, b;
3 output [3:0] q;
4 wire [4:0] a, b;
5 assign q = a & b;
6 endmodule
```

VLG5.4

Message

`Port size of array instance does not match`

Default Severity

Error

Description

In one or more cases, port sizes of an array instance do not match.

Example

In the following example, output o has a port size of `[2:0]`, but inputs `a` and `b` have a port size of `[1:0]` for instance array `u1[1:0]`. See lines 2 and 3.

```
1 module test(o,a,b);
2 output [2:0] o;
3 input [1:0] a,b;
4 and u1[1:0] (o,a,b);
5 endmodule
```

VLG5.5

Message

`Internal primitive is recognized`

Default Severity

Warning

Description

Conformal triggers this rule check when it encounters an internal primitive. This rule check helps identify module instances that you should treat as blackboxes.

Example

In the following example, you might want to treat AND as a blackbox. See line 2.

```
1 module test4(input aa, bb, output oo);
2     AND u0(oo, aa, bb);
3 endmodule
```

VLG5.6

Message

`Named port association is ignored for primitive gate`

Default Severity

Warning

Description

Conformal ignores the named port association for the primitive gate.

Example

In the following example, line 2 uses the named port association:

```
1 module test(input aa, bb, output oo);
2   AND u0(.o(oo), .a(aa), .b(bb));
3 endmodule
```

VLG6.1

Message

```
Globally referenced variable is unresolved
```

Default Severity

Warning

Description

Conformal cannot find the globally referenced variable. Thus, the variable remains unresolved.

Example

In the following example, on line 16, Conformal cannot find variable `x` in module `sub`. Thus, variable `x` remains as an undriven net.

```
 1 module sub (a, b, c);
 2 input [7:0] a, b;
 3 output [7:0] c;
 4 wire [7:0] wiretmp ;
 5
 6 assign wiretmp = a & b;
 7 assign c = wiretmp ;
 8 endmodule
 9
10 module main (i1, i2, o1, o2);
11 input [7:0] i1, i2;
12 output [7:0] o1;
13 output o2;
14
15 sub inst1 (i1, i2, o1);
16 assign o2= inst1.x;
17 endmodule
```

VLG6.1a

Message

`Unresolved hierarchical reference`

Default Severity

Error

Description

A Verilog hierarchical, referenced variable does not exist in the Verilog module.

Example

The following example triggers this message because `bb.cc` is unresolved.

```
module t2(aa, oo);
input aa;
output oo;
generate
  if (1) begin: bb
    wire CC;
    assign CC = 1'b1;
  end
endgenerate
  assign oo = aa && bb.cc; //  correct oo = aa && bb.CC;
endmodule
```

VLG6.2

Message

`Globally referenced variable is resolved`

Default Severity

Note

Description

The globally referenced variable is found and resolved.

Example

In the following example, on line 16, Conformal finds variable `tmp` in module `sub`.

```
 1 module sub (a, b, c);
 2 input [0:0] a, b;
 3 output [0:0] c;
 4 wire [0:0] tmp ;
 5
 6 assign tmp = a & b;
 7 assign c = tmp ;
 8 endmodule
 9
10 module main (i1, i2, o0, o1, o2, o3);
11 input [0:0] i1, i2;
12 output o0;
13 output [0:0] o1, o2, o3;
14
15 sub inst1 (.a(i1), .b(i2), .c(o1));
16 assign o2 = inst1.tmp ;
17 endmodule
```

VLG6.2a

Message

```
Variable size is default to one bit
```

Default Severity

Warning

Description

Indicates that in Verilog, the size of an undeclared variable is set to one bit by default.

Example

In the following example, on line 4, Conformal the size of the variable tmp defaults to one bit.

```
1 module top(ii, oo);
2 input ii;
3 output oo;
4     assign tmp = ii;
5     assign oo = tmp;
6 endmodule
```

VLG6.2b

Message

```
Globally referenced variable is resolved as unconnected net
```

Default Severity

Warning

Description

The globally referenced variable is found and resolved. Since the referenced variable exists in a separate module hierarchy from the current module, the variable is modeled as an unconnected net.

Note: See the documentation of "`SET_HDL_OPTION -HIEREF_TO_PORT_CONN on`" to connect the hierarchically referenced variable to the actual net.

Example

In the example below on line 12, "`ptest_u.foo`" refers to an existing net "`foo`" in submodule "`ptest_u`". Although the hierarchical reference is resolved, "`ptest_u.foo`" still remains unconnected.

```
 1  module ptest(input clk, rst, input [3:0] d, output reg [3:0] q);
 2      wire foo;
 3      always @(posedge clk or negedge rst)
 4         if (!rst)
 5             q <= 0;
 6         else
 7             q <= d;
 8      assign foo = q[3];
 9  endmodule
10
11  module test(input clk, rst, input [3:0] d, output [3:0] q, output msb);
12      assign msb = ptest_u.foo;
13      ptest ptest_u(.*);
14  endmodule
```

VLG6.3

Message

`Unsupported system function call (converted to 1'b1)`

Default Severity

Warning

Description

Conformal does not support the `$system/$user_pli` function calls; instead, Conformal approximates the `$system/$user_pli` function call to `1'b1`.

Example

In the following example, Conformal approximates `$random` and `$time` to `1'b1`. See lines 4 and 5.

```
1 module test(aa, bb, o1, o2);
2 input aa, bb;
3 output o1, o2;
4   assign o1 = $random;
5   assign o2 = $time;
6 endmodule
```

VLG6.3a

Message

`Unsupported system task call`

Default Severity

Error

Description

`$systemtask(...)` is called as a concurrent statement in Verilog.

Note: This check has default error severity level, but you can set it to lower severity level to skip the error checking.

Example

In the following example, concurrent systemtask `$mytask` is not supported

```
module m1(aa, bb, oo);
input aa, bb;
output oo;
  $mytask(aa, bb, oo);
endmodule
```

VLG6.3b

Message

```
Unsupported system function call (converted to 1'b0)
```

Default Severity

Warning

Description

The tool does not support the `$system/$user_pli` function calls; instead, Conformal approximates the `$system/$user_pli` function call to `1'b0`.

Example

In the following example, Conformal approximates `$isunknown` to `1'b0`. See line 4.

```
1 module test(in, out);
2    input [3:0] in;
3    output out;
4    assign out = $isunknown(in[2:1]);
5 endmodule
```

VLG6.4

Message

```
Supported system datapath function call
```

Default Severity

Note

Description

Conformal triggers this rule check when it encounters any supported datapath functions. Conformal supports the following datapath functions.

```
$abs()
$blend()
$carrysave()
$compge()
$intround()
$inttrunc()
$lead0()
$lead1()
$log2()
$max()
$min()
$rotatel()
$rotater()
$round()
$sat()
$sgnmult()
```

Functions $carrysave(), $intround(), and $inttrunc() are not verifiable because their functions may vary with different implementations. These functions will be modeled as blackboxes.

Example

The following example uses the datapath function `$min()`:

```
module sub(o);
  parameter p1 = 0;
  parameter p2 = 0;
  output [31:0] o;
    assign o = $min(p1, p2);
endmodule

module top(o);
```

```
  output [31:0] o;
    sub #(4, 8) inst1(o);
endmodule
```

VLG6.4a

Message

```
System datapath function modeling is implementation dependent and is black boxed
```

Default Severity

Warning

Description

The datapath function is not verifiable because the function may vary with different implementations. The function will be modeled as a blackbox.

The following are the list of such datapath functions:

```
$carrysave()
$intround()
$inttrunc()
```

Example

The following example uses the datapath function `$carrysave() on line 9 and will trigger VLG6.4a`:

```
1 module test2e (din, sum);
2    input [11:0] din;
3    output [3:0] sum;
4    wire [3:0]   s1, s2;
5    wire [7:0]   cs;
6
7    assign s1 = din[3:0] + din[8];
8    assign s2 = s1 + din[7:4];
9    assign cs = $carrysave(s2 + {din[11:9], 1'b0});
10    assign sum = cs[7:4] + cs[3:0];
11 endmodule
```

VLG6.5

Message

`Time literal is unsupported`

Default Severity

Warning

Description

Conformal does not support time literals, such as `1fs`, `2ps`, and `3ns`, and that it converted the time literals to `1'b1`.

Example

In the following example, Conformal converts `(2ns > 1ns)` to `(1'b1 > 1'b1)`. See line 4.

```
1 module test5(input aa, bb, output oo);
2      reg oo;
3      always @* begin
4           if (2ns > 1ns) oo = !aa;
5           else if  (aa || bb)  oo = aa;
6           else                 oo = 1'b0;
7      end
8 endmodule
```

VLG6.6

Message

```
Event object is unsupported
```

Default Severity

Warning

Description

The event object is not supported.

Example

In the following example, `-> ev` (see line 7) is not supported:

```
1 module top (input in, output out);
2 test test1(in, out);
3 endmodule
4 module test(input aa, output oo);
5 event ev;
6 always begin
7 -> ev;
8 oo = aa;
9 end
10endmodule
```

VLG6.7

Message

```
Hierarchical function call is not supported (blackboxed)
```

Default Severity

Warning

Description

The hierarchical function call is not supported and will be blackboxed.

Example

In the following example, line 16 shows the unsupported hierarchical function call:

```
 1 module mod1 (input in1, output out1);
 2 reg out1;
 3 function ftn1;
 4 input in1;
 5 begin
 6 if (in1 == 0) ftn1 = in1;
 7 else ftn1 = 1;
 8 end
 9 endfunction
10 always @(in1)
11 out1 = ftn1(in1);
12 endmodule
13 module mod2 (input in2,  output out2);
14 reg out2;
15 always @(in2)
16 out2 = mod1.ftn1(in2);
17 endmodule
```

VLG6.8

Message

`Specify block is ignored`

Default Severity

Ignore

Description

The specify block is ignored. In a design, Verilog specify blocks contain path delay information that is not used in static verification. As a result, the contents of a specify block are ignored. Some library cells may have notifier registers used in a specify block that can potentially affect the logic. Specify blocks are non-synthesizable.

Example

In the following example, `specify` (line 5) is ignored:

```
1 module specify_blk ( o, i);
2   input   i;
3   output  o;
4   buf(o, i);
5   specify
6     specparam T1RISE$ = 2.7;
7     ( i *> o ) =(  183:311:549 ,  179:304:536  );
8   endspecify
9 endmodule
```

VLG6.9

Message

```
Space(s) not allowed after backslash '\' for continuation
```

Default Severity

Error

Description

Spaces are not allowed after a backslash.

Example

```
‘define    ADD(a,b,c)    assign c =  \
                                      ^^ Spaces are not allowed here
 a+b;
```

VLG6.10

Message

`Intra-assignment event specification is not supported`

Default Severity

Warning

Description

This construct is not supported because intra-assignment event specifications are not synthesizable. Cadence recommends that you remodel your HDL source code.

Example

In the following example, the intra-assignment in line 8 is not supported:

```
 1 module intra_assign_evt (clk, cout);
 2 input clk;
 3 output [3:0] cout;
 4 reg [3:0] cout;
 5 always @(posedge clk)
 6 begin
 7     cout = 4'b0;
 8     cout = @(posedge clk) cout + 1;
 9 end
10 endmodule
```

VLG6.12

Message

`fork-join constructs are not supported`

Default Severity

Warning

Description

The design includes fork-join constructs. The fork-join constructs cannot be synthesized, and are not supported. To avoid this error, remodel the design.

Example

In the following example, the `fork` and `join` constructs (lines 7 and 10) are not supported:

```
 1 module neg_fork_join (clk, cout);
 2   input clk;
 3   output cout;
 4   reg cout;
 5   always @(posedge clk)
 6   begin
 7     fork
 8         cout = 1'b0;
 9         cout = 1'b1;
10     join
11   end
12 endmodule
```

VLG6.13

Message

`force-release constructs are not supported because they cannot be synthesized`

Default Severity

Warning

Description

The design includes force-release constructs. The force-release constructs cannot be synthesized. To avoid this error, remodel your design.

Example

In the following example, the force and `release` constructs (lines 7 and 10) are not supported:

```
 1 module force_release (clk, cin, cout);
 2 input clk;
 3 input cin;
 4 output cout;
 5 reg cout;
 6 always @ (posedge clk)
 7 begin
 8     force cout = 1'b1;
 9     cout = cin;
10     release cout;
11 end
12 endmodule
```

VLG6.14

Message

```
Global reference on the left side of the assignment is not supported
```

Default Severity

Warning

Description

A global reference on left side of the assignment is not supported.

Example

In the following example, `botInst.memNd` (line 5) is a global reference on left side of the assignment and is not supported:

```
1 module test(d);
2 input [1:0] d;
3 bot botInst();
4 always @(d)
5   botInst.memNd[0] = d;
6 endmodule
7 module bot();
8 reg [1:0] memNd [7:0];
9 endmodule
```

VLG6.15

Message

`disable construct is not supported`

Default Severity

Warning

Description

The `disable` construct is supported only when applied to an enclosing named block. The Conformal software cannot disable tasks and non-enclosing named blocks. Remodel the design to avoid this error.

Example

In the following example, `specify` (line 5) is ignored:

```
 1 module test;
 2 always
 3 begin : break
 4 end
 5 endmodule
 6 module top;
 7 test t();
 8 integer i;
 9   always
10   begin
11     if (i==1)
12     begin:cont
13       disable top.t.break;
14     end
15   end
16 endmodule
```

VLG6.16

Message

`wait construct is not supported`

Default Severity

Warning

Description

The design includes a Verilog `wait` event control, which is not supported. You will need to remodel the design.

Example

In the following example, the `wait` event control (line 7) is not supported:

```
 1 module neg_wait (enable, a, b, c, d);
 2 input enable, b, d;
 3 output a, c;
 4 reg a, c;
 5 always @(enable or b or d)
 6 begin
 7     wait (!enable) #10 a = b;
 8     #10 c = d;
 9 end
10 endmodule
```

VLG7

Message

`Nets are renamed after removing backslash`

Default Severity

Warning

Description

Conformal has renamed one or more net names. Conformal renames all net names that include a backslash (\).

Example

In the following example, net name `\out3[0]` will be renamed `out3[0]1`. See line 4.

```
1 module test ( in1, in2, out3 );
2 output [1:0] out3;
3 input  in1, in2;
4 wire \out3[0] ;
5     assign \out3[0] = in1 & in2;
6     assign out3[1] = \out3[0] ;
7     assign out3[0] = \out3[0] ;
8 endmodule
```

VLG8

Message

`Buffer inserted`

Default Severity

Note

Description

Conformal has inserted one or more buffers because the design includes the `$setuphold` or `$recrem` timing checks from the Verilog IEEE Std P1364-Y2K.

Example

In the following example, the design uses the `$setuphold` timing check. See line 11.

```
 1 module test (CK,D,O);
 2 output O;
 3 reg O;
 4 input CK,D;
 5 wire D_del,CK_del;
 6 buf (CK_del,CK);
 7 buf (D_del,D);
 8 always @(posedge CK_del)
 9 O <= D;
10 specify
11 $setuphold(posedge CK,posedge D,tsu_d_h_ck,th_ck_d_l,notifier,,,CK_del,D_del);
12 endspecify
13 endmodule
```

VLG9

Message

`Names conflict with previous declarations`

Default Severity

Error

Description

Conformal found a name conflict among declarations.

Example

In the following example on line 12, instance `a` has the same name as input `a` at line 9.

```
1 module test (din, dout);
2 input din;
3 output dout;
4
5 assign dout=din;
6 endmodule
7
8 module top (a , b);
9 input a;
10 output b;
11
12 test a (a,b);
13 endmodule
```

Note: It is not legal to use the same name already being used in a prior declaration in the same name space. Further details can be found IEEE2005 Sect. 19.13, IEEE2009 Sect. 3.13, and IEEE2012 Sect. 3.13. on ’Name spaces’, and on IEEE2005 Sect. 19.13, IEEE2009 Sect. 23.9, and IEEE2012 Sect. 23.9 on ’Scope Rules’.

VLG9.1

Message

`Text macro is redefined`

Default Severity

Warning

Description

The text macro is redefined.

Example

In the following example, `val1` is redefined (see line 2):

```
1 `define val1 0
2 `define val1 1
3 module test (din, dout);
4 input din;
5 output dout;
6 assign dout=din;
7 endmodule
```

VLG9.2

Message

```
The `define macro is used
```

Default Severity

Note

Description

The `` `define `` macro is used in the Verilog files.

Example

In the following example, `` `define `` macro is used (see line 1):

```
1 `define VAL
2 module test(in,out);
3 input in;
4 output out;
5 assign out = in;
6 endmodule
```

VLG9.3

Message

`Text macro expansion has extra space after the ` character`

Default Severity

Warning

Description

The macro expansion contains extra space between the ` character and the macro name. All examples in the Verilog language reference manuals do not contain extra space between the ` character and the macro name.

Example

In the following example, VLG9.3 is reported.

```
module test(in,out);
 `define macro_name 1'b1
 input in;
 output out;
   assign out = ` macro_name; // Extra space triggers this warning
endmodule
```

VLG9.4

Message

`Instance name conflicts with previous declarations`

Default Severity

Error

Description

Conformal found a name conflict among instance declarations.

Example

In the following example, `u0` is redefined (see line 6):

```
1 module GCLK (clko, clki, ena);
2 input clki, ena;
3 output clko;
4 not  u0 (clki, clki);
5 DLAT l0 (q0, 1'b0, 1'b0, clki_, ena);
6 and  u0 (clko, q0, clki);
7 endmodule
```

VLG9.5

Message

`Text macro is not defined`

Default Severity

Warning

Description

The text macro is not defined and will be treated as null text.

Example

In the following example, text macro `` `VAL `` is not defined on line 4.

```
1 module top (ii, oo);
2 input  [2:0] ii;
3 output [2:0] oo;
4   assign oo = `VAL ii;
5 endmodule
```

VLG9.6

Message

```
Number of actual arguments and formal arguments do not match
```

Default Severity

Warning

Description

The number of actual arguments in the macro does not match the number of formal arguments in the macro declaration.

Example

For example:

```
1    `define D(a,b)  $display("x=", a, " y=", b);
2
3    module top(input [31:0] x, y, output [31:0] o);
4      always @(x or y) begin
5        `D(x, y)      // legal
6        `D(x, )       // legal
7        `D(,)         // legal
8        `D(x)         // illegal, only one argument
9        `D()          // illegal, only one empty argument
10        `D(x, y, o)   // illegal, more actual than formal arguments
11      end
12    endmodule
```

LEC warns about too few or too many actuals for formal arguments in Verilog macro.

```
// Warning: (VLG9.6) Number of actual arguments and formal arguments do no match
// Warning: Number of actual arguments 1 is different from number of formal
// arguments 2 on line 8 at column 6 in file 'test.v'
// Warning: (VLG9.6) Number of actual arguments and formal arguments do no match
// Warning: Number of actual arguments 0 is different from number of formal
// arguments 2 on line 9 at column 6 in file 'test.v'
// Warning: (VLG9.6) Number of actual arguments and formal arguments do no match
// Warning: Number of actual arguments 3 is different from number of formal
// arguments 2 on line 10 at column 11 in file 'test.v'
```

VLG9.7

Message

```
Text macro contains recursion
```

Default Severity

Error

Description

The tools encounters recursion when expanding definition of a text macro.

Example

In the following example, expansion of the macro `a(BKSIZE) on line 9 encounters a recursion when expanding macro `d(A) which points back to macro `a(A).

```
1  `define a(A) `b(A)
2  `define b(A) `c(A) + `d(A)
3  `define c(A) (A << 1)
4  `define d(A) `a(A)
5  module test(input clk, input in, output reg out);
6      parameter BKSIZE = 10;
7      always@(posedge clk)
8      begin
9      out = in & `a(BKSIZE);
10      end
11  endmodule
```

VLG9.8

Message

```
The `undefineall is used
```

Default Severity

Note

Description

The `undefineall macro is used after SystemVerilog 2009.

Example

In the following example, `undefineall macro is used. (see line 4)

```
1 `define ABC 1
2 `define DEF 2
3
4 `undefineall
5 `define DEF 2
6 module top();
7     logic [`DEF:0] a;
8 endmodule1
```

VLG9.9

Message

`Text macro has duplicated formal arguments`

Default Severity

Error

Description

The text macro contains duplicated formal arguments.

Example

Duplicated formal arguments cause ambiguity. Line 1 of the following example triggers this message, because Conformal would not know which value to be picked.

```
1 `define SUM(a, b, c, c) a+b+c
2  module test(out);
3  output [7:0] out;
4  // Conformal cannot determine whether out should be 1+2+3 or 1+2+4
5  assign out = `SUM(1, 2, 3, 4);
6  endmodule
```

VLG10

Message

`Non-blocking assignment is in disabled block`

Default Severity

Warning

Description

There are non-blocking assignments in a disabled block. These kind of assignments might have undefined behavior.

Example

In the following example, the disabled `blk1` block contains a non-blocking assignment on line 6.

```
1 module test (clk, a,b);
2  input clk,a;
3  output b;
4
5 always @(posedge clk) begin:blk1
6  b <= a;
7  disable blk1;
8 end
9 endmodule
```

VLG10.1

Message

`Non-blocking assignment encountered in function`

Default Severity

Error

Description

Non-blocking assignment was detected inside the function. You should not use non-blocking statements inside functions. Such usage can lead to compilation failures.

Note: Applies only to Verilog mode. Non-blocking assignments inside functions is supported in SystemVerilog mode.

Example

In the following example, `get_address` (line 12) is a non-blocking assignment:

```
1 module top (clk, result);
2  input        clk;
3  output [3:0] result;
4  reg    [3:0] result;
5  reg    [1:0] state_var;
6  function [3:0]  get_address;
7    input  [1:0]  state_var;
8  begin
9    case (state_var)
10      2'b00:
11      begin
12        get_address <= "0000";
13      end
14    endcase
15  end
16  endfunction
17  always @(posedge clk)
18  begin
19    result = get_address(2'b0);
20  end
21 endmodule
```

VLG10.2

Message

`deassign statements cannot be synthesized and are not supported`

Default Severity

Warning

Description

The deassign statements cannot be synthesized.

Example

In the following example, the `deassign` statement in line 13 is not supported:

```
1 module DeassignMod(a,b);
2 input a;
3 output b;
4 reg b;
5 reg reset, clk;
6 always
7 begin
8 if(clk)
9   b = a;
10 else if(reset)
11   b = ~a;
12 else
13 deassign b;
14 end
15 endmodule
```

VLG10.3

Message

```
Sequential assign statement is not supported
```

Default Severity

Warning

Description

Sequential 'assign' statement is not supported.

Example

In the following example, the sequential `assign` statement in line 13 is not supported:

```
1 module assignMod(a,b);
2 input a;
3 output b;
4 reg b;
5 reg reset, clk;
6 always
7 begin
8 if(clk)
9   b = a;
10 else if(reset)
11   b = ~a;
12 else
13 assign b = 0;
14 end
15 endmodule
```

VLG10.4

Message

```
The target of a Verilog concurrent assignment cannot be a 'reg' object
```

Default Severity

Error

Description

In Verilog designs, the target of a concurrent assignment cannot be a `'reg'` object .

VLG10.4 will not be reported by default for designs written in SystemVerilog. You can also manually control the reporting of this rule check using the "`SET HDL OPTION -report_vlg10_4_in_sv`" command.

Example

Line 4 of the following example triggers this message, because the target `oo` of the concurrent assignment cannot be a `'reg'` object.

```
1 module test(ii, oo);
2 input ii;
3 output reg oo;
4   assign oo = ii;
5 endmodule
```

VLG10.4a

Message

```
The target of a Verilog procedural assignment cannot be a 'net' object
```

Default Severity

Error

Description

In Verilog designs, the target of a procedural assignment cannot be a '`net`' object for Verilog designs.

Example

In line 5 of the following example, the target `oo` of the procedural assignment cannot be '`net`' object type:

```
1 module test (clk, ii, oo);
2 input clk, ii;
3 output oo;
4   always @ (posedge clk)
5     oo <= ii;
6 endmodule
```

VLG11.1

Message

`Combinational logic is inferred in an always_latch block`

Default Severity

Warning

Description

There is combinational logic inferred in an `always_latch` block. Latches are expected to have inferred logic.

Example

In the following example, the `always_latch` block contains combinational logic on line 7.

```
1 module test(a,in1,in2,o);
2  input a;
3  input byte in1, in2;
4  output byte o;
5
6 always_latch begin
7    o = in1 & in2;
8 end
9 endmodule
```

VLG11.2

Message

`Latches are inferred in an always_comb block`

Default Severity

Warning

Description

This rule check tells that there are inferred latches in an `always_comb` block. Combinational logic is expected to have inferred logic.

Example

In the following example, the `always_comb` block contains an inferred latch on line 7.

```
1 module test(a,in1,in2,o);
2  input a;
3  input byte in1, in2;
4  output byte o;
5
6 always_comb begin
7  if (a)
8   o = in1 & in2;
9 end
10 endmodule
```

VLG11.3

Message

```
Special always_latch or always_comb block should not contain sensitivity
     list or any event control statement
```

Default Severity

Warning

Description

An 'always_latch' or 'always_comb' block should not contains a trailing sensitivity list or any other event control statement within the block. The sensitivity list is automatically inferred from the contents of the block.

Example

In the following example, the event control '@(enable or address or data_in)' does not need to be specified and is inferred automatically from the 'always_latch' block. Conformal will issue VLG11.3 rule check message.

```
 module top(input logic enable, address, data_in,output logic [1:0] buffer);

  always_latch @(enable or address or datain)
  begin
      if (enable) buffer[address] <= data_in;
  end
endmodule
```

VLG12.1

Message

`Null statement ';' is not allowed inside a begin-end block and is ignored`

Default Severity

Warning

Description

The null statement ':' is not allowed inside a begin-end block and will be ignored.

Example

In the following example, the extra null statement (line 7) will be ignored.

```
1 module test(clk, out1, in1);
2 input clk, in1;
3 output out1;
4 reg out1;
5 always @(posedge clk)
6 begin
7   out1 <= in1;;
8 end
9 endmodule
```

VLG13.1

Message

`Unpacked dimension is not allowed`

Default Severity

Warning

Description

The unpacked dimension declaration is not allowed. It is only allowed for SystemVerilog.

Example

In the following example, in line 4, the unpacked dimension for the wire declaration is only allowed in SystemVerilog.

```
1 module test (aa, bb, oo);
2 input aa, bb;
3 output oo;
4 wire tt[1:0];
5 assign tt[0] = aa;
6 assign tt[1] = bb;
7 assign oo = tt[0] && tt[1];
8 endmodule
```

VLG13.2

Message

```
Unsized dimension is not allowed
```

Default Severity

Warning

Description

The unsized dimension is not allowed. Using unsized dimension in an array declaration will cause SystemVerilog module to be blackboxed.

Example

The following SystemVerilog examples are not allowed and will cause modules to be blackboxed:

```
task foo( string arr[] ); // Dynamic array of strings
bit [3:0] nibble[]; // Dynamic array of 4-bit vectors
integer mem[]; // Dynamic array of integers
```

VLG13.3

Message

```
Associative dimension is not allowed
```

Default Severity

Warning

Description

The SystemVerilog associative dimension is not allowed. Using associative dimension will cause SystemVerilog module to be blackboxed.

Example

The following SystemVerilog examples are not allowed and will cause module to be blackboxed:

```
integer i_array[*]; // associative array of integer (unspecified index)
bit [20:0] array_b[string];
  // associative array of 21-bit vector, indexed by string
event ev_array[myClass]; // associative array of event indexed by class myClass
```

VLG14.1

Message

`Missing module instance name`

Default Severity

Error

Description

The instance name is missing for the module.

Example

In the following example, in line 3, the instance name is missing for instantiation of module `sub`:

```
1 module sub (o, a, b);
2   input a, b;
3   output o;
4 endmodule
5 module top();
6   wire m, n, k;
7    sub (.o(k), .a(m), .b(n));
8 endmodule
```

VLG14.2

Message

```
Cannot append to elaborated module
```

Default Severity

Error

Description

This is triggered when you try to append information to an elaborated module.

Example

For the following command:

```
read design top.v -noelaborate
elaborate design top
read design append.v -append
```

When trying to append the information in append.v to the elaborated module top, VLG14.2 will be reported.

```
// top.v
module top(in1, in2, out0, out1);
input in1, in2;
output out0, out1;
assign out0 = in1;
endmodule
```

```
// append.v
module top(.*);
assign out1 = in2;
endmodule
```

VLG15.1

Message

`Block name is previously declared`

Default Severity

Error

Description

Indicates that the block name is previously declared in the same scope.

Example

In the following example, in line 9, block name '`b1`' in redeclared:

```
01 module test(clk,in, out);
02 input in,clk;
03 output reg out;
04 always @(clk)
05 begin
06 begin:b1
07   out = in;
08 end
09 begin: b1
10   out = ~in;
11 end
12 end
13 endmodule
```

VLG15.1a

Message

```
Generate block name is previously declared
```

Default Severity

Warning

Description

Indicates that the generate block name has been previously declared in the same module, package, or interface scope.

Example

In the following example, the two 'if' statements in the generate block evaluate to true. Hence, there are two blocks with the same name 'b1' in module 'test'.

```
module test (input a, b, output c);
  parameter P1 = 1;
  parameter P2 = 1;
generate
  if (P1) begin :b1
  assign c=a;
  end
  if (P2) begin :b1
  assign b=a;
  end
endgenerate
endmodule
```

VLG15.2

Message

```
Generate block is not named explicitly by user. LEC will use IEEE naming convention
```

Default Severity

Warning

Description

Indicates that the generate block name is not named and LEC will use IEEE 1800-2017, 27.6 conventional naming method to name these blocks.

Example

In the following example, the if-conditional generate block on line-4 has no name. Conformal will name the block 'genblk1' according to the IEEE naming convention for unnamed block, and will report rule check violation VLG15.2.

```
01  module ptest
02        #(parameter GEN = 0)
03         (input clk, rst, input [3:0] d, output reg [3:0] q);
04    if (GEN == 0) begin
05      always @(posedge clk or negedge rst)
06         if (!rst) q <= '0;
07         else q <= d;
08    end
09    else begin
10      always @(posedge clk or negedge rst)
11         if (!rst) q <= '1;
12         else q <= d;
13    end
14  endmodule
```

VLG16.1

Message

`Invalid Verilog syntax`

Default Severity

Error

Description

Indicates that a syntax error was found in the Verilog instantiation.

The correct syntax is '`<modname> <mod_instance_name>(...)`'.

Example

In the following example, in line 4, the correct syntax is '`test sub(...)`'.

```
1 module top(din, dout);
2 input din;
3 output dout;
4 test sub;
5 endmodule
```

VLG16.2

Message

`Verilog usage does not conform to IEEE standard`

Default Severity

Note

Description

The Verilog syntax used does not conform with the IEEE Verilog LRM.

Example

In this Verilog example,

```
1  module top(in1, in2, out1, out2);
2    input in1, in2;
3    output out1, out2;
4
5    assign {(out1), (out2)} = {in1, in2};
6  endmodule
```

The tool issues the following note, because of the extraneous parentheses in line 5:

```
// Parsing file test.v ...
// Note: (VLG16.2) Verilog usage does not conform to IEEE standard
// Note: Extraneous parentheses found in the left hand side net/variable of the
assignment on line 5 at column 12 in file 'test.v'
// Note: (VLG16.2) Verilog usage does not conform to IEEE standard
// Note: Extraneous parentheses found in the left hand side net/variable of the
assignment on line 5 at column 20 in file 'test.v'
```