

JasperGold Superlint Checks Reference

June 2018

© 2018 Cadence Design Systems, Inc. All rights reserved.

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

Product JasperGold Apps incorporates software developed by others and redistributed according to license agreement. For further details, see doc/third_party_readme.txt.

Trademarks: Open SystemC, Open SystemC Initiative, OSCI, SystemC, and SystemC Initiative are trademarks or registered trademarks of Open SystemC Initiative, Inc. in the United States and other countries and are used with permission.

Trademarks and service marks of Cadence Design Systems, Inc. contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address shown above or call 800.862.4522. All other trademarks are the property of their respective holders.

Restricted Permission: This publication is protected by copyright law and international treaties and contains trade secrets and proprietary information owned by Cadence. Unauthorized reproduction or distribution of this publication, or any portion of it, may result in civil and criminal penalties. Except as specified in this permission statement, this publication may not be copied, reproduced, modified, published, uploaded, posted, transmitted, or distributed in any way, without prior written permission from Cadence. Unless otherwise agreed to by Cadence in writing, this statement grants Cadence customers permission to print one (1) hard copy of this publication subject to the following conditions:

1. The publication may be used only in accordance with a written agreement between Cadence and its customer.
2. The publication may not be modified in any way.
3. Any authorized copy of the publication or portion thereof must include all original copyright, trademark, and other proprietary notices and this permission statement.
4. The information contained in this document cannot be used in the development of like products or software, whether for internal or external use, and shall not be used for the benefit of any other party, whether or not for consideration.

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

Restricted Rights: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor

Contents

<u>Preface</u>	13
<u>How This Reference Is Organized</u>	14
<u>Related References</u>	14
<u>Conventions Used in JasperGold Apps Documents</u>	15
<u>1</u>	
<u>Lint Checks</u>	17
<u>NAMING</u>	18
<u>BLK_NF_NMCV</u>	19
<u>FNC_NF_NMCV</u>	20
<u>IDN_NF_ALCA</u>	21
<u>IDN_NF_FCNL</u>	22
<u>IDN_NR_ESCA</u>	23
<u>INS_NF_NMCV</u>	24
<u>INT_NF_NMCV</u>	25
<u>MOD_NF_NMCV</u>	26
<u>PAR_NF_NMCV</u>	27
<u>REA_NF_NMCV</u>	28
<u>REG_NF_NMCV</u>	29
<u>SIG_NF_NMCV</u>	30
<u>SIG_NF_TSTN</u>	31
<u>TSK_NF_NMCV</u>	32
<u>WIR_NF_NMCV</u>	33
<u>FILEFORMAT</u>	34
<u>ARY_MS_DRNG</u>	35
<u>FIL_MS_DUNM</u>	36
<u>FIL_NF_NMCV</u>	37
<u>FIL_NR_MMOD</u>	37
<u>FIL_NS_SUFx</u>	38
<u>IDN_NF_NMCV</u>	39
<u>IDN_NR_AMKY</u>	40

JasperGold Superlint Checks Reference

<u>IDN NR CKYW</u>	40
<u>IDN NR SVKW</u>	41
<u>IDN NR SVKY</u>	42
<u>IDN NR VKYW</u>	43
<u>MOD NO PRTD</u>	44
<u>NAM NR REPU</u>	44
<u>SIG NF NMLN</u>	46
<u>CODINGSTYLE</u>	47
<u>ALW IS TASK</u>	52
<u>ALW NR OPSL</u>	53
<u>ALW NR TRIL</u>	53
<u>ARC NR GLSG</u>	54
<u>ARC NR UDAT</u>	56
<u>ARY NR LBND</u>	57
<u>ARY NR LOPR</u>	58
<u>ARY NR SLRG</u>	59
<u>ASG MS RPAD</u>	60
<u>ASG MS RTRU</u>	61
<u>ASG NR LMSB</u>	62
<u>ASG NR MINP</u>	63
<u>ASG NR NBFC</u>	64
<u>ASG NS TRNB</u>	65
<u>CAS NO CNST</u>	66
<u>CAS NO DEFA</u>	67
<u>CAS NR CMUL</u>	68
<u>CAS NR DEFA</u>	69
<u>CAS NR DEFN</u>	70
<u>CAS NR EXCS</u>	72
<u>CAS NR OVCI</u>	73
<u>CAS NR UCIT</u>	74
<u>CAS NR XCAZ</u>	75
<u>CLK XC LDTH</u>	76
<u>CND IR CCAS</u>	77
<u>CND NR EVXZ</u>	78
<u>CND NS MBEX</u>	79
<u>CST MS LPDZ</u>	80

JasperGold Superlint Checks Reference

<u>CST MS SIZE</u>	81
<u>CST NO DELY</u>	82
<u>DLY NO CSAG</u>	82
<u>DLY NR NEG</u>	83
<u>ENT NR DECL</u>	84
<u>EXP NR ITYC</u>	85
<u>EXP NR OVFB</u>	86
<u>FIL NR MTMS</u>	87
<u>FLP NR MASG</u>	88
<u>FLP NR MXCS</u>	89
<u>FNC MS AFPR</u>	90
<u>FNC MS MTYP</u>	91
<u>FNC NO AVAC</u>	92
<u>FNC NO USED</u>	93
<u>FNC NR AVGV</u>	94
<u>FNC NR NARG</u>	95
<u>FNC NR UGLV</u>	96
<u>IDX MS INDL</u>	97
<u>IDX NR DTTY</u>	99
<u>IFC NO FALW</u>	100
<u>IFC NR DGEL</u>	101
<u>INP NO USED</u>	102
<u>INP NR UNRD</u>	103
<u>INP UC INST</u>	104
<u>INS MS PSIZ</u>	105
<u>INS NR PODL</u>	106
<u>INS NR PTEX</u>	108
<u>INT NR PSBT</u>	109
<u>IOP NR UASG</u>	110
<u>LOP NR CTCE</u>	111
<u>LOP NR IDTY</u>	112
<u>LOP NR MLPV</u>	113
<u>LOP NR RPVR</u>	114
<u>MAC NR DMUL</u>	115
<u>MOD NO TMSL</u>	116
<u>MOD NR ALLD</u>	117

JasperGold Superlint Checks Reference

<u>MOD NR CASX</u>	118
<u>MOD NR CASZ</u>	119
<u>MOD NR PGAT</u>	120
<u>MOD NR PINS</u>	120
<u>MOD NR PLIF</u>	121
<u>MOD NR SYTS</u>	122
<u>MOD NR UNGN</u>	123
<u>OPR NR LOSD</u>	123
<u>OPR NR REAL</u>	124
<u>OPR NR TRNB</u>	126
<u>OPR NR UCMP</u>	127
<u>OPR NR UEOP</u>	128
<u>OPR NR UREL</u>	129
<u>OTP NO FDRV</u>	130
<u>OTP NR ASYA</u>	132
<u>OTP NR TSUP</u>	133
<u>OTP NR UDRV</u>	134
<u>OTP UC INST</u>	135
<u>PAR MS SDAS</u>	137
<u>PRO NR WAIT</u>	138
<u>PRT NR DFRG</u>	139
<u>PRT NR IOPT</u>	140
<u>PRT UC INST</u>	141
<u>REG NO READ</u>	142
<u>REG NR MBNT</u>	143
<u>REG NR RDBA</u>	144
<u>REG NR UASR</u>	145
<u>RST IS NFST</u>	145
<u>RST XC LDTH</u>	147
<u>SEQ NR BLKA</u>	148
<u>SIG NO ASIG</u>	149
<u>SIG NO READ</u>	150
<u>SIG NO USED</u>	151
<u>SIG NR IDRQ</u>	152
<u>TSK NO USED</u>	153
<u>TSK NR ASGV</u>	154

JasperGold Superlint Checks Reference

<u>TSK NR ESDE</u>	155
<u>TSK NR UGLV</u>	156
<u>VAR NO ASIG</u>	157
<u>VAR NO EVTR</u>	158
<u>VAR NO INTL</u>	158
<u>VAR NO READ</u>	159
<u>VAR NO USED</u>	161
<u>VAR NR INDL</u>	162
<u>VAR NR PRCD</u>	163
<u>WIR NO READ</u>	165
<u>WIR NO USED</u>	166
<u>WIR NR UASR</u>	167
<u>SIM SYNTH</u>	167
<u>ALW NR MSLV</u>	168
<u>ALW NR UNUV</u>	169
<u>CST NO BWID</u>	170
<u>CST NR MSBX</u>	171
<u>CST NR MSBZ</u>	172
<u>DLY NR XZVL</u>	172
<u>MOD NR SYXZ</u>	173
<u>SIG NR NDCL</u>	174
<u>SYNTHESIS</u>	175
<u>ALW IC SENL</u>	178
<u>ALW NO COMB</u>	179
<u>ALW NO ETRG</u>	179
<u>ALW NO EVTS</u>	180
<u>ALW NO FFLP</u>	181
<u>ALW NO LATH</u>	182
<u>ALW NR MCLK</u>	182
<u>ALW NR MXCK</u>	183
<u>ALW NR TCST</u>	184
<u>ASG NR NBCB</u>	185
<u>ASG NR SUPN</u>	186
<u>CAS NR EVLX</u>	187
<u>CLK IS NSYT</u>	188
<u>CLK NR DDBD</u>	189

JasperGold Superlint Checks Reference

<u>CLK NR EDGE</u>	190
<u>CND NR CMXZ</u>	190
<u>FLP NR ASMX</u>	191
<u>FLP NR MBCK</u>	194
<u>FNC NR CREC</u>	194
<u>IDX NR ORNG</u>	195
<u>INP NR ASGN</u>	196
<u>LAT NR BLAS</u>	197
<u>LAT NR MXCB</u>	197
<u>LOP NR FCND</u>	198
<u>LOP NR GLID</u>	199
<u>LOP NR INFL</u>	201
<u>LOP NR SRLG</u>	202
<u>MOD NR ALAS</u>	203
<u>MOD NR ASLD</u>	204
<u>MOD NR CNDO</u>	205
<u>MOD NR DSBC</u>	206
<u>MOD NR EVRP</u>	207
<u>MOD NR FINB</u>	208
<u>MOD NR FKJN</u>	209
<u>MOD NR FORE</u>	209
<u>MOD NR FREL</u>	210
<u>MOD NR IFSM</u>	211
<u>MOD NR INIB</u>	212
<u>MOD NR LDLY</u>	213
<u>MOD NR NSLP</u>	214
<u>MOD NR USWC</u>	215
<u>MOD NS ADAS</u>	216
<u>MOD NS DCSP</u>	217
<u>MOD NS GTIN</u>	218
<u>REG NR MNBA</u>	219
<u>RST IS CPLX</u>	220
<u>RST NR ASRO</u>	221
<u>SIG NO HIER</u>	222
<u>SIG NR MDRV</u>	223
<u>TSK NR CLKE</u>	224

JasperGold Superlint Checks Reference

<u>VAR NO COMR</u>	225
<u>VAR NR OUTR</u>	226
<u>VAR NR REAL</u>	227
<u>VAR NR TIME</u>	227
<u>STRUCTURAL</u>	228
<u>CLK IS MCDM</u>	229
<u>CLK NO HGHI</u>	230
<u>CLK NO INPT</u>	231
<u>CLK NS EDMX</u>	232
<u>CMB NR TLIO</u>	233
<u>FLP NO ASRT</u>	234
<u>FLP NR ASRT</u>	235
<u>FLP NR ENCT</u>	236
<u>INS NR INPR</u>	237
<u>LAT IS FLSE</u>	238
<u>MOD IS SYAS</u>	239
<u>MOD NO IPRG</u>	240
<u>MOD NS DCLK</u>	241
<u>MOD NS GLGC</u>	242
<u>RST IS DCMB</u>	243
<u>RST IS DFLP</u>	244
<u>RST IS DLAT</u>	245
<u>RST NO HGHI</u>	246
<u>RST NR MULT</u>	247
<u>RST NR PENA</u>	248
<u>SIG IS INTB</u>	249
<u>SIG IS MDRV</u>	250
<u>RACES</u>	251
<u>REG NR RWRC</u>	251
<u>REG NR TRRC</u>	252
<u>REG NR WWRC</u>	253

2

<u>DFT Checks</u>	255
<u>INTEGRATION</u>	256

JasperGold Superlint Checks Reference

<u>CON IS VRFD</u>	256
<u>CON NO VRFD</u>	257
<u>SIG MS WIDT</u>	258
<u>DFT FUNCTIONAL</u>	259
<u>CLK IS ACRF</u>	260
<u>CLK IS ACRL</u>	261
<u>CLK IS CDLA</u>	262
<u>CLK IS CDTE</u>	263
<u>CLK IS DDCF</u>	264
<u>CLK IS DDCL</u>	265
<u>CLK IS DLAT</u>	267
<u>CLK IS DRFF</u>	268
<u>CLK IS DRLA</u>	269
<u>CLK IS MMCK</u>	270
<u>CLK IS NDPI</u>	271
<u>FLP IS ASFL</u>	272
<u>FLP IS CDFF</u>	273
<u>FLP IS CSTD</u>	274
<u>FLP IS GATC</u>	275
<u>FLP IS GTCK</u>	276
<u>FLP IS TNEF</u>	277
<u>FLP NO CNPI</u>	278
<u>FLP NO SRST</u>	279
<u>FLP SR SAME</u>	279
<u>LAT EN NCPI</u>	280
<u>LAT IS INFR</u>	281
<u>MOD IS CMBL</u>	283
<u>OTP NO RGTM</u>	285
<u>RST IS DDAF</u>	286
<u>RST MX EDGE</u>	287
<u>RST MX SYAS</u>	288
<u>TRI NO EPTB</u>	291
<u>DFT SHIFT CAPTURE</u>	292
<u>CCN IS VRFD</u>	294
<u>CCN NO VRFD</u>	294
<u>CLK_FF CDCD</u>	295

JasperGold Superlint Checks Reference

<u>CLK NC GTEN</u>	296
<u>CLK NR DSFF</u>	297
<u>CLK NR RSTN</u>	298
<u>CLK TM DCDL</u>	299
<u>CLK TM DDFE</u>	300
<u>CLK TM DDLA</u>	301
<u>CLK TM DMCD</u>	302
<u>CLK TM PROP</u>	303
<u>FLP NO CTCL</u>	304
<u>FLP NO SCAN</u>	305
<u>FLP NR DBBM</u>	306
<u>FLP NR UVSC</u>	307
<u>ICG IS CTSC</u>	308
<u>LAT NO TRTM</u>	309
<u>MEM NC CKCT</u>	310
<u>MEM NC INPC</u>	311
<u>MEM NC OTPC</u>	312
<u>MEM NO FPMC</u>	313
<u>MEM NO MCFF</u>	315
<u>MEM NR MCLK</u>	316
<u>MEM SM MCMB</u>	317
<u>RST NC CTCL</u>	319
<u>RST NR CKSC</u>	320
<u>RST NR NCTL</u>	321
<u>RST NR TCLK</u>	322
<u>RST TM NCSC</u>	323
<u>SCN IS VRFD</u>	324
<u>SCN NO VRFD</u>	325
<u>SIG CM DNCF</u>	326
<u>SIG CT PROP</u>	327
<u>SIG NR ICGR</u>	328
<u>SIG SM DNCF</u>	329
<u>SIG ST PROP</u>	330
<u>TRI NC ENSS</u>	331
<u>TRI NO DZSS</u>	332

3

<u>Automatic Formal Checks</u>	335
<u>AUTO_FORMAL_CASE</u>	336
<u>CAS_IS_DFRC</u>	336
<u>CAS_NO_PRIO</u>	337
<u>CAS_NO_UNIQ</u>	338
<u>AUTO_FORMAL_SIGNALS</u>	339
<u>SIG_IS_DLCK</u>	341
<u>SIG_IS_STCK</u>	343
<u>SIG_NO_TGFL</u>	345
<u>SIG_NO_TGRS</u>	347
<u>SIG_NO_TGST</u>	349
<u>AUTO_FORMAL_X_ASSIGNMENT</u>	351
<u>ASG_IS_XRCH</u>	351
<u>AUTO_FORMAL_ARITHMETIC_OVERFLOW</u>	352
<u>EXP_IS_OVFL</u>	352
<u>AUTO_FORMAL_BUS</u>	354
<u>BUS_IS_CONT</u>	354
<u>BUS_IS_FLOT</u>	355
<u>AUTO_FORMAL_DEAD_CODE</u>	356
<u>BLK_NO_RCHB</u>	356
<u>AUTO_FORMAL_FSM</u>	357
<u>FSM_IS_DLCK</u>	358
<u>FSM_IS_LLCK</u>	360
<u>FSM_NO_MTRN</u>	362
<u>FSM_NO_RCHB</u>	364
<u>FSM_NO_TRRN</u>	365
<u>AUTO_FORMAL_OUT_OF_BOUND_INDEXING</u>	367
<u>ARY_IS_OOBI</u>	367
<u>AUTO_FORMAL_COMBO_LOOP</u>	368
<u>MOD_IS_FCMB</u>	368

Preface

This document provides descriptions, examples, and syntax to help you understand and implement lint, DFT, and automatic formal checks with the JasperGold® Superlint App using the JasperGold front end.

This preface includes the following information:

- [How This Reference Is Organized](#) on page 14
- [Related References](#) on page 14
- [Conventions Used in JasperGold Apps Documents](#) on page 15

Cadence® Design Systems, Inc. prohibits the use of our software in a way that does not comply with our written guidelines and documentation.

How This Reference Is Organized

This reference is organized as follows:

- Preface
Describes the purpose and scope of this reference, includes a preview of its contents, and lists typographic conventions.
- Chapter 1, “Lint Checks”
Describes checks in the lint domain.
- Chapter 2, “DFT Checks”
Describes checks in the DFT domain.
- Chapter 3, “Automatic Formal Checks”
Describes checks in the automatic formal checks domain.

Related References

The following related references are available from the tool:

- *JasperGold Apps Command Reference Manual* – Includes detailed syntax for all JasperGold commands (*Help – Command Reference Manual*).
- *JasperGold Platform and Formal Property Verification App User Guide* – Contains detailed information about the shared features of JasperGold Apps (*Help – User Guide*).
- *Superlint App User Guide* – Describes GUI features and procedures (*Help – Mini Guides – Superlint App User Guide*).

Conventions Used in JasperGold Apps Documents

The following tables list conventions used in syntax and text.

Table 1-1 JasperGold Apps Syntax Conventions

Convention	Definition
<code>Courier font</code>	Indicates text you will type on the command line.
<code>-underscore_separation</code>	Indicates a command switch. Switches are not case-sensitive.
<code>[]</code>	Indicates optional arguments. Do not type the square brackets.
<code> </code>	Indicates a choice (a logical OR) among alternatives. Do not type the vertical bar.
<code>\</code>	The backslash character (\) at the end of a line indicates that the command you are entering continues on the next line.
<code>*</code>	Indicates the preceding argument appears zero or more times per command.
<code>+</code>	Indicates the preceding argument appears one or more times per command.
<code>‘ ’</code>	Indicates the enclosed character(s) should be explicitly included on the command line. Do not type the single quotation marks.
<code><Italics></code>	Indicates a command option that you will replace with a valid value. Do not type the angle brackets.
<code>()</code>	Used as a grouping convention. Do not type the parentheses. For example, parentheses in the following syntax indicate that if you use the <code>-bbox</code> option, you will follow it with either the 0 or 1. <code>[-bbox (0 1)]</code>

JasperGold Superlint Checks Reference

Preface

Table 1-2 JasperGold Apps Text Conventions

Convention	Definition
<code>Courier font</code>	<p>This style indicates:</p> <ul style="list-style-type: none">■ Text you will type in GUI fields■ Commands and options■ Filenames and paths■ Code samples
<i>Italics</i>	User interface items such as button and field names.
<i>Menu – Option</i>	<p>GUI command sequence; that is, click on a menu followed by an option.</p> <p>Example: <i>Help – Command Reference Manual</i></p> <p>Meaning: Click on the <i>Help</i> menu and choose the option <i>Command Reference Manual</i>.</p>
<u>Blue text</u>	<p>Hyperlinked cross-reference.</p> <p>When you view the PDF version of Jasper® manuals from a computer screen, click on the blue text to view related information.</p>
➔	Single-step procedure.

Lint Checks

The LINT domain includes the following categories:

- NAMING on page 18
- FILEFORMAT on page 34
- CODINGSTYLE on page 47
- SIM_SYNTH on page 167
- SYNTHESIS on page 175
- STRUCTURAL on page 228
- RACES on page 251

NAMING

Using standard naming conventions in the design helps develop consistency across designs and makes them more reusable. Giving meaningful names to modules, instances, functions, tasks, clocks, and assertions helps in understanding and debugging designs. This category details rules for defining naming conventions of design constructs.

The NAMING category includes the following rules:

- BLK_NF_NMCV on page 19
- FNC_NF_NMCV on page 20
- IDN_NF_ALCA on page 21
- IDN_NF_FCNL on page 22
- IDN_NR_ESCA on page 23
- INS_NF_NMCV on page 24
- INT_NF_NMCV on page 25
- MOD_NF_NMCV on page 26
- PAR_NF_NMCV on page 27
- REA_NF_NMCV on page 28
- REG_NF_NMCV on page 29
- SIG_NF_NMCV on page 30
- SIG_NF_TSTN on page 31
- TSK_NF_NMCV on page 32
- WIR_NF_NMCV on page 33

JasperGold Superlint Checks Reference

Lint Checks

BLK_NF_NMCV

Short Message: *Begin/end block name '%s' does not follow the convention.*

Severity	Warning
Description	<p>This warning indicates that the block name does not follow the convention specified in the default rules file. The default rules file defines that the block name must end with <code>_blk</code>. However, you can modify the default naming convention by customizing this rule. Customize this rule by modifying the <code>pattern</code> parameter for this check in the <code>superlint.def</code> file:</p> <pre>params BLK_NF_NMCV {pattern="*_blk"}</pre>

The following code illustrates the occurrence of BLK_NF_NMCV:

```
module test1 (a_in, out1 );
input a_in;
output out1;
reg out1;
reg clock;
initial
begin
    clock =1'b0;
end

always
    #50 clock = ~clock;

always @(posedge clock)
begin:assgn_block
    out1 = a_in;
end
endmodule
```

In the above code, the block is named `assgn_block`. However, as per naming convention, block name must end with `_blk`. To avoid this warning, change the block name to `assgn_blk`.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

FNC_NF_NMCV

Short Message: *Function name '%s' does not follow the convention.*

Severity	Warning
Description	<p>This identifier violates function naming convention. To improve design readability, all function names should follow a naming convention. Specify a naming convention by modifying the <code>pattern</code> parameter in the default rules file. The default value of the <code>pattern</code> parameter is <code>func</code>. However, you can specify another value.</p> <pre>params FNC_NF_NMCV {pattern="func*"}</pre> <p>For example, if you specify the value for the <code>pattern</code> parameter as follows:</p> <pre>params FNC_NF_NMC{ pattern="func_*"}</pre> <p>FNC_NF_NMCV is reported if the name of the function does not start with <code>func_</code>.</p>

The following code illustrates the occurrence of FNC_NF_NMCV:

```
module mod_a(clk_a);
input clk_a;
reg calc_parity;
function clk_parity;
input [31:0] port_a;
begin
calc_parity = ^port_a;
end
endfunction
endmodule
```

To avoid this violation, rename `clk_parity` as `func_clk_parity`.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

IDN_NF_ALCA

Short Message: *Identifier '%s' contains characters that are not allowed.*

Severity	Warning
Description	<p>The specified identifier includes characters, which are not a part of the allowed character set specified in the default rules file. The default rules file defines that the identifiers must be composed of alphanumeric characters (A-Z, a-z, 0-9) or underscores (_). Identifier names composed of alphanumeric characters and underscores improves readability of the design.</p> <p>The default allowable character set is specified using the following parameter in the default rules file, and can be modified as required.</p> <pre>params IDN_NF_ALCA {pattern="^([_]?([a-zA-Z0-9:]+_?[a-zA-Z0-9:]*))*\$"}</pre> <p>At times, you might want to restrict the use of certain keywords as identifier names. To do so, specify those keywords in the reserved_keyword_list parameter as follows:</p> <pre>params IDN_NF_ALCA {reserved_keyword_list=""}</pre> <p>The tool reports a violation for identifiers that are named as keywords specified in the reserved_keyword_list parameter. By default, this parameter does not contain any value.</p> <p>The keywords specified in the reserved_keyword_list parameter are case sensitive. To ignore the case while reporting this check on keywords specified in the reserved_keyword_list parameter, set the value of the following parameter in the default rules file to no.</p> <pre>params IDN_NF_ALCA {reserved_keyword_list_case_sensitive="yes" "no"}</pre>

The following code illustrates the occurrence of IDN_NF_ALCA.

```
module mod_a (port__a, port_b);  
    input port__a, port_b;  
endmodule
```

In the above code, port `port__a` uses double underscore, which is not part of the allowable character set. If the parameter `reserved_keyword_list` includes `port_b` and the

JasperGold Superlint Checks Reference

Lint Checks

parameter `reserved_keyword_list_case_sensitive` is set to `no`, then the check is reported for port `port_b` also.

The following modified code eliminates this issue.

```
module mod_a (port_a, port_c);
    input port_a, port_c;
endmodule
```

[Back to Top](#)

IDN_NF_FCNL

Short Message: *First character of identifier '%s' is not a letter.*

Severity	Warning
Description	Identifier names not starting with a letter are not generically portable and can create problems with other tools in the design flow. In addition, using a letter as the first character improves readability of the design. To avoid this warning, modify the identifier name such that it starts with a letter.

The following code illustrates the occurrence of IDN_NF_FCNL.

```
module mod_a ( _port_a , port_b);
    input _port_a , port_b;
endmodule
```

In the above code, port `_port_a` starts with an underscore, which is not a letter. To avoid this warning, change `_port_a` to `port_a`. The following modified code eliminates this issue.

```
module mod_a (port_a, port_b);
    input port_a, port_b;
endmodule
```

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

IDN_NR_ESCA

Short Message: *Identifier '%s' contains escaped names, which should not be used.*

Severity	Warning
Description	Escaped names should not be used as identifiers. Escaped names are not generically portable and can create problems with other tools in the design flow. Avoiding escaped names improves readability of the design. To avoid this warning, do not use escaped names as identifiers.
Parameter associated	<pre>params IDN_NR_ESCA {checks_on_netlist=no yes}</pre> <p>The default value of this parameter is <code>no</code>.</p> <p>When the value is set to <code>no</code>, <code>IDN_NR_ESCA</code> is checked only on RTL.</p> <p>When the value is set to <code>yes</code>, <code>IDN_NR_ESCA</code> is checked on netlist also.</p>

The following code illustrates the occurrence of `IDN_NR_ESCA`.

```
module mod_a ();  
    wire \wir_* ;  
endmodule
```

In the above code, `wire \wir_*` is an escaped name. The following modified code eliminates this issue.

```
module mod_a ();  
    wire wir_a;  
endmodule
```

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

INS_NF_NMCV

Short Message: *Instance name '%s' does not follow the naming convention.*

Severity	Warning
Description	<p>To improve design readability, ensure that all instance names follow the convention specified in the <code>superlint.def</code> file. The convention for instance names is specified through a pattern parameter provided in the rules file. The default rules file is located at <code><install_dir>/etc/res/rtltds/rules/superlint.def</code> and the parameter is specified as follows:</p> <pre>params INS_NF_NMCV {pattern="*"}</pre> <p>A violation is reported if the instance name does not follow the convention specified in the rules file.</p>

The following code illustrates the occurrence of INS_NF_NMCV.

```
module mod_b (port_a, port_b);
  input port_a;
  output port_b;

  mod_a inst_a(port_a, port_b);

endmodule

module mod_a (port_a, port_b);
  input port_b;
  output port_a;
  assign port_a = ~port_b;
endmodule
```

If the value of the pattern parameter is specified as `*_$_MODNAME` in the rules file, then the above code reports a violation for instance name `inst_a`. To avoid this violation, rename `inst_a` as `inst_a_mod_a` or modify the pattern parameter to match the convention in the design.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

INT_NF_NMCV

Short Message: *Integer var name '%s' does not follow the naming convention.*

Severity	Warning
Description	<p>This warning indicates that the integer variable name does not follow the convention specified in the rules file. By default, Superlint does not put any restriction on integer variable naming. As a result, any name is acceptable and no warning is generated by default. However, you can customize this rule to specify a naming convention for naming instances in your design. Customize this rule by modifying the <code>pattern</code> parameter in the <code>superlint.def</code> file.</p> <pre>params INT_NF_NMCV {pattern="*"}</pre> <p>You can modify the pattern to impose a naming convention.</p>

Consider changing the pattern parameter as follows:

```
params INT_NF_NMCV {pattern="*_intg"}
```

After changing the `pattern` parameter, `INT_NF_NMCV` warning is generated if the name of integer variables do not end with `_intg`. The following code illustrates the occurrence of `INT_NF_NMCV` for `P`.

```
module mod_a(port_a, clk, port_b);
input port_a, clk;
output port_b;
reg [1:6] reg_a;
integer P;
always@(negedge clk)
begin
    for (P = 1; P < 6; P = P+1)
        reg_a[P+1] = reg_a[P];
        reg_a[1] = port_a;
    end
assign port_b = reg_a[6];
endmodule
```

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

MOD_NF_NMCV

Short Message: *Module name '%s' does not follow the naming convention.*

Severity	Warning
Description	<p>To improve design readability, all module names should follow the recommended naming convention, which is specified through the following parameter in the default rules file located at <install_dir>/etc/res/rtlids/rules/superlint.def:</p> <pre>params MOD_NF_NMCV {pattern="*_mod"}</pre> <p>A violation is reported if module names do not follow the convention specified in the rules file.</p>

The following code illustrates the occurrence of MOD_NF_NMCV.

```
module mod_a(rst);  
  input rst;  
  reg reg_a;  
  always@(negedge rst)  
    reg_a = 0;  
endmodule
```

If the value of the `pattern` parameter is specified as `_mod` in the rules file, then the above code reports a violation for module `mod_a`. To avoid this violation, rename `mod_a` as `a_mod` or modify the parameter to match the convention in the design.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

PAR_NF_NMCV

Short Message: *Parameter name '%s' does not follow the naming convention.*

Severity	Warning
Description	<p>This warning indicates that the parameter name does not follow the convention specified in the default rules file. By default, Superlint does not put any restriction on parameter naming. As a result, any name is acceptable and no warning is generated by default. However, you can customize this rule to specify a naming convention for naming parameters in your design. Customize this rule by modifying the pattern parameter in the <code>superlint.def</code> file.</p> <pre>params PAR_NF_NMCV {pattern="*"}</pre> <p>You can modify the pattern to impose a naming convention.</p>

Consider changing the pattern parameter as follows:

```
params PAR_NF_NMCV {pattern="*_par"}
```

After changing the pattern parameter, PAR_NF_NMCV warning is generated if the parameter name does not end with `_par`. The following code illustrates the occurrence of PAR_NF_NMCV for `par_a`.

```
module mod_a(port_a, clk, port_b);
input port_a, clk;
output port_b;
parameter par_a=6;
reg [1:par_a] reg_a ;
integer P;
always @(negedge clk)
begin
    for (P = 1; P < par_a; P = P+1)
        reg_a[P+1] = reg_a[P];
        reg_a[1] = port_a;
    end
    assign port_b = reg_a[par_a];
endmodule
```

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

REA_NF_NMCV

Short Message: *Real variable name '%s' does not follow the naming convention.*

Severity	Warning
Description	<p>This identifier violates real variable naming convention. By default, Superlint does not impose any restriction on real variable naming. As a result, any name is acceptable and no warning is generated by default. However, you can customize this behavior and specify a naming convention for naming real variable instances in your design using the following parameter:</p> <pre>params REA_NF_NMCV {pattern="*"}</pre>

Consider changing the pattern parameter as follows:

```
params REA_NF_NMCV {pattern="*_rea"}
```

Now, the tool generates an REA_NF_NMCV warning for the example below.

```
module mod_a;
  reg [7:0] reg_a [5:0];
  reg [7:0] reg_b [5:0];
  reg [15:0] reg_c [7:0][10:0];
  integer i,j;
  real x,y;
  initial
  begin
    for(x=0;x<7;x=x+1)
      begin
        for(y=0;y<7;y=y+1)
          begin
            reg_c[i][j][7+:2]= reg_a[i][j] * reg_b[i][j];
          end
        end
      end
    end
  endmodule
```

To avoid this violation, rename x and y as x_rea and y_rea, respectively.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

REG_NF_NMCV

Short Message: *Register name '%s' does not follow the naming convention.*

Severity	Warning
Description	<p>This identifier violates register naming convention. By default, Superlint does not impose any restriction on register naming. As a result, any name is acceptable and no warning is generated by default. However, you can customize this behavior and specify a convention for naming register instances in your design. This can be done by modifying the pattern parameter in the default rules file.</p> <pre>params REG_NF_NMCV {pattern="*"}</pre>

Consider changing the pattern parameter as follows:

```
params REG_NF_NMCV {pattern="*_reg"}
```

Now, the tool reports REG_NR_NMCV if the name of the register does not end with `_reg`. See the example below.

```
module mod_a (port_a, port_b, port_c, port_d, port_e, clk_a);
output port_a, port_b, port_c;
input port_d, port_e;
input clk_a;
reg port_c;
always @(clk_a or port_e)
    if (clk_a)
        port_c = port_e;
endmodule
```

To avoid this violation, rename `port_c` as `port_c_reg`.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

SIG_NF_NMCV

Short Message: *Signal name '%s' does not follow the naming convention.*

Severity	Warning
Description	<p>This warning indicates that the signal name does not follow the convention specified in the default rules file. By default, Superlint does not put any restriction on naming signals. As a result, any name is acceptable and no warning is generated by default. However, you can customize this rule to specify a naming convention for naming signals in your design. You customize this rule by modifying the <code>pattern</code> parameter in the <code>superlint.def</code> file.</p> <pre>params SIGLNM {pattern="*"}</pre> <p>You can modify the pattern to impose a naming convention.</p>

Consider changing the pattern parameter as follows:

```
params SIG_NF_NMCV {pattern="*_sig"}
```

After changing the pattern parameter, SIG_NF_NMCV warning is generated if the signal name does not end with `_sig`. The following code illustrates the occurrence of SIG_NF_NMCV for count.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
entity test is
  port (
    s1 : out std_logic_vector (4 downto 0));
end test;
architecture a of test is
  signal count :std_logic_vector (4 downto 0);
begin process
  begin
    if (count = "000")then
      s1 <= "00111";
    else
      s1 <= "00000";
    end if;
```

JasperGold Superlint Checks Reference

Lint Checks

```
end process;  
end a;
```

[Back to Top](#)

SIG_NF_TSTN

Short Message: *DFT signal '%s' does not follow the recommended naming convention.*

Severity	Warning
Description	<p>To improve design readability, all test mode signal names must follow the recommended naming convention. The naming convention is specified through the following pattern parameter provided in the rules file.</p> <pre>params SIG_NF_TSTN {pattern="*_test"}</pre> <p>You can modify the naming convention by modifying the pattern parameter. A violation is reported if the test mode signal name does not follow the convention specified in the rules file.</p>

The following code illustrates the occurrence of SIG_NF_TSTN.

```
module test(port_a, clk);  
    input port_a;  
    input clk;  
    reg reg_a;  
    always @(clk)  
        reg_a=port_a;  
endmodule  
Required DFT constraints in input Tcl file:  
config_rtltds -signal -constant {{port_a 1'b1}} -mode shift
```

If the value of the pattern parameter is specified as `*_test` in the rules file, then the tool reports a violation for test mode signal `port_a`. To avoid this warning, rename `port_a` as `port_a_test` or modify the pattern parameter to match the convention in the design.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

TSK_NF_NMCV

Short Message: *Task name '%s' does not follow the naming convention.*

Severity	Warning
Description	<p>This warning indicates that the task name does not follow the convention specified in the default rules file. The default rules file defines that the task name must begin with task. However, you can modify the default naming convention by customizing this rule. Customize this rule by modifying the <code>pattern</code> parameter in the <code>superlint.def</code>:</p> <pre>params TSK_NF_NMCV {pattern="task*"}</pre>

The following code illustrates the occurrence of TSK_NF_NMCV.

```
module FULL_ADDER (a, b,cin, async_reset, cout, sum);
input a,b,cin ;
input async_reset;
output cout,sum ;
reg cout,sum ;
reg s1;
always @(a or b or cin or async_reset)
begin
    if (async_reset == 1)
        begin
            s1 = 1'b0;
        end
    else if(a)
        begin
            s1 = (a ^ b) ;
            sum = (s1 ^ cin) ;
            H_adder;
        end
end

task H_adder ;
    reg t1,t2,t3 ;
    begin
        t1 = (a & b) ;
```


JasperGold Superlint Checks Reference

Lint Checks

```
t2 = (a & cin) ;
t3 = (b & cin) ;
cout = (t1 | t2 | t3) ;
end
endtask
endmodule
```

In the above code, the task is named as `H_adder`. However, as per naming convention, the task name must begin with `task`. You can avoid this warning by changing the task name to `task_adder`.

[Back to Top](#)

WIR_NF_NMCV

Short Message: *Wire name '%s' does not follow the naming convention.*

Severity	Warning
Description	<p>This identifier violates wire naming convention. By default, Superlint does not impose any restriction on wire naming. As a result, any name is acceptable and no warning is generated by default. However, you can customize this behavior and specify a convention for naming wire instances in your design. This can be done by modifying the pattern parameter in the default rules file.</p> <pre>params WIR_NF_NMCV {pattern="*"}</pre> <p>For example, if you specify the value for pattern parameter as</p> <pre>params WIR_NF_NMCV {pattern="*_wire"}</pre> <p>then, <code>WIR_NF_NMCV</code> is reported if the name of the wire does not end with <code>_wire</code>.</p>

The following code illustrates the occurrence of `WIR_NF_NMCV`.

```
module mod_a;
wire [1:0]wire_a;
reg reg_a;
reg reg_b;
always @(wire_a)
begin
    reg_a = wire_a[1] && wire_a[0];
end
```

JasperGold Superlint Checks Reference

Lint Checks

```
    reg_b = wire_a[1] && wire_a[0];  
end  
endmodule
```

To avoid this violation, rename `wire_a` as `wire_a_wire`.

[Back to Top](#)

FILEFORMAT

The FILEFORMAT category includes the following rules:

- [ARY_MS_DRNG](#) on page 35
- [FIL_MS_DUNM](#) on page 36
- [FIL_NF_NMCV](#) on page 37
- [FIL_NR_MMOD](#) on page 37
- [FIL_NS_SUFEX](#) on page 38
- [IDN_NF_NMCV](#) on page 39
- [IDN_NR_AMKY](#) on page 40
- [IDN_NR_CKYW](#) on page 40
- [IDN_NR_SVKW](#) on page 41
- [IDN_NR_SVKY](#) on page 42
- [IDN_NR_VKYW](#) on page 43
- [MOD_NO_PRTD](#) on page 44
- [NAM_NR_REPU](#) on page 44
- [SIG_NF_NMLN](#) on page 46

JasperGold Superlint Checks Reference

Lint Checks

ARY_MS_DRNG

Short Message: *Inconsistent ordering of bits in range declarations in module/design-unit %s -- should be all %s ranges.*

Severity	Warning
Description	A Reuse Methodology Manual HDL convention recommends that when defining multi-bit registers or buses, a consistent ordering should be used. For VHDL, you should use y downto x or x to y. For Verilog®, you should use x:0 or 0:x. By default, the direction is descending y downto x. However, you can change this by setting params ARY_MS_DRNG {direction=ascending} in the supelint.def file.
Parameter associated	direction=[ascending descending] Warns about inconsistent direction declarations. By default, the direction for the ARY_MS_DRNG rule is descending. Declarations that differ from this direction are flagged as errors.

The following code illustrates the occurrence of ARY_MS_DRNG.

```
module outp_wire_diff_decl_rng (in1, in2, in3, in4, out1, out2);
  input [3:0] in1;
  input [3:0] in2;
  input [0:3] in3;
  input [3:0] in4;
  output [3:0] out1;
  output [1:4] out2;
  wire [3:0] out1;
  wire [1:4] out2;
  assign out1 = in1 & in2 & in3 & in4;
  assign out2 = in1 | in2 | in3 | in4;
endmodule
```

In the above code, the range declaration of input in3, output out2, and wire out2 is in ascending order. When defining multi-bit registers or buses, a consistent ordering should be used.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

FIL_MS_DUNM

Short Message: *%s name '%s' differs from file name '%s'.*

Severity	Warning
Description	As per the RMM naming conventions, the name of the HDL file should be the same as the top-level design unit defined in the file. This enables easy identification of the design files and eases the maintenance of the code.

The following code illustrates the occurrence of FIL_MS_DUNM.

```
module abc1();
  reg [7:0] a, b, c, d;
  initial
  begin
    a= 'b1010_0110;
    b= 'ha6;
    c= 'o246;
    d= 'd166;
  end
endmodule
```

The name of the HDL file should be same as the top-level design unit. The above code reports a violation if the HDL file is named other than `abc1.v`.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

FIL_NF_NMCV

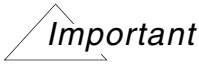
Short Message: *File name '%s' does not follow the recommended naming convention%.*

Severity	Warning
Description	<p>The file name should ideally be same as the name of the module included in that file. This ensures easy identification of the HDL files and better organization of the project files. To ensure the file name is the same as the name of the module included in that file, the default naming convention specifies the pattern parameter in the <code>superlint.def</code> file as follows:</p> <pre>params FIL_NF_NMCV {pattern="\$MODNAME"}</pre>

[Back to Top](#)

FIL_NR_MMOD

Short Message: *More than one design unit definition in file '%s'.*

Severity	Warning
Description	<p>To make the design modular and easy to reuse, each HDL source file must contain only one design unit. This is also as per Reuse Methodology Manual HDL conventions.</p> <div><p>This check is available for Verilog designs only.</p></div>

The following code illustrates the occurrence of FIL_NR_MMOD.

```
module neg_UNDRIV_oomr(a, b, c);  
  output c;  
  input a,b;  
  reg c;  
  lower I (a, b);  
endmodule
```

JasperGold Superlint Checks Reference

Lint Checks

```
module lower (a,b);
input a,b;
always @ (a or b)
begin
    neg_UNDRIV_oomr.c = a & b;
end
endmodule
```

In the above code, both the modules, `neg_UNDRIV_oomr` and `lower` are defined in a single file. Defining multiple modules in a single file prevents the module from being modular. Instead, split module definitions into separate files.

[Back to Top](#)

FIL_NS_SUFIX

Short Message: *The file name '%s' is missing a valid HDL file name extension.*

Severity	Warning
Description	<p>The VHDL files should have a consistent suffix. This makes it easy to identify the VHDL files when they are used with other files.</p> <p>The suffix is controlled by the following parameters:</p> <ul style="list-style-type: none">■ <code>params FIL_NS_SUFIX {pattern=" (vhd\$ vhdl\$ v\$ cpp\$ cxx\$ c++\$ cc\$ c\$) "</code>■ <code>params FIL_NS_SUFIX {local_regex_style="full"}</code>

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

IDN_NF_NMCV

Short Message: *Identifier '%s' does not follow the recommended naming convention.*

Severity	Warning
Description	<p>To avoid conflicts with back-end tools, it is recommended that special keywords, such as VSS, VDD, and so on are not used in identifier names. All identifier names must follow the recommended naming convention. The keywords that should be excluded from identifier names are specified through a pattern parameter provided in the rules file. The default rules file is located at <code><install_dir>/etc/res/rtlids/rules/superlint.def</code> and the parameter is specified as follows:</p> <pre>params IDN_NF_NMCV {pattern=".*(VSS VDD GND VCC).* .*(vss vdd gnd vcc) .*"}</pre> <p>By default, the naming convention suggests that no identifier names should contain VSS, VDD, GND, and VCC. However, you can modify the naming convention by modifying the pattern parameter. A violation is reported if the identifier names do not follow the convention specified in the rules file.</p>

The following code illustrates the occurrence of IDN_NF_NMCV.

```
module mod_a(VSS_port_a, port_b, port_c);
input VSS_port_a, port_b;
output port_c;
mod1 GND_DATA(.a(VSS_port_a), .b(port_b), .z(port_c));
endmodule
```

If the value of the `pattern` parameter is specified as `.*(VSS|VDD|GND|VCC).*` in the rules file, then the above code reports a violation for identifiers `VSS_port_a` and `GND_DATA`. To avoid this violation, rename these identifiers.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

IDN_NR_AMKY

Short Message: *AMS reserved word '%s' used as an identifier or label.*

Severity	Warning
Description	An AMS reserved keyword is used as an identifier name in the design. This can lead to portability issues and should be avoided. AMS reserved keywords are the ones that are present in AMS only.

The following code illustrates the occurrence of IDN_NR_AMKY.

```
module mod_a (port_a, port_b, abs);  
  input port_a, port_b;  
  output abs;  
  reg abs;  
endmodule
```

In the above code, a violation is reported for `output abs` and `reg abs` because `abs` is an AMS reserved keyword and should be avoided.

[Back to Top](#)

IDN_NR_CKYW

Short Message: *C/C++ reserved word '%s' used as an identifier or label.*

Severity	Warning
Description	Use of C/C++ reserved words as identifier names might lead to portability issues. To avoid portability issues, do not use any C/C++ reserved words as identifier names.

The following code illustrates the occurrence of IDN_NR_CKYW.

```
module asm_mod ( input port_a, port_b, port_c, port_d,  
                 output reg port_e  
               );  
  wire delete;  
  reg double, dynamic_cast, explicit;
```


JasperGold Superlint Checks Reference

Lint Checks

```
endmodule
```

In the above code, C++ reserved words `delete`, `double`, `dynamic_cast`, and `explicit` are used as identifier names. As a result, the tool reports a violation. Rename these identifiers to avoid this violation.

[Back to Top](#)

IDN_NR_SVKW

Short Message: *SystemVerilog reserved word '%s' used as an identifier or label.*

Severity	Warning
Description	A SystemVerilog reserved keyword is used as an identifier name in the design. This can lead to portability issues. SystemVerilog reserved keywords are the ones that are present in SystemVerilog only.

The following code illustrates the occurrence of IDN_NR_SVKW.

```
module mod_a (port_a, port_b, covergroup);  
    input port_a, port_b;  
    output covergroup;  
    reg covergroup;  
endmodule
```

In the above code, a violation is reported for `output covergroup` and `reg covergroup` because `covergroup` is a SystemVerilog reserved keyword and should be avoided.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

IDN_NR_SVKY

Short Message: *VHDL reserved word '%s' used as an identifier or label.*

Severity	Warning
Description	As per the Reuse Methodology Manual (RMM) conventions [Section - 5.2.9], VHDL reserved words must not be used in a Verilog design. This is because the macro designs must be translatable from Verilog to VHDL.

The following code illustrates the occurrence of IDN_NR_SVKY.

```
module test(b,a);  
  input a;  
  output b;  
  assign b = my_function(a);  
  function my_function;  
    input in;  
    reg unused;  
    my_function = ~in;  
  endfunction  
endmodule
```

In the above code, an input port `in` and an output port `out` is declared. Both `in` and `out` are VHDL keywords and hence should be avoided in a Verilog module.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

IDN_NR_VKYW

Short Message: *Verilog reserved word '%s' used as an identifier or label.*

Severity	Warning
Description	<p>As per the Reuse Methodology Manual (RMM) conventions [Section - 5.2.9], Verilog reserved words must not be used as identifiers or labels in a VHDL design. This is because the macro designs must be translatable from VHDL to Verilog.</p> <p>Also, Verilog keywords should not be used as identifiers or labels in a Verilog design because it can lead to compilation errors.</p>

The following code illustrates the occurrence of IDN_NR_VKYW.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity proc_tester is
    port(d_in:in integer;
         d_out:out integer);
end entity proc_tester;

architecture rtl of proc_tester is
    begin
        alu:process is
            procedure do_arith(
                signal tri:in integer) is
                variable always,weak1,b:integer;
            begin
                always:=tri;
                b:=+always+tri;
                d_out<=b+always;
            end procedure do_arith;
        begin
            do_arith(d_in);
        end process alu;
    end architecture rtl;
```

In the above code, Verilog reserved word `tri` is used as a signal name in a VHDL design. As a result, a violation is reported. Remodel the design to avoid this issue.

JasperGold Superlint Checks Reference

Lint Checks

[Back to Top](#)

MOD_NO_PRTD

Short Message: *Top-level %s '%s' has no inputs/outputs/inouts.*

Severity	Warning
Description	A design unit/module should not be without port declarations. This makes the design uncontrollable and unverifiable. A testbench is often the only HDL design unit with no inputs/outputs/inouts.

The following code illustrates the occurrence of MOD_NO_PRTD

```
module test;
    reg a, b;
endmodule
```

In the above code, module `test` is without port declaration. Such use makes the design uncontrollable and unverifiable.

[Back to Top](#)

NAM_NR_REPU

Short Message: *Identifier, label, instance, or module name '%s' reused with just case difference as %s.*

Severity	Warning
Description	A previously used identifier, label, instance, or module name has been used again with case difference. Names that differ only by case may be considered as same or distinct objects depending on the language and/or tool. This could lead to unwanted behavior due to possible incorrect connection of two different nodes by downstream tools and component binding issues in case of mixed language.

The following code illustrates the occurrence of NAM_NR_REPU.

```
module test(q1, q2, q3, i1, i2, clk);
```

JasperGold Superlint Checks Reference

Lint Checks

```
output q1, q2, q3;
input i1, i2;
input clk;
reg q3;
lat I1(q1, i1, clk);
lat I2(q2, i1, clk);
always @(clk or i2)
    if(clk)
        q3 = i2;
endmodule
```

In the above code, `i1` is used as an input port and `I1` is used as an instance name for module `lat`. Similarly, `i2` is used as an input port and `I2` is used as an instance name. Here, the identifier is the same with a difference in case only. Names that differ by case only might be considered as same or distinct objects depending on the language and/or tool. Such usage leads to errors and hence should be avoided.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

SIG_NF_NMLN

Short Message: *Signal name '%s' is not of appropriate length (%d to %d characters).*

Severity	Warning
Description	<p>For readability and maintainability, the signal name length should be within a specified limit defined in the Superlint definition file. You can change this limit using the <code>min_length</code> and <code>max_length</code> parameters in the <code>superlint.def</code> file as follows:</p> <pre>params SIG_NF_NMLN {min_length=1 max_length=32}</pre> <p><code>min_length</code> specifies the minimum allowed signal name length and <code>max_length</code> the maximum allowed signal name length. To avoid this warning, specify signal name with an appropriate length. At times, you might want to specify valid signal names with a length less than the minimum specified limit. To avoid this warning for such signal names, you can include those signal names in the <code>exception_list</code> parameter as:</p> <pre>params SIG_NF_NMLN { exception_list=""}</pre> <p>By default, <code>exception_list</code> does not contain any value. The tool will not issue a <code>SIG_NF_NMLN</code> warning for signal names specified in the exception list.</p>

The following code illustrates the occurrence of `SIG_NF_NMLN`.

```
module mod_a();
function func_a;
    input[7:0] port_a;
    input[7:0] B_this_name_is_not_okay_is_too_long;
    reg reg_a;
    begin
    end
endfunction
endmodule
```

If the `max_length` is specified as 32 in the `superlint.def` file, then in the above code, signal name `B_this_name_is_not_okay_is_too_long` will result in a warning because it exceeds 32 characters. To avoid this warning, change the signal name so that it is within the limit defined in the `superlint.def` file.

[Back to Top](#)

CODINGSTYLE

This category consists of the following types of checks:

- Interface consistency checks, such as unconnected ports, incorrect number or type of task and/or function arguments, incorrect signal assignments to input ports
- Checks for unused and unassigned variables or undriven signals, range and index consistency checks, such as single-bit memory words, bit and/or part selects that are out of range, case expressions that are out of range
- Expression consistency checks, such as unequal operand lengths, real/time values that are used in expressions, incorrect rounding and/or truncation
- Assertion writing checks, such as hierarchical references passed as arguments to assertions and labels missing for statements `endproperty` and `endsequence`

The CODINGSTYLE category includes the following rules:

- [ALW_IS_TASK](#) on page 52
- [ALW_NR_OPSL](#) on page 53
- [ALW_NR_TRIL](#) on page 53
- [ARC_NR_GLSG](#) on page 54
- [ARC_NR_UDAT](#) on page 56
- [ARY_NR_LBND](#) on page 57
- [ARY_NR_LOPR](#) on page 58
- [ARY_NR_SLRG](#) on page 59
- [ASG_MS_RPAD](#) on page 60
- [ASG_MS_RTRU](#) on page 61
- [ASG_NR_LMSB](#) on page 62
- [ASG_NR_MINP](#) on page 63
- [ASG_NR_NBFC](#) on page 64
- [ASG_NS_TRNB](#) on page 65

JasperGold Superlint Checks Reference

Lint Checks

- CAS_NO_CNST on page 66
- CAS_NO_DEFA on page 67
- CAS_NR_CMUL on page 68
- CAS_NR_DEFA on page 69
- CAS_NR_DEFN on page 70
- CAS_NR_EXCS on page 72
- CAS_NR_OVCI on page 73
- CAS_NR_UCIT on page 74
- CAS_NR_XCAZ on page 75
- CLK_XC_LDTH on page 76
- CND_IR_CCAS on page 77
- CND_NR_EVXZ on page 78
- CND_NS_MBEX on page 79
- CST_MS_LPDZ on page 80
- CST_MS_SIZE on page 81
- CST_NO_DELY on page 82
- DLY_NO_CSAG on page 82
- DLY_NR_NEGT on page 83
- ENT_NR_DECL on page 84
- EXP_NR_ITYC on page 85
- EXP_NR_OVFB on page 86
- FIL_NR_MTMS on page 87
- FLP_NR_MASG on page 88
- FLP_NR_MXCS on page 89
- FNC_MS_AFPR on page 90
- FNC_MS_MTYP on page 91
- FNC_NO_AVAC on page 92

JasperGold Superlint Checks Reference

Lint Checks

- FNC_NO_USED on page 93
- FNC_NR_AVGV on page 94
- FNC_NR_NARG on page 95
- FNC_NR_UGLV on page 96
- IDX_MS_INDL on page 97
- IDX_NR_DTTY on page 99
- FNC_NR_UGLV on page 96
- IFC_NO_FALW on page 100
- IFC_NR_DGEL on page 101
- INP_NO_USED on page 102
- INP_NR_UNRD on page 103
- INP_UC_INST on page 104
- INS_MS_PSIZ on page 105
- INS_NR_PODL on page 106
- INS_NR_PTEX on page 108
- INT_NR_PSBT on page 109
- IOP_NR_UASG on page 110
- LOP_NR_CTCE on page 111
- LOP_NR_IDTY on page 112
- LOP_NR_MLPV on page 113
- LOP_NR_RPVR on page 114
- MAC_NR_DMUL on page 115
- MOD_NO_TMSL on page 116
- MOD_NR_ALLD on page 117
- MOD_NR_CASX on page 118
- MOD_NR_CASZ on page 119
- MOD_NR_PGAT on page 120

JasperGold Superlint Checks Reference

Lint Checks

- MOD_NR_PINS on page 120
- MOD_NR_PLIF on page 121
- MOD_NR_SYTS on page 122
- MOD_NR_UNGN on page 123
- OPR_NR_LOSD on page 123
- OPR_NR_REAL on page 124
- OPR_NR_TRNB on page 126
- OPR_NR_UCMP on page 127
- OPR_NR_UEOP on page 128
- OPR_NR_UREL on page 129
- OTP_NO_FDRV on page 130
- OTP_NR_ASYA on page 132
- OTP_NR_TSUP on page 133
- OTP_NR_UDRV on page 134
- OTP_UC_INST on page 135
- PAR_MS_SDAS on page 137
- PRO_NR_WAIT on page 138
- PRT_NR_DFRG on page 139
- PRT_NR_IOPT on page 140
- PRT_UC_INST on page 141
- REG_NO_READ on page 142
- REG_NR_MBNT on page 143
- REG_NR_RDBA on page 144
- REG_NR_UASR on page 145
- RST_IS_NFST on page 145
- RST_XC_LDTH on page 147
- SEQ_NR_BLK_A on page 148

JasperGold Superlint Checks Reference

Lint Checks

- SIG_NO_ASIG on page 149
- SIG_NO_READ on page 150
- SIG_NO_USED on page 151
- SIG_NR_IDRG on page 152
- TSK_NO_USED on page 153
- TSK_NR_ASGV on page 154
- TSK_NR_ESDE on page 155
- TSK_NR_UGLV on page 156
- VAR_NO_ASIG on page 157
- VAR_NO_EVTR on page 158
- VAR_NO_INTL on page 158
- VAR_NO_READ on page 159
- VAR_NO_USED on page 161
- VAR_NR_INDL on page 162
- VAR_NR_PRCd on page 163
- WIR_NO_READ on page 165
- WIR_NO_USED on page 166
- WIR_NR_UASR on page 167

ALW_IS_TASK

Short Message: *Task '%s' used in always block.*

Severity	Info
Description	For improved performance, it is recommended to avoid the use of task calls in always blocks.

The following code illustrates the occurrence of ALW_IS_TASK.

```
module mod_a(clk, port_a, port_b, port_c, port_d);
input clk;
input port_a, port_b;
output port_c, port_d;
reg port_c, port_d;
    always @(posedge clk)
        begin
            task_a(port_c, port_d, port_a, port_b);
        end

    task task_a;
    output out_a, out_b;
    input in_a, in_b;
    begin
        out_a = in_a ^ in_b;
        out_b = in_a & in_b;
    end
endtask
endmodule
```

In the given example, a violation is reported for `task_a` in the `always` block.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

ALW_NR_OPST

Short Message: *Operator other than 'or' or ',' used in sensitivity list.*

Severity	Warning
Description	Use of operators other than the event operators <code>or</code> and <code>,</code> in the sensitivity list can lead to unintended behavior. When operators other than the event operators are used in the sensitivity list, those operators are handled as events whose occurrence depends on the result of the operation involved.

The following code illustrates the occurrence of ALW_NR_OPST.

```
module mod_a(clk, port_a, port_b);
  input clk;
  input [1:0]port_a;
  output [1:0]port_b;
  reg [1:0]port_b;
  always @(clk || port_a)
  begin
    port_b <= port_a;
  end
endmodule
```

In the above code, a logical `||` operator is used in the sensitivity list of the `always` block. As a result, a violation is reported. Remodel the design to avoid this violation.

[Back to Top](#)

ALW_NR_TRIL

Short Message: *Tristate logic inferred in %s block.*

Severity	Warning
Description	Some synthesis tools cannot infer the tristate logic defined in an <code>always/process</code> block. Such usage might lead to a mismatch between RTL simulation and synthesis results. To avoid this issue, do not define tristate logic in an <code>always/process</code> block.

JasperGold Superlint Checks Reference

Lint Checks

The following code illustrates the occurrence of ALW_NR_TRIL.

```
module mod_a(port_clk,port_a,port_b);
input port_clk;
input port_a;
output port_b;
reg port_b;
always @ ( port_a or port_clk )
begin
    if(port_clk)
        port_b = port_b;
    else
        port_b = 1'bz;
    end
endmodule
```

In the above code, tristate logic is inferred for port_b in the always block, resulting in a violation. To avoid this violation, use continuous assignment as shown below:

```
module mod_a(clk,port_a,port_b);
input clk;
input port_a;
output port_b;
wire port_b;
    assign port_b = clk ? port_a:1'bz;
endmodule
```

[Back to Top](#)

ARC_NR_GLSG

Short Message: *Global signal '%s' is being used in architecture '%s'.*

Severity	Warning
Description	Signal being used is a global signal. Global signals are signals declared in a VHDL package.

The following code illustrates the occurrence of ARC_NR_GLSG.

```
library ieee;
use ieee.std_logic_1164.all;
```

JasperGold Superlint Checks Reference

Lint Checks

```
use ieee.numeric_std.all;

package ApprovedType is
type LEC_ECR is (LECTURE, ECRITURE);
constant TAILLE_MAX:integer:=200;
subtype TAILLE is integer range 0 to TAILLE_MAX;
signal KOsignal1,KOsignal2, KOsignal3:std_logic;

function SIMPLE_SUB_min(A:in std_logic;
                        B:in std_logic)
return std_logic;
end ApprovedType;

package body ApprovedType is
function SIMPLE_SUB_min(A:in std_logic;
                        B:in std_logic)
return std_logic is
begin
    if A<B then
        return A;
    end if;
end SIMPLE_SUB_min;
end ApprovedType;

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
library worklib;
use worklib.ApprovedType.all;

entity GlobalSignals is
port (sort : out std_logic;
      ena : in std_logic );
end GlobalSignals;

architecture rtl_all_ResolvedGlobalSignals of GlobalSignals is
begin
    P1:process(KOsignal1,KOsignal2)
    begin
        sort <=SIMPLE_SUB_min(KOsignal3,KOsignal1);
    end process P1;
```

JasperGold Superlint Checks Reference

Lint Checks

```
end rtl_all_ResolvedGlobalSignals;
```

In the above code, `KOsignal1` is declared in the package `ApprovedType` and used in process, `P1`.

[Back to Top](#)

ARC_NR_UDAT

Short Message: *User-defined attribute '%s' is being used in architecture '%s'.*

Severity	Warning
Description	User-defined attribute is being used.

The following code illustrates the occurrence of `ARC_NR_UDAT`.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity UserDefinedAttributesUsage is
generic (size : integer := 3);
port( ena :in unsigned (size downto 0);
sort:out unsigned (size downto 0));
end UserDefinedAttributesUsage;
architecture BEH_UserDefinedAttributesUsage of UserDefinedAttributesUsage is
component and2
port ( in1 :in unsigned (size downto 0);
      in2 :in unsigned (size downto 0);
      out1 :out unsigned (size downto 0));
end component;
signal s : unsigned(size downto 0);
signal s1 : boolean;
attribute Preserve_signal : boolean;
attribute Preserve_signal of s: signal is true;
begin
    U1: and2 port map (in1=>ena, in2=>s, out1=>sort);
    s1 <= s'Preserve_signal;
end BEH_UserDefinedAttributesUsage;
```


JasperGold Superlint Checks Reference

Lint Checks

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity and2 is
generic (size : integer := 3);
port(in1 :in unsigned (size downto 0);
     in2 :in unsigned (size downto 0);
     out1 :out unsigned (size downto 0));
end and2;

architecture rtl_and2 of and2 is
begin
out1 <= in1 and in2;
end rtl_and2;
```

In the above code, user -defined attribute, `Preserve_signal` is used.

[Back to Top](#)

ARY_NR_LBND

Short Message: *Lower bound of '%s' is not '%d' in module/design-unit %s.*

Severity	Warning
Description	<p>To prevent problems in connecting blocks, declare all arrays, vectors, and ports with a consistent lower bound. Non-arithmetic data types in VHDL (integer, real, file, record, and bit) are an exception to this rule. The lower bound is specified through a parameter provided in the rules file. The default rules file is located at <code><install_dir>/etc/res/rtds/rules/superlint.def</code> and the parameter is specified as follows:</p> <pre>params ARY_NR_LBND {lower_bound=0}</pre> <p>By default, the lower bound is specified as 0. However, you can modify the lower bound. A violation is reported if the lower bound of arrays, vectors, and ports does not follow the convention specified in the rules file.</p>

The following code illustrates the occurrence of ARY_NR_LBND.

JasperGold Superlint Checks Reference

Lint Checks

```
module mod_a( port_a, port_b);
input [1:0] port_a, port_b;
wire [4:1]wir_a;
assign wir_a = {port_a, port_b};
endmodule
```

If the value of the `lower_bound` parameter is specified as 0 in the rules file, then the above code will report a violation for wire `wir_a`. To avoid this violation, declare `wir_a` as follows:

```
wire [3:0]wir_a;
```

[Back to Top](#)

ARY_NR_LOPR

Short Message: Logical %s operator applied to multi-bit operand %s in module/design-unit %s.

Severity	Warning
Description	It is not advisable to apply a logical AND (&&), logical NOT (!) or logical OR () to a vectored operand. Use explicit conversion of the multibit operands to a logical value and avoid any implicit conversion, which might lead to unexpected results.

The following code illustrates the occurrence of ARY_NR_LOPR.

```
module ARY_NR_LOPR (x, a, b);
output x;
input [1:0] a,b;
reg y, z;
assign x = !{y,z}; // bad, even if legal Verilog construct
always @(a or b)
begin
    y = a && 2'b10; // bad, even if legal Verilog construct
    z = 2 || b ; // bad, even if legal Verilog construct
end
endmodule
```

In the above code, the problem can be fixed by explicitly converting multibit operands to a logical value. For example, you can use reduction operator on `a` before the logical operator `&&` as shown below:

JasperGold Superlint Checks Reference

Lint Checks

```
y = (&(a)) && ((2'b10)) ; // good
```

[Back to Top](#)

ARY_NR_SLRG

Short Message: *The signal '%s', defined in %s '%s', is a one pin bus in its instance '%s'.*

Severity	Warning
Description	The signals defined as vectors have identical left and right range, leading to a single bit width. This can lead to issues in timing model and back end implementation. Define distinct left and right range for the signal to resolve this issue.
Parameter associated	<p>The behavior of this check is controlled using the following parameter in the default rules file:</p> <pre>params ARY_NR_SLRG {top_only=yes no}</pre> <p>The default value of this parameter is no, and in this case, Superlint checks for one pin buses in the entire design.</p> <p>When the value of this parameter is set to yes, one pin buses are checked only in the top module.</p>

The following code illustrates the occurrence of ARY_NR_SLRG.

```
module mod_a (port_a, port_b, port_c);  
    input [3:3] port_a;  
    input [1:2] port_b;  
    input [4:0] port_c;  
endmodule
```

In the given code, a violation will be reported on `port_a`. As a solution, declare `port_a` as scalar.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

ASG_MS_RPAD

Short Message: *Unequal length operands in assignment in module/design-unit '%s'. Length of RHS is less than LHS. LHS '%s' (%s) - %s bit(s), RHS '%s' (%s) - %s bit(s). RHS will be padded by %s bit(s).*

Severity	Warning
Description	Unequal length operands should not be used in assignments. If the bit width of RHS is less than the bit width of LHS, then padding of RHS takes place as defined by the sizing rules of the HDL. This might lead to unexpected results. Remodel the design such that equal length operands are used in assignments.

The following code illustrates the occurrence of ASG_MS_RPAD.

```
module mod_a(port_a, port_b, port_c);
  input [3:0] port_a;
  input [1:0] port_b;
  output [5:0] port_c;
  reg [5:0] port_c;
  always @(port_a or port_b)
  begin
    port_c = port_a;
  end
endmodule
```

In the above code, `port_a` (4-bit variable) is assigned to `port_c` (6-bit variable). As the bit width of RHS (`port_a`) is less than the bit width of LHS (`port_c`), `port_a` will be padded by two bits during assignment. Remodel the design to use equal length operands in assignments.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

ASG_MS_RTRU

Short Message: *Unequal length operands in assignment in module/design-unit '%s'. Length of RHS is greater than LHS. LHS '%s' (%s) - %s bit(s), RHS '%s' (%s) - %s bit(s). %s most significant bit(s) will be lost.*

Severity	Warning
Description	Unequal length operands should not be used in assignments. If the bit width of RHS is greater than the bit width of LHS, then truncation of RHS takes place as defined by the sizing rules of the HDL. This might lead to unexpected results. Remodel the design such that equal length operands are used in assignments.
Associated parameter	apply_loop_constraint= yes no The default value of this parameter is no. When <code>apply_loop_constraint</code> is set to yes, the integer size is determined by the maximum value that the integer can take.

The following code illustrates the occurrence of ASG_MS_RTRU.

```
module mod_a(port_a, port_b, port_c);
input [3:0] port_a;
input [6:0] port_b;
output [5:0] port_c;
reg [5:0] port_c;
always @(port_a)
begin
    port_c = port_b;
end
endmodule
```

In the above code, `port_b` (7-bit variable) is assigned to `port_c` (6-bit variable). As the bit width of RHS (`port_b`) is greater than the bit width of LHS (`port_c`), `port_b` will be truncated by one bit during assignment. Remodel the design to use equal length operands in assignments.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

ASG_NR_LMSB

Short Message: *Truncation of bits in a constant in module/design-unit %s. The most significant bits are lost.*

Severity	Warning
Description	A constant used in an expression has a width larger than the expected operand width. At least one of the bits which is to be truncated was determined to be 1. The constant value will be truncated, and a possible loss of precision might result. Review this expression, and adjust the operand widths so that no loss of precision occurs.

The following code illustrates the occurrence of ASG_NR_LMSB.

```
module const_truncation_lost_bits (a, b, out1, out2, out3);
input [3:0] a;
input [3:0] b;
output [7:0] out1;
output [7:0] out2;
output [7:0] out3;
wire [7:0] out1;
wire [7:0] out2;
reg [7:0] out3;
    assign out1 = 9'b110101010;
    assign out2 = 9'b010101010;
endmodule
```

In the above example, `out1` is of 8-bits while the RHS is of 9 bits. This results in truncation of the MSB. Review and remodel the design according to your needs. If you do not intend to lose the MSB, increase the size of `out1`.

[Back to Top](#)

ASG_NR_MINP

Short Message: *Assignment to a %s %s '%s' is not supported.*

Severity	Warning
Description	As a good coding practice, the module/task/function port which is defined as an <code>input</code> should not be modified inside the design. If the port is bi-directional, modify this port definition to an <code>inout</code> port.



This check is available for Verilog designs only.

The following code illustrates the occurrence of ASG_NR_MINP.

```
module assign_to_input (a, in1, in2, in3, in4, in5, in6, in7, out1);
  input a;
  input in1;
  input in2;
  input in3;
  input in4;
  input in5;
  input in6;
  input in7;
  output out1;
  wire out1;
  assign in1 = 1'b1;
  assign out1 = in1 & in2 & in3 & in4;
endmodule
```

In the above code, the input `a` is assigned a constant value `1'b1`. Delete this assignment or modify the declaration of `a` as `inout`.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

ASG_NR_NBFC

Short Message: *Non-blocking assignment encountered in function '%s'.*

Severity	Warning
Description	As per LRM[10.3.4 (f)] 1364-2001, you should not use non-blocking statements inside functions. Such use leads to compilation failure in HDL-ICE import, the RTL compiler of Palladium. The modules with non-blocking statements are directed to NC-Sim bin.

The following code illustrates the occurrence of ASG_NR_NBFC.

```
module top (clk, result);
input clk;
output [3:0] result;
reg [3:0] result;
reg [1:0] state_var;
function [3:0] get_address;
input [1:0] state_var;
begin
    case (state_var)
        2'b00:
            begin
                get_address <= "0000";
            end
    endcase
end
endfunction

always @(posedge clk)
begin
    result = get_address(2'b0);
end
endmodule
```

In the above example, function `get_address` is given a value using non-blocking assignment statement. To avoid this error, use blocking assignment. You can replace statement `get_address <= "0000";` with `get_address = "0000";`.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

ASG_NS_TRNB

Short Message: *Truncation in constant conversion without a loss of bits in module/design-unit %s.*

Severity	Warning
Description	A constant used in an expression has a width larger than the expected operand width. The constant value was truncated, and the truncated bits were determined to be zeros. This will result in no loss of precision.

The following code illustrates the occurrence of ASG_NS_TRNB.

```
module const_truncation_lost_bits (a, b, out1, out2, out3);
input [3:0] a;
input [3:0] b;
output [7:0] out1;
output [7:0] out2;
output [7:0] out3;
wire [7:0] out1;
wire [7:0] out2;
reg [7:0] out3;
    assign out1 = 9'b110101010;
    assign out2 = 9'b010101010;
endmodule
```

The boldface lines of code in the above example illustrate this problem. Truncation does happen but without loss of bits.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

CAS_NO_CNST

Short Message: *Case item expression is not a constant.*

Severity	Warning
Description	A case item expression that is not a constant value was encountered. This can result in complicated selection logic in a synthesized design.

The following code illustrates the occurrence of CAS_NO_CNST.

```
module dc_in_case_tag (a, b, c, sel, out1);
input a;
input b;
input [3:0] c;
input [3:0] sel;
output out1;
reg out1;
always @(a or b or c or sel)
begin case (sel)
4'b0010: out1 = a;
c:      out1 = b;
4'b011?: out1 = c;
4'b0x11: out1 = b;
default: out1 = 1'b1;
endcase
end
endmodule
```

In the above code, primary input `c` is used as a case item expression. As a result, the tool reports a violation. To avoid this violation, use any constant in place of the case item expression `c`.

[Back to Top](#)

Lint Checks

CAS_NO_DEFA

Short Message: *Case statement with no default. Case is too wide to check if all cases are covered.*

Severity	Warning
Description	To determine if a case statement is complete, it is necessary to expand all case expressions that involve do not care bits. This can be a computationally expensive operation for wide-case expressions. Currently, a limit of 16 bits has been set for expansion of case tags. If this limit is exceeded, this warning will be issued.

The following code illustrates the occurrence of `CAS_NO_DEFA`.

```
module case_wo_def_wide_sel (a, b, c, d, sel, out1);  
    input a;  
    input b;  
    input c;  
    input d;  
    input [259:0] sel;  
    output out1;  
    reg out1;  
  
    always @(a or b or c or d or sel)  
        begin case (sel)  
            260'h00000000000000000000000000000000000000000000000000000000000000000000 : out1 = a;  
            260'h00000000000000000000000000000000000000000000000000000000000000ff : out1 = b;  
            260'h000000000000000000000000000000000000000000000000000000000000ffff : out1 = c;  
            260'h0000000000000000000000000000000000000000000000000000000000fffff : out1 = d;  
        endcase  
    end  
endmodule
```

In the above code, the case statement does not have a default statement. The width of the selector `sel` is 260 bits, which exceeds the current limit of 16 bits set for expansion of case tags. As a result, this case statement will not be analyzed to see if all the cases are covered. To remove this problem, specify a default clause for the case statement.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

CAS_NR_CMUL

Short Message: *Case item expression covered more than once (covers same case item expression as in line %d) in module/design-unit %s.*

Severity	Warning
Description	More than one case item controlled by the same case item expression value was encountered. This can result in differences in the operation of a synthesized design versus a simulation model as the selection of which branch to evaluate might be different for the different tools.

The following code illustrates the occurrence of CAS_NR_CMUL.

```
module top(a,b,c,sel,out1);
input a, b, c;
input [3:0] sel;
output out1;
reg out1;
always @ (a or b or c or sel)
begin
    case(sel)
        0 : out1 = a;
        2 : out1 = b;
        0 : out1 = c;
        default : out1 = 1'b0;
    endcase
end
endmodule
```

In the above example, tag expression 0 is covered in more than one case items. Modify the case statement to remove duplication of case item expressions.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

CAS_NR_DEFA

Short Message: *The case items of the case statement in module/design-unit %s cover all the numerical values of the case expression. The default clause is not required.*

Severity	Warning
Description	A complete case statement with a redundant default clause may lead to simulation and synthesis mismatch. The default clause will never be reached after synthesis as there will be no logic synthesized for the default clause, but it might be reached during simulation for <i>x</i> and <i>z</i> values in the case selector.

The following code illustrates the occurrence of CAS_NR_DEFA.

```
module muxz (a,b,c,d,sel,out1);
input [3:0] a,b,c,d;
input [1:0] sel;
output [3:0] out1;
reg [3:0] out1;

always @(a or b or c or d or sel)
    casez(sel)
        2'b00: out1 = a;
        2'b01: out1 = b;
        2'b10: out1 = c;
        2'b11: out1 = d;
        2'bxx: out1 = 4'b00xx;
        2'bzz: out1 = 4'b00zz;
        default: out1 = 4'bx;
    endcase
endmodule
```

In the above code, all cases are covered in the case statement. Default clause is not required in this case.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

CAS_NR_DEFN

Short Message: *Case statement incomplete, %s.*

Severity	Error
Description	<p>This warning is generated if any of the following conditions is true:</p> <ul style="list-style-type: none">■ The case statement does not include a default clause and is incomplete (all possible values of the case selector are not covered by the possible case expressions).■ The signals to which a value is assigned in the case statement are not assigned a value prior to the case statement.■ The complete case statement includes a default clause. <p>The behavior of this check is controlled by following parameters in the Superlint rules file.</p> <pre>params CAS_NR_DEFN {case_default_specification="relax" "strict"}</pre> <p>The default value of this parameter is <code>relax</code>.</p> <p>Setting this parameter to <code>relax</code> causes Superlint to issue a warning if in an incomplete case statement either of the following applies:</p> <ul style="list-style-type: none">■ There is no default clause■ Signals to which a value is assigned in the case statement are not assigned a value prior to the case statement. <p>Setting this parameter to <code>strict</code> causes Superlint to issue a warning if signals to which a value is assigned in the case statement are not assigned a value prior to the case statement.</p> <p>Continued below.</p>

JasperGold Superlint Checks Reference

Lint Checks

CAS_NR_DEFN	params CAS_NR_DEFN
Description	{combinational_block_only="yes" "no"}
(Continued)	<p>The default value of this parameter is <code>no</code>. When the value of this parameter is set to <code>no</code>, the check is reported for case statements inside combinational as well as sequential always blocks. When the value of this parameter is set to <code>yes</code>, the check is reported only for case statements inside pure combinational always blocks.</p> <pre>params CAS_NR_DEFN {full_case_with_no_default_allowed="yes" "no"}</pre> <p>The default value of this parameter is <code>yes</code>. When the value of this parameter is set to <code>yes</code>, the check is not reported for a complete case statement with no default clause. When the value of this parameter is set to <code>no</code>, the check is reported for a complete case statement with no default clause.</p>

The following code illustrates the occurrence of CAS_NR_DEFN.

```
module case_wo_default (a, b, c, d, sel, out1);
input a;
input b;
input c;
input d;
input [3:0] sel;
output out1;
reg out1;

always @(a or b or c or d or sel)
begin
case (sel)
4'b0000: out1 = a;
4'b0001: out1 = b;
4'b0010: out1 = c;
4'b0011: out1 = d;
endcase
end
endmodule
```

In the above code, output `out1` is not assigned a value before the case statement. As a result, a violation is reported. Assign a value to `out1` prior to the case statement.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

CAS_NR_EXCS

Short Message: *Case selector %s is an expression.*

Severity	Warning
Description	<p>If an expression is used as a case selector, there is a possibility that the circuit might become complex and result in problems during synthesis. If it is necessary to evaluate the result of an expression, use a variable to evaluate the expression, and then use that variable as the case selector.</p> <p>By default, a violation is reported for all expressions used as case selectors except for expressions that use operators specified in the <code>ignore_operator</code> parameter list in the default rules file. The default rules file is located at <code><install_dir>/etc/res/rtlids/rules/superlint.def</code> and the <code>ignore_operator</code> parameter is specified as follows:</p> <pre>params CAS_NR_EXCS {ignore_operator= "concat:replicate:add:subtract:multiply:divide:modu lus"}</pre> <p>You can add operators to or remove operators from this list.</p>

The following code illustrates the occurrence of CAS_NR_EXCS.

```
module test (port_a, port_b, port_c);
input [2:0] port_a,port_b;
output[1:0] port_c;
reg [1:0] port_c;
always @( port_a or port_b )
begin
case( port_a & port_b )
3'b000 : port_c = 2'b00;
3'b001 : port_c = 2'b01;
default : port_c = 2'bxx;
endcase
end
endmodule
```

In the above code, expression `port_a & port_b` is used as a case selector. As a result, a violation is reported. To avoid this violation, remodel the design as follows:

JasperGold Superlint Checks Reference

Lint Checks

```
....
wire [2:0] wir_a;
assign wir_a = port_a & port_b;
always @( port_a or port_b )
begin
case( wir_a )
3'b000 : port_c = 2'b00;
3'b001 : port_c = 2'b01;
default : port_c = 2'bxx;
endcase
....
```

[Back to Top](#)

CAS_NR_OVCI

Short Message: *Case item expression overlapping with case item expression at line %d in module/design-unit %s.*

Severity	Warning
Description	Overlapping case items in a <code>casex/casez</code> statement might lead to differences in simulation or synthesis results due to creation of priority logic. This is because different simulation and synthesis tools evaluate x and z values in a case item differently. Such usage should be avoided.
Parameter associated	<p>The following parameters refine the behavior of the CAS_NR_OVCI check:</p> <pre>params CAS_NR_OVCI {ignore_z_overlap = "no" "yes"} params CAS_NR_OVCI {ignore_within_list_overlap = "no" "yes"} params CAS_NR_OVCI {ignore_overlap_except_unique = "no" "yes"}</pre> <p>The default for all is no.</p>

The following code illustrates the occurrence of CAS_NR_OVCI.

```
module mod_a (sel,port_a);
input [1:0] sel ;
```

JasperGold Superlint Checks Reference

Lint Checks

```
output [1:0] port_a;
reg [1:0] port_a;
always@(sel)
begin
  casex (sel)
    2'b0x : port_a = 2'b11;
    2'bx0 : port_a = 2'b01;
    2'b11 : port_a = 2'b01;
    default : port_a = 2'bxx;
  endcase
end
endmodule
```

In the above code, case item 2'bx0 overlaps with case item 2'b0x. As a result, a violation is reported. Remodel the design to avoid overlapping case items.

[Back to Top](#)

CAS_NR_UCIT

Short Message: *Unequal length in case item comparison (selector is %d bits, case tag expression is %d bits) in module/design-unit %s.*

Severity	Warning
Description	A case tag expression was found to have a width, which was inconsistent with the case selector. This can result in unexpected comparison results in simulation and/or synthesis. It is recommended that all case tag expressions are specified with the same width as the case selector.

The following code illustrates the occurrence of CAS_NR_UCIT.

```
module uneq_length_case_item (a, b, c, sel, out1);
input a;
input b;
input c;
input [3:0] sel;
output out1;
reg out1;
always @(a or b or c or sel)
```

JasperGold Superlint Checks Reference

Lint Checks

```
begin
  case (sel)
    4'b0010: out1 = a;
    2'h4d: out1 = b;
    8: out1 = c;
  endcase
end
endmodule
```

In the above code, the size of the case selector `sel` is 4 bits. However, the tags in the case items have widths of 4, 2, and 32 bits, respectively.

[Back to Top](#)

CAS_NR_XCAZ

Short Message: *Case item expression contains 'x' for a casez statement (useful only in casex statements) in module/design-unit %s.*

Severity	Warning
Description	For simulation, a case item expression containing an x found in a <code>case</code> or <code>casez</code> statement will be ignored. Comparisons to the case expression will never match.

The following code illustrates the occurrence of CAS_NR_XCAZ.

```
module muxz (sel, out1);
  output [3:0] out1;
  input sel;
  reg [3:0] out1;

  always @(sel)
    casez (sel)
      2'bxx: out1 = 4'b00xx;
      2'bzz: out1 = 4'b0000;
      default: out1 = 4'b1111;
    endcase
endmodule
```

In the above code, `x` is found in the case item expression and hence will be ignored.

JasperGold Superlint Checks Reference

Lint Checks

[Back to Top](#)

CLK_XC_LDTH

Short Message: *The clock '%s' drives a combinational logic. Depth '%s' exceeded at '%s'.*

Severity	Warning
Description	The depth of combinational logic driven by the clock exceeds the threshold depth specified in the Superlint rules file. Longer logic paths result in longer path delays and, therefore, timing closure can become difficult.
Parameter associated	<p>The threshold value of depth can be controlled by the following parameter:</p> <pre>params CLK_XC_LDTH {clock_logic_depth_threshold=0}</pre> <p>By default, the value of this parameter is 0.</p> <p>The depth of the combinational logic is calculated on the basis of the weight assigned to different logic gates using the parameters present in the Superlint rules file.</p>

The following code illustrates the occurrence of CLK_XC_LDTH.

```
module mod_a (in_a, in_b, rst, clk, out_a, out_b);
input in_a, in_b, clk, rst;
output out_a, out_b;
reg out_a;
assign out_b = clk & in_b;
always @(posedge clk or negedge rst)
    if(!rst)
        out_a <= 1'b0;
    else
        out_a <= in_a;
endmodule
```

The Superlint rules file includes:

```
params CLK_XC_LDTH {clock_logic_depth_threshold=1}
```

JasperGold Superlint Checks Reference

Lint Checks

The above code reports a violation for the clock `clk` that drives the combinational logic and has logic depth more than the threshold value. To avoid this violation, remodel your design and ensure that clock `clk` is supplied only to the clock pin of a flip-flop.

[Back to Top](#)

CND_IR_CCAS

Short Message: *Constant conditional expression encountered.*

Severity	Warning
Description	If a constant expression is used in the condition of an if/case statement or a ternary operator, the same branch might be executed repeatedly, resulting in dead code. To avoid dead code, do not use a constant expression in the condition of an if/case statement or a ternary operator.

The following code illustrates the occurrence of CND_IR_CCAS.

```
module mod_a (input [2:0] port_a, output reg [1:0] port_b, port_c);
  parameter param_a = 2;
  parameter param_b = 3;
  parameter param_c = (param_a < param_b) ? 0 : 1;
  always @( port_a )
    case( 1'b1 )
      port_a[0] : port_b = 2'b00;
      port_a[1] : port_b = 2'b01;
      port_a[2] : port_b = 2'b10;
      default : port_b = 2'bxx;
    endcase
  if(port_a && (param_a == 0))
    port_c = 2'b00;
endmodule
```

In the above code, constant expressions are used in the if statement, the case statement, and the ternary operator. As a result, a violation is reported. Remodel the design to avoid this issue.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

CND_NR_EVXZ

Short Message: *Conditional expression evaluates to 'x' or 'z'.*

Severity	Warning
Description	When the conditional expression evaluates to <code>x</code> or <code>z</code> , the expression is always treated as false and such an expression might make some lines of code redundant. For example, in the case of an <code>if-else</code> statement, if the expression is always false then the true condition of the <code>if</code> block will never get executed. Similarly, in a case statement, some of the branches might become redundant.

The following code illustrates the occurrence of CND_NR_EVXZ.

```
module m (in1,out);
  input [3:0] in1;
  output out;
  reg [3:0] out;
  assign w = 1'bz;
  always @ (w or in1) begin
    if(w)
      out = 1'b1;
    else out = in1;
  end
endmodule
```

In the above example, in the `if` statement the conditional expression `w` is assigned a value `z`. In this case, as `w` is always treated as false, the `if` block will never get executed. To avoid this warning, do not assign `x` or `z` values to condition expressions.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

CND_NS_MBEX

Short Message: *Expression in condition does not result in a single bit value in module/design-unit %s.*

Severity	Warning
Description	<p>As a good coding practice, and to make the design behavior definite, the condition expression should be written such that it evaluates to TRUE or FALSE. A multi-bit result of a conditional expression can lead to ambiguous and nonstandard simulator behavior. The condition in the following constructs will be evaluated to see that it results in a single bit value:</p> <ul style="list-style-type: none">■ <code>if(condition) .. else if(condition)</code>■ <code>while(condition)</code>■ <code>for(initial_assignment; condition; step_assignment)</code>■ <code>@(posedge condition or ...) @(negedge condition or ...)</code>■ <code>variable = (condition)? expr1 : expr2</code>

The following code illustrates the occurrence of CND_NS_MBEX.

```
module bad ( clk, out);
input clk;
output out;
reg [2:0] reg_a;
reg [4:0] reg_b;
always @( posedge clk)
begin
    if ( reg_b ) reg_a = 3'b0;
end
endmodule
```

In the above code, the conditional expression for the if statement is `reg_b`. `reg_b` is a 5-bit register, which does not result in single-bit value. To avoid this warning, modify the conditional expression such that it results in a single-bit value.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

CST_MS_LPDZ

Short Message: *Constant '%s' will be left-padded by %d '0' bits.*

Severity	Warning
Description	A constant used in an expression has a width greater than the operand width. As a result, the constant will be left-padded with an appropriate number of 0's. This can lead to ambiguous results. It is recommended that you modify the expression and use operands of equal widths.

The following code illustrates the occurrence of CST_MS_LPDZ.

```
module mod_a();  
    wire [1:0] wire_a= 1'b0;  
endmodule
```

In the above code example, a two-bit wire `wire_a` is being assigned a one-bit value.

The following modified code eliminates this issue.

```
module mod_a();  
    wire [1:0] wire_a= 2'b0;  
endmodule
```

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

CST_MS_SIZE

Short Message: *Constant '%s' has a value greater than its specified size.*

Severity	Warning
Description	<p>A constant has a value greater than its specified size. This can lead to truncation of data and result in data loss.</p> <p>This rule covers all binary, octal, decimal, and hexadecimal scenarios. For example, the tool returns a CST_MS_SIZE violation in each of the following instances:</p> <ul style="list-style-type: none">■ 3'b10011 has a 3-bit size but is assigned a 5-bit value.■ 8'h123 has an 8-bit size but is assigned a 12-bit value since each hexadecimal value is assigned in 4 bits.■ 5'o123 has a 5-bit size but is assigned a 9-bit value since each octal value is assigned in 3 bits.

The following code illustrates the occurrence of CST_MS_SIZE.

```
module top(outp);  
  output outp;  
  reg outp = 3'b11111;          // 5-bit value to a 3-bit constant  
endmodule
```

In the above example, `outp` has 3-bit size and is assigned a 5-bit value, which can result in data loss. Remodel the design to avoid this warning.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

CST_NO_DELY

Short Message: *Delay is not a constant expression.*

Severity	Warning
Description	In general, simulation time delays are not used in static verification and hence are ignored. Some synthesis tools support only integer delays that are multiples of the specified clock cycle for a design. Non-constant delay expressions are usually not supported by synthesis tools.

The following code illustrates the occurrence of CST_NO_DELY.

```
module non_constant_delay (data, out1);
    output out1;
    input data;
    reg out1;
    integer d1;
    always @(data)
        out1 <= #d1 data;
endmodule
```

In the above code, a non-constant delay has been used. The highlighted code illustrates this problem. Use constants or integers as delays to avoid problems during synthesis.

[Back to Top](#)

DLY_NO_CSAG

Short Message: *Delay used in conditional signal assignment.*

Severity	Error
Description	Delays should not be used in conditional signal assignment.

The following code illustrates the occurrence of DLY_NO_CSAG.

```
library ieee;
use ieee.std_logic_1164.all;
```

JasperGold Superlint Checks Reference

Lint Checks

```
entity test2 is
port ( KOSignal: in std_logic;
      ena:      in std_logic;
      sort:     out std_logic;
      del2:     out std_logic);
end test2;

architecture t_est of test2 is
begin
sort <= KOSignal after 5 ns when ena = '1';
del2 <= KOSignal when ena = '1';
end t_est;
```

The boldfaced code above shows a delay used in a conditional signal assignment. Remodel the design to avoid this violation.

[Back to Top](#)

DLY_NR_NEGT

Short Message: *Negative delay '%s' used in the statement.*

Severity	Error
Description	Negative delays are not allowed in the statement.

The following code illustrates the occurrence of DLY_NR_NEGT.

```
module test (a, b, sel, sel2, out);
input a, b, sel, sel2;
output out;
reg out;
always @(a or b or sel2)
  if (sel2)
    out = #10 a;
  else
    out = #(-10) b;
endmodule
```

The boldfaced code above shows a negative delay used in the statement. To fix this violation, ensure that the delay values are non-negative.

JasperGold Superlint Checks Reference

Lint Checks

[Back to Top](#)

ENT_NR_DECL

Short Message: *A VHDL entity should only consist of generic and port interface lists.*

Severity	Warning
Description	Good coding practice suggests that entity declarations should not contain declarations, use clauses, or statements.

The following code illustrates the occurrence of ENT_NR_DECL.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity EntityDeclaration is
  generic (g1 : integer := 8);
  port( b:in std_logic;
        i1: out integer;
        a:out std_logic);
  type Instruction is array(1 to 5) of natural;
  type Program is array(natural range <>) of Instruction;
  constant ROM_Code:Program:=
    (
      (20, 10, 11, 47, 5),
      (15, 55, 66, 8, 14)
    );
end EntityDeclaration ;

architecture rtl_rang of EntityDeclaration is
  type desc_level is (behavioral,rtl,structural);
  attribute description_level : desc_level;
  attribute description_level of rtl_rang: architecture is rtl;
begin
  a<=b;
  i1 <= g1;
end rtl_rang;
```

JasperGold Superlint Checks Reference

Lint Checks

In the above code, type definition and constant declaration is done inside the entity declaration. Such usage violates the coding guidelines and should be avoided.

[Back to Top](#)

EXP_NR_ITYC

Short Message: *Expression '%s' implicitly converted to type '%s' from type '%s'.*

Severity	Warning
Description	The expression is implicitly converted from one data type to another. Such conversions might lead to unpredictable results and should be avoided. While performing assignments, binary operations, and port mappings, the left and right sides must be of the same type.

The following code illustrates the occurrence of EXP_NR_ITYC.

```
module test();
  reg signed [31:0] reg_a;
  integer int_b;
  initial
  begin
    reg_a = int_b; // bad implicit conversion of type 'reg' (unsigned)
                  //to 'integer'(signed)
  end
endmodule
```

In the above code, an unsigned register `reg_a` is implicitly converted to an integer. This might lead to unpredictable results. Remodel the design to avoid this violation.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

EXP_NR_OVFB

Short Message: *Shift overflow in module/design-unit %s, some bits will be lost.*

Severity	Warning
Description	The shift distance found in the specified shift operation will result in a computed value that is larger than the destination variable. This will result in a loss of data, which will be shifted out of the most significant or least significant bit position of the destination variable. Review this shift operation to determine if the potential loss of data bits is acceptable.

The following code illustrates the occurrence of EXP_NR_OVFB.

```
module shift_overflow (a, b, out1, out2);
input [3:0] a;
input [3:0] b;
output [7:0] out1;
output [7:0] out2;
reg [7:0] out1;
reg [7:0] out2;
always @(a or b)
begin out1 = a << 9;
      out2 = b >> 3;
end
endmodule
```

In the above mentioned example, a is shifted by 9 bits and assigned to out1. But the size of out1 is 8 bits only. Increase the size of out1 or reduce the size of the shift operation.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

FIL_NR_MTMS

Short Message: *Multiple timescales exist in the design.*

Severity	Warning
Description	Specifying multiple timescales in a design might lead to problems when interconnecting different blocks. If blocks with different timescales are interconnected, unexpected simulation results might be generated due to a difference in timing behavior of interconnected blocks. To avoid unexpected simulation results, use a consistent timescale for all blocks in the design.

The following code illustrates the occurrence of FIL_NR_MTMS.

```
`timescale 1ns/1ns
module top_mod(out, port_a, port_b);
    input port_a;
    input port_b;
    output out;
    wire w;
    not_mod n1(w, port_a);
    assign out = w & port_b;
endmodule

`timescale 1ps/1ps
module not_mod(out, port_a);
    input port_a;
    output out;
    assign out = ~port_a;
endmodule
```

In the above code, the timescale of module `top_mod` is `1ns/1ns` but the timescale of module `not_mod` is `1ps/1ps`. As a result, a violation is reported. Remodel the design to use a consistent timescale for all blocks in the design.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

FLP_NR_MASG

Short Message: *Multiple set/reset signals used in flip-flop '%s'.*

Severity	Warning
Description	If multiple asynchronous set/reset signals are used in a flip-flop, the flip-flop might be set/reset unexpectedly. To avoid this issue, do not use multiple asynchronous set/reset signals in a flip-flop.

The following code illustrates the occurrence of FLP_NR_MASG.

```
module mod_a ( port_a, clk, rst, en, port_b );
input port_a, clk, rst, en;
output port_b;
reg port_b;
always @ (posedge clk or negedge rst or posedge en)
begin
    if( !rst || en )
        port_b <= 1'b0;
    else
        port_b <= port_a;
end
endmodule
```

In the above code, multiple asynchronous reset signals (`rst` and `en`) are used in a flip-flop. As a result, a violation is reported. To avoid this violation, remodel the design so that multiple asynchronous set/reset signals are not used in a flip-flop.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

FLP_NR_MXCS

Short Message: *Combinational logic detected in sequential block for flop '%s'.*

Severity	Warning
Description	As per coding guidelines, combinational logic should not be a part of specification of sequential logic block. The combinational logic should be written in a separate block that should be connected as an input to the sequential logic block.

The following code illustrates the occurrence of FLP_NR_MXCS.

```
module top(input in1, in2, clk, rst, output out1, out2);
  reg out1;
  reg out2;
  always@(posedge clk or negedge rst) begin // Sequential Logic Block
    if(!rst)
      out = 0;
    else begin
      out = in1 && in2; // Combinational logic embedded
    end
  end
end
endmodule
```

In the above code, combinational logic is embedded inside the sequential block. To avoid this violation, move combinational logic to a separate block.


[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

FNC_MS_AFPR

Short Message: *Size mismatch between formal '%s' of %s bit(s) and actual '%s' of %s bit(s) parameters of function '%s'. %s.*

Severity	Warning
Description	The size of the formal parameters and actual parameters of a function are not the same. To avoid undefined states and as a good coding practice, the size of formal parameters and actual parameters should be the same.
<div> <i>Important</i></div> <p>This check is available for Verilog designs only.</p>	

The following code illustrates the occurrence of FNC_MS_AFPR.

```
module task_arg_wrong_type (a, out1);
input a;
output out1;
integer out1;
always @(a)
t1(a, out1);
task t1;
input a;
output d;
begin
d = a;
end
endtask
endmodule
```

In the above code, the formal parameter `d` of task `t1` has a size of 1 bit, while the actual parameter `out1` has a size of 32 bits. This will result in a loss of 31 most significant bits of the actual parameter.

[Back to Top](#)

FNC_MS_MTYP

Short Message: *Task/function call argument %d is of wrong type (%s vs. %s).*

Severity	Warning
Description	An actual argument for a task/function call is found to have a type that is inconsistent with the formal argument in the task/function definition. This could lead to an incorrect computation within the task/function body.

The following code illustrates the occurrence of FNC_MS_MTYP.

```
module task_arg_wrong_type (a, out1);
  input a;
  output out1;
  integer out1;
  always @(a)
    t1(a, out1);

  task t1;
  input a;
  output out1;
  begin
    out1 = a;
  end
endtask
endmodule
```

In the above code, the output port of the task which is of type `reg` is mapped to a signal of type `integer` when the task is called inside a module. The formal and actual task / function arguments should have consistent types.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

FNC_NO_AVAC

Short Message: *Function '%s' is not assigned a value in some of the branches of conditional statement(s).*

Severity	Warning
Description	<p>A function that is not assigned a value in some branches of the conditional statement results in an unknown \times value. This unknown value might cause differences in simulation/synthesis results.</p> <p>To avoid this warning, ensure that the function returns a value in all the branches of the conditional statement.</p>

The following code illustrates the occurrence of FNC_NO_AVAC.

```
module mod_a(port_a,port_b,port_c);
  input port_a, port_b;
  output port_c;
  wire port_c;
  assign port_c = func_a (port_a,port_b);
  function func_a;
    input port_a,port_b;
    if (port_b)
      func_a = port_a;
    endfunction
endmodule
```

In the above code example, the function `func_a` is assigned a value in the `if` branch, but it is not assigned a value through the `else` branch. This might lead to simulation/synthesis result mismatches. Remodel your design and assign a value to the function in all the branches of the conditional statements.

For example, the following modified code eliminates this issue:

```
module mod_a(port_a,port_b,port_c);
  input port_a, port_b;
  output port_c;
  wire port_c;
  assign port_c = func_a (port_a,port_b);
  function func_a;
    input port_a,port_b;
    if (port_b)
```

JasperGold Superlint Checks Reference

Lint Checks

```
        func_a = port_a;
    else
        func_a = port_b;
    endfunction
endmodule
```

[Back to Top](#)

FNC_NO_USED

Short Message: *Function '%s' defined in %s '%s' is unused.*

Severity	Warning
Description	The specified function is defined, but no invocation of the function is found in the HDL model. This might be redundant/dead code and should be removed to clean up the design and avoid confusion.

The following code illustrates the occurrence of `FNC_NO_USED`.

```
module unused_ftn (in1, in2, out1, out2);
    input in1;
    input in2;
    output out1;
    output out2;
    reg out1;
    reg g1;
    function ftn1;
        input in1;
        input in2;
        begin
            ftn1 = in2;
        end
    endfunction

    always @(in1 or in2)
        out1 = in1 & in2;
        assign out2 = g1;
endmodule
```

JasperGold Superlint Checks Reference

Lint Checks

In the above code, the function `ftn1` is defined in the module `unused_ftn`, but it is not used.

[Back to Top](#)

FNC_NR_AVGV

Short Message: *Function '%s' in module '%s' assigns a value to global variable '%s'.*

Severity	Warning
Description	In a function block, a value should not be assigned to a global variable. Such assignments lead to redundant logic and degrade the circuit quality. Assignment of a value should be performed for the function name as a return value.

The following code illustrates the occurrence of FNC_NR_AVGV.

```
module mod_a(port_a);
output port_a;
integer int_a;
function func_a;
    input port_b;
    input port_c;
    if(port_b == 0)
        func_a = 1'b0;
    else
        port_a = port_c;
    endfunction
endmodule
```

In the above code, global variable `port_a` is assigned a value inside function `func_a`. Remodel the design to avoid this violation.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

FNC_NR_NARG

Short Message: *Task/function call has wrong number of arguments.*

Severity	Warning
Description	A task or function invocation is found to have an incorrect number of actual arguments when compared to the task or function definition. This could lead to an incorrect or unexpected computation within the task or function body. Modify the task or function call to specify the correct number of arguments.

The following code illustrates the occurrence of FNC_NR_NARG.

```
module test();
integer a;
wire [15:0] b;
wire c, d, e, f;

function multiply;
input a, b;
real a, b;
multiply = ((1.2 * a) * (b * 0.17)) * 5.1;
endfunction

initial begin
a = multiply(a,c,d);
end

endmodule
```

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

FNC_NR_UGLV

Short Message: *Function '%s' in module '%s' uses global variable '%s'.*

Severity	Warning
Description	The specified function references a variable that was not supplied as a function input or defined local to the function. This might lead to unpredictable change in the value of the variable. This global variable should be supplied as a function input.

The following code illustrates the occurrence of FNC_NR_UGLV.

```
module ftn_uses_global_var (in1, in2, out1, out2);
input in1;
input in2;
output out1;
output out2;
reg out1;
reg g1;
reg g2;
reg g3;
reg g4;
function ftn1;
input in1;
input in2;
reg t1;
reg t2;
reg t3;
reg t4;
begin
  case (g1)
    1'b0: t1 = in1;
    1'b1: t1 = in2;
  endcase
  case (in1)
    g1: t2 = 1'b0;
    g2: t2 = 1'b1;
  endcase
  if (g3)
```


JasperGold Superlint Checks Reference

Lint Checks

```
        t3 = in1;
    else
        t3 = in2;
        t4 = g4;
        ftn1 = t1 & t2 & t3 & t4;
    end
endfunction

always @(in1 or in2)
    out1 = ftn1(in1, in2);
assign out2 = g1;
endmodule
```

In the above code, variables `g1`, `g3`, and `g4` are global variables and are assigned values within the function `ftn1`.

[Back to Top](#)

IDX_MS_INDL

Short Message: *Port '%s' has index bounds mismatch between component declaration (%s) and entity declaration (%s).*

Severity	Error
Description	An index bounds mismatch is detected for the port in the component instantiation and declaration. This should be avoided as some emulators do not support it. Use of matching port index bounds in component instantiation and its declaration is a good coding practice.

The following code illustrates the occurrence of `IDX_MS_INDL`.

```
library ieee;
use ieee.std_logic_1164.all;

Entity bottom is
    Port (Inp_port : out std_logic_vector(2 downto 0);
          In1 : in std_logic_vector(2 downto 0)
        );
end bottom;
```

JasperGold Superlint Checks Reference

Lint Checks

```
architecture des of bottom is
begin
    Inp_port <= not(in1);
end des;

library ieee;
use ieee.std_logic_1164.all;
Entity top is
    Port (Inp_port : out std_logic_vector(2 downto 0);
          in2 : in std_logic_vector(2 downto 0)
        );
end top;

architecture behav of top is
component bottom
Port (Inp_port : out std_logic_vector(4 downto 2);
      In1 : in std_logic_vector(2 downto 0)
    );
end component;

signal clk:std_logic;
signal rec:std_logic_vector(2 downto 0);
begin
b1 : bottom port map (Inp_port=>Inp_port,in1=>in2);
process (clk)
begin
    Inp_port <= rec;
end process;
End behav;
```

In the above code, in the instantiation of bottom, 4 downto 2 is used for Inp_port. However, in declaration it is 2 downto 0.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

IDX_NR_DTTY

Short Message: *Variable '%s' used as index in expression '%s' should be of type int or genvar.*

Severity	Warning
Description	Indexing on left side of assignments must not use a 4-state data type that can be X.

The following code illustrates the occurrence of IDX_NR_DTTY.

```
module top();
    parameter p = 1'bx;
    reg [10:0] a;
    reg [10:0] b;
    logic c;
    logic [1:0]d;
    integer e;
    int ai;
    int i;

    always @*
    begin
        e = 9;
        b[c] = a[c];
        b[c + 1] = a[c + 1];
        b[p] = a[0];
        b[ai[i]] = a[1];
    end
endmodule
```

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

IFC_NO_FALW

Short Message: *The “if” statement specifying an asynchronous set/reset is not the first statement of the “always” block.*

Severity	Warning
Description	If the <code>if</code> statement specifying an asynchronous set/reset is not the first statement of the <code>always</code> block, logic synthesis cannot be performed. To avoid this issue, the <code>if</code> statement specifying the asynchronous set/reset signal must be the first statement of the <code>always</code> block.

The following code illustrates the occurrence of IFC_NO_FALW.

```
module test_0(clk,reset,port_a,port_b);
input clk;
input reset;
input port_a;
output port_b;
reg port_b;
always @ ( posedge clk or negedge reset )
begin
    port_b <= port_a;
    if ( !reset )
        port_b <= 1'b0;
end
endmodule
```

In the above code, the `if` statement specifying the asynchronous reset is not the first statement of the `always` block. As a result, logic synthesis cannot be performed. To avoid this issue, modify the code such that the `if` statement specifying the asynchronous reset signal is the first statement of the `always` block.

The following modified code eliminates this issue.

```
module test_0(clk,reset,port_a,port_b);
input clk;
input reset;
input port_a;
output port_b;
reg port_b;
always @ ( posedge clk or negedge reset )
begin
    if ( !reset )
        port_b <= 1'b0;
    port_b <= port_a;
end
endmodule
```

JasperGold Superlint Checks Reference

Lint Checks

```
begin
  if ( !reset )
    port_b <= 1'b0;
  else
    port_b <= port_a;
  end
endmodule
```

[Back to Top](#)

IFC_NR_DGEL

Short Message: *Ambiguous else statement in the nested if statement. It is recommended to enclose the inner if statement in a begin/end block.*

Severity	Warning
Description	A dangling else is a statement in which there is ambiguity in identifying the association of <code>else</code> keyword with the <code>if</code> statement. By default, the <code>else</code> keyword is branched with the innermost <code>if</code> statement.

The following code illustrates the occurrence of IFC_NR_DGEL.

```
module dangling_else (a, b, sel, sel2, out);
input a, b, sel, sel2;
output out;
reg out;
always @(a or b or sel or sel2)
begin
  if (sel)
    if (sel2)
      out = a;
  else
    out = b;
end
endmodule
```

In the above code, a dangling else statement is present in the module definition. In this code, the association of `else` with the first `if` statement is not apparent. To avoid this issue, the inner `if` block should be enclosed in a `begin/end` block. The following modified code eliminates the issue:

JasperGold Superlint Checks Reference

Lint Checks

```
module dangling_else (a, b, sel, sel2, out);
input a, b, sel, sel2;
output out;
reg out;
always @(a or b or sel or sel2)
begin
    if (sel)
        begin
            if (sel2)
                out = a;
            end
        else
            out = b;
        end
end
endmodule
```

[Back to Top](#)

INP_NO_USED

Short Message: *The Input/inout port '%s' defined in the %s '%s' is unused (neither read nor assigned).*

Severity	Warning
Description	The specified input port is defined, but no usage of the port is found in the specified scope. To avoid redundant code and avoid confusion, any unused port in the design should be removed. Presence of unused port may also prevent potential errors from being caught.

The following code illustrates the occurrence of INP_NO_USED.

```
module param_partsel (in1, out1);
input in1;
output [6:0] out1;
    parameter [3:0] p1 = 4'b0110;
    parameter [4:1] p2 = 4'b1010;
    assign out1[5:0] = {2'b10, p2[5], p1[2:0]};
endmodule
```

JasperGold Superlint Checks Reference

Lint Checks

In the above mentioned example, the primary input port `in1` is not used in the design description. Assign this input if required or remove it from the design.

[Back to Top](#)

INP_NR_UNRD

Short Message: *The input/inout port '%s' defined in the %s '%s' is unread, but assigned.*

Severity	Warning
Description	The specified port is unconnected (unread) but is assigned to some object in the specified scope. To avoid redundant code, prevent undefined states, and prevent any potential errors from being missed, any unread port should be properly read or, if not required, the port should be removed from the design description. This rule honors the synthesis off/on pragmas, which implies that a port is considered unread when it is read inside synthesis off/ on pragma and its declaration is outside the pragma.

The following code illustrates the occurrence of INP_NR_UNRD.

```
module mux(a_I,b_I,c_I,d_I,select_I,mux);
input a_I,b_I,select_I;
input c_I;
input d_I;
output mux;
wire wira;
reg mux;
assign d_I = wira;
always @ (a_I or b_I or select_I)
    case (select_I)
        1'b1 : mux = a_I;
        default : mux = b_I;
    endcase
endmodule
```

In the above code, the port `d_I` is assigned but not read. To prevent undefined conditions, read this port.

JasperGold Superlint Checks Reference

Lint Checks

[Back to Top](#)

INP_UC_INST

Short Message: *Input port '%s' of entity/module '%s' is being used inside architecture/module but not connected (either partially or completely) in its instance '%s'.*

Severity	Error
Description	<p>The specified input port is unconnected in its instance. Unconnected inputs are ones in which formal ports are used inside the module/architecture, but corresponding actual ports are either missing or connected to some unconnected local signal. To avoid the possibility of undefined states, the design must not contain unconnected inputs to instance. This helps in avoiding potential problems with back-end tools. In the case of a black-boxed module, it is assumed that formal input ports are being used inside the module.</p>
Associated parameter	<p>ignore_explicitly_unconnected_port=no yes</p> <p>The default value for this parameter is <code>no</code>.</p> <p>When the value of this parameter is set to <code>no</code>, <code>INP_UC_INST</code> is reported for explicitly unconnected input ports. When the value of this parameter is set to <code>yes</code>, <code>INP_UC_INST</code> is not reported for explicitly unconnected input ports.</p> <p>ignore_port_with_no_load=no yes</p> <p>The default value for this parameter is <code>no</code>.</p> <p>When the value of this parameter is set to <code>no</code>, <code>INP_UC_INST</code> is reported for input ports with no load. When the value of this parameter is set to <code>yes</code>, <code>INP_UC_INST</code> is not reported for unconnected input ports with no load.</p>

The following code illustrates the occurrence of `INP_UC_INST`.

```
module mod_a(port_a, port_b);
  input port_a;
  output port_b;
  mod_b inst_a(.port_a(), .port_b(port_b), .port_c());
endmodule
```


JasperGold Superlint Checks Reference

Lint Checks

```
module mod_b(port_a, port_b, port_c);
input port_a, port_c;
output port_b;
wire wir_b;
    assign port_b = port_a;
    assign wir_b = port_c;
endmodule
```

In the above code, a violation is reported for the following ports:

- port_a of instance inst_a because formal port port_a is used inside module mod_b, but the corresponding actual port is missing.
- port_c of instance inst_a because formal port port_c is used inside module mod_b, but the corresponding actual port is missing.

Remodel the design to avoid this violation.

For the given code, if the value of parameter `ignore_explicitly_unconnected_port` is set to `yes`, then `INP_UC_INST` is not reported for port_a and port_c of instance inst_a because port_a and port_c are explicitly left unconnected.

For the given code, if the value of parameter `ignore_port_with_no_load` is set to `yes`, then `INP_UC_INST` is not reported for port_c of instance inst_a because port_c has no load (driving an internal wire wir_b in module mod_b) and is unconnected.

[Back to Top](#)

INS_MS_PSIZ

Short Message: Port '%s' has size mismatch between module instantiation and declaration, '%s' bits at instantiation and '%s' bits at declaration.

Severity	Error
Description	Size mismatch is detected for the port between the module instantiation and declaration. This should be avoided as some emulators do not support such size mismatches. As a good coding practice, use same port sizes in module instantiation and declaration.

The following code illustrates the occurrence of `INS_MS_PSIZ`.

JasperGold Superlint Checks Reference

Lint Checks

```
module port_size_mismatch (data, out);
  input data;
  output out;
  submod m1 (.o(out), .d(data));
endmodule
```

```
module submod(o, d);
  output o;
  input [1:0] d;
  assign o = d[1];
endmodule
```

In the above example, the formal port `d` of module `submod` is 2 bits wide whereas in the instance `I1` of `submod` in module `port_size_mismatch`, the actual connected to formal port `d` is only 1-bit wide.

[Back to Top](#)

INS_NR_PODL

Short Message: *Port connections for instance '%s' of %s '%s' should be made by name rather than by positional ordered list.*

Severity	Warning
Description	Rather than relying on listing the actual port expressions for the module instance in the same order as the formal input/output/inout ports defined for the module, explicitly link formal to actual module port connections. This eliminates problems such as accidental module instance pin swaps, which could be introduced when the HDL description is modified and also improve overall design robustness and code readability.



This check is available for Verilog designs only.

JasperGold Superlint Checks Reference

Lint Checks

Parameter associated

check_tech_cells=yes

Setting this parameter to `yes` causes Superlint to issue `INS_NR_PODL` check for library cells such as modules with ``celldefine` statement in Verilog and VITAL design units in VHDL with `VITAL_LEVEL0&1` attributes.

By default Superlint reports this check for technology cells instantiated in the design. Set this parameter to `no` to switch off this check.

The following code illustrates the occurrence of `INS_NR_PODL`.

```
module testve_rtl041 (in_1,
    in_2,
    clk,
    out
);
input [2:0] in_1;
input [2:0] in_2;
input clk;
output [2:0] out;

wire temp, tmp;
reg [2:0] out;

m1 inst(in_1[0], clk, temp, tmp);
endmodule
module m1 (in, clk, out, out_1);

input in, clk;
output out, out_1;
reg out;
reg out_1;

always @ (clk)
begin
    out = in;
    out_1 = in;
end

endmodule
```

JasperGold Superlint Checks Reference

Lint Checks

In the above mentioned code, actual port expressions `in_1[0]`, `clk`, `temp` and `tmp` for instance `inst` are mapped to formal ports `in`, `clk`, `out` and `out_1`, respectively for module `m1` by positional association. Specify port connections using named association as illustrated below:

```
m1 inst(.in(in_1[0]), .clk(clk), .out(temp), .out_1(tmp) );
```

[Back to Top](#)

INS_NR_PTEX

Short Message: *%s is used in a port expression.*

Severity	Warning
Description	The actual port of the module should not be an integer or a constant. The port mapping to the formal port of the module should be modified to be of a type consistent with the module port declaration.
Parameter associated	<p>The following parameter controls the behavior of the INS_NR_PTEX check:</p> <pre>params INS_NR_PTEX {ignore_sized_literal = "no" "yes"}</pre> <p>The default is <code>no</code>.</p>

The following code illustrates the occurrence of INS_NR_PTEX

```
module mod_a (port_a, port_b);
input port_a;
output port_b;
mod_b inst_a(port_b, port_a, 16);

endmodule

module mod_b (port_a, port_b, port_c);
input port_b;
input port_c;
output port_a;
    assign port_a = port_b & port_c;
endmodule
```

JasperGold Superlint Checks Reference

Lint Checks

In the above code, integer value 16 is used to map to a module port. Make the signal mapped to the module port consistent with the declaration of the port.

[Back to Top](#)

INT_NR_PSBT

Short Message: *Bit/part select of %s variable '%s' encountered.*

Severity	Warning
Description	Bits of a variable of type integer or time should not be selected and used. The data types integer and time define a size of 32 bits. If only part of the variable is used, it leads to excess memory consumption. If usage of specific bits of such a variable is intentional, select the bits using mask and shift operations. Otherwise, use a <code>reg</code> variable of the appropriate size.

The following code illustrates the occurrence of INT_NR_PSBT.

```
module bitssel_int_time_var (out1, out2);
  reg [7:0] reg_a;
  integer i;
  for ( i = 0; i <= 7; i = i + 1 )
    begin
      case (i[2:1])
        2'b00 : reg_a[i] = 1'b0;
        2'b01 : reg_a[i] = 1'b1;
        2'b10 : reg_a[i] = 1'b1;
        2'b11 : reg_a[i] = 1'b0;
        default : reg_a[i] = 1'bx;
      endcase
    end
endmodule
```

In the above code, `i` is declared as an integer. However, only three least significant bits are used. To avoid this violation, declare `i` as a three-bit `reg` variable as shown below:

```
reg[2:0] i;
```

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

IOP_NR_UASG

Short Message: *The input/inout port '%s' defined in the %s '%s' is unassigned, but read*

Severity	Warning
Description	The specified port is not assigned. It is connected to an object indicating that the port has been read at least once. To avoid redundant code, prevent undefined states, and avoid any potential errors from being caught, any unassigned port should be properly assigned, or if not required, then the port should be removed from the design description. This rule honors the synthesis off/on pragmas. This implies that a port is considered unassigned when assignment is made inside synthesis off/on pragma and its declaration is outside the pragma.

The following code illustrates the occurrence of IOP_NR_UASG.

```
module mux(a_I,b_I,c_I,d_I,select_I,mux);
  inout a_I,b_I,select_I;
  input c_I;
  input d_I;
  output mux;
  wire wira;
  reg mux;
  assign d_I = wira;
  always @ (a_I or b_I or select_I)
  case (select_I)
    1'b1 : mux = a_I;
    default : mux = b_I;
  endcase
endmodule
```

In the above code, inout ports are read but not assigned anywhere. Assign these ports to prevent undefined conditions.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

LOP_NR_CTCE

Short Message: *Module '%s' contains a loop with a constant conditional expression.*

Severity	Warning
Description	The specified loop was found to have a constant termination condition/ expression. This will result in the loop being executed only once.

The following code illustrates the occurrence of LOP_NR_CTCE.

```
module const_while_loop_cond (clk, q1);
input clk;
output [15:0] q1;
reg [15:0] a;
integer i;
initial
begin
    i = 0;
    while ( 0 )
        a[i] = 1'b0;
end
endmodule
```

The while loop in the given example is the terminating loop in the design.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

LOP_NR_IDTY

Short Message: *Loop variable '%s' of the 'for' loop is not of recommended data type(s) (%s).*

Severity	Warning
Description	<p>If the loop variable is not of the recommended data type, there is a possibility that the value stored in the variable can be misinterpreted, causing incorrect results. The recommended data types are specified through the <code>allow_loopvar</code> parameter provided in the default rules file:</p> <pre>params LOP_NR_IDTY {allow_loopvar= integer:int:shortint:longint:reg:reg_signed}</pre> <p>By default, the loop variable can be of <code>integer</code>, <code>int</code>, <code>shortint</code>, <code>longint</code>, <code>reg</code>, or <code>reg_signed</code> data type. However, you can modify the recommended data types by modifying the <code>allow_loopvar</code> parameter.</p> <p>A violation is reported if the loop variable is not of a data type specified by this parameter.</p>

The following code illustrates the occurrence of LOP_NR_IDTY.

```
module mod_a(input [7:0] port_a, output reg [7:0] port_b);
  logic [31:0] i;
  always @(port_a)
  begin
    port_b = 8'd0;
    for ( i=7; i>=0; i=i-1)
      port_b[i] = port_a[i];
  end
endmodule
```

In the above code, loop variable `i` is of type `logic`, which is not a recommended data type. As a result, a violation is reported. Remodel the design such that loop variable `i` is of a type specified in the `allow_loopvar` parameter in the rules file.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

LOP_NR_MLPV

Short Message: *Value of loop variable '%s' modified within the loop.*

Severity	Error
Description	To avoid errors during synthesis, the value of a loop variable should not be modified within the loop.

The following code illustrates the occurrence of LOP_NR_MLPV.

```
module mod_a(port_a);
input port_a;
integer int_a;
reg [0:7] reg_a;
always@(port_a)
begin
    for(int_a = 0; int_a < 4; int_a = int_a + 1)
        begin
            reg_a[2] = 1'b1;
            int_a = 12;
        end
    end
end
endmodule
```

In the above code, loop variable `int_a` is modified within the for loop. This might lead to errors during synthesis. To avoid this violation, do not change the value of `int_a` inside the for loop.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

LOP_NR_RPVR

Short Message: *Variable '%l' is used repetitively in the statement inside loop body.*

Severity	Warning
Description	If a variable is used repeatedly in a statement inside the loop body, a long and complex chain like structure might be generated, resulting in degradation of circuit quality. Such use should be avoided.

The following code illustrates the occurrence of LOP_NR_RPVR.

```
module test(out, port_a);
  input [7:0]port_a;
  output out;
  reg out;
  always@(port_a)
  begin
    out = port_a[0]^port_a[1];
    for(int i =2; i<=7; i++)
    begin
      out =out^port_a[i]; //issue LOP_NR_RPVR
    end
  end
endmodule
```

In the above code, variable `out` is used repeatedly in the statement inside the `for` loop. As a result, a violation is reported. Remodel the design to avoid this issue.

[Back to Top](#)

MAC_NR_DMUL

Short Message: *Macro '%s' is defined with same definition.*

Severity	Information
Description	The specified macro is defined in multiple files and has the same value. As a good coding practice, macros should be defined at a common location.

The following code illustrates the occurrence of MAC_NR_DMUL.

```
top.v
`define W 1
module top;
  reg [`W:0] sig;
  bot but();
endmodule
```

```
bot.v
`define W 1
module bot;
  reg [`W:0] sig;
endmodule
```

In the above code, macro `W` is defined in both the file `top.v` and `bot.v`. To avoid this violation, use this macro at a common location.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

MOD_NO_TMSL

Short Message: *Timescale is missing for the module '%s'.*

Severity	Warning
Description	<p>If the timescale is not specified for a module that includes a specify block, the tool automatically sets the default timescale, which might not be intended. This might lead to unexpected results.</p> <p>Even though timescale can be specified during a Superlint run using the <code>xmelab</code> command-line option <code>-timescale</code>; it is recommended that timescale be specified explicitly using the <code>`timescale</code> directive.</p>

The following code illustrates the occurrence of MOD_NO_TMSL.

```
module test(out, port_a);
input [3:0]port_a;
output [3:0]out;
reg [3:0]reg_a;
    specify
        (port_a => out) = 9;
    endspecify
assign out = reg_a;
endmodule
```

In the above code, timescale is not specified for module test that includes a specify block. As a result, a violation is reported. Specify a timescale for the module using the ``timescale` directive to avoid this issue.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

MOD_NR_ALLD

Short Message: *In design-unit '%s', unsynthesizable allocator/deallocate is encountered.*

Severity	Warning
Description	Allocators are not supported and will be ignored whenever encountered. VHDL built-in function DEALLOCATE will also be ignored. They are not supported even by synthesis tools. Remodel your design.

The following code illustrates the occurrence of MOD_NR_ALLD.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use std.textio.all;

entity record_type is
end entity record_type;

architecture record_type_arch of record_type is
    --type AccBV is access Bit_Vector(7 downto 0);
begin
    P0: process
        --variable Ptr1, Ptr2 : AccBV;
        variable l      : LINE ;
    begin
        -- Ptr1 := new Bit_Vector(7 downto 0);
        --Ptr2 := Ptr1;
        DEALLOCATE(P=>l);
    end process P0;
end architecture record_type_arch;
```

The DEALLOCATE function in the above code will be ignored. Remodel your design to avoid this warning.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

MOD_NR_CASX

Short Message: *'casex' statement used in module '%s'.*

Severity	Warning
Description	The Verilog <code>casex</code> statement treats the <code>x</code> and <code>z</code> states as do not cares in synthesis, which can lead to differences in the pre-synthesis and post-synthesis simulation results. To avoid this warning, use <code>case</code> statement instead of <code>casex</code> statement.

The following code illustrates the occurrence of MOD_NR_CASX.

```
module mod_a();
function func_a;
input[7:0] port_a;
reg reg_a;
begin
    reg_a = 0;
    casex(port_a[4:2])
        3`bxx0: if(port_a[7:6] == 3) reg_a = 1;
        3`bxx1: if(port_a[2:1] == 2) reg_a = 1;
    endcase
end
endfunction
endmodule
```

In the above code, `casex` statement is used. To avoid this warning, replace the `casex` statement with the `case` statement.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

MOD_NR_CASZ

Short Message: *Casez statement used in the design unit %s.*

Severity	Warning
Description	The use of <code>casez</code> statements can potentially cause simulation synthesis mismatch.

The following code illustrates the occurrence of MOD_NR_CASZ.

```
module mod_a(en, port_a, port_b, port_c, port_d);
input en;
input[1:0] port_a;
output reg port_b, port_c, port_d;
always @ (en or port_a)
begin
    {port_b, port_c, port_d} = 3'b0;
    casez({port_a, en})
        3'b101: port_b = 1'b1;
        3'b111: port_c = 1'b1;
        3'b0?1: port_d = 1'b1;
    endcase
end
endmodule
```

In the given code, an erroneous match can occur if one of the bits of the selector is set as `z`. Therefore, a violation is reported if the `casez` statement is used in a design unit. To avoid this issue, remove the `casez` statement and recompile.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

MOD_NR_PGAT

Short Message: *Gate instances are not expected in an RTL design.*

Severity	Warning
Description	An RTL HDL model is assumed to consist of behavioral HDL statements. The specified primitive gate instance was not expected to be encountered in this modeling style.

The following code illustrates the occurrence of MOD_NR_PGAT.

```
module outp_asgned_via_gate (a, out1);  
    input a;  
    output out1;  
    buf (out1,a);  
endmodule
```

In the above example, the highlighted code shows the use of a primitive gate in the design.

[Back to Top](#)

MOD_NR_PINS

Short Message: *Primitive instances are not expected in an RTL design.*

Severity	Error
Description	An RTL HDL model is assumed to consist of behavioral HDL statements. The specified primitive UDP instance was not expected to be encountered in this modeling style.

The following code illustrates the occurrence of MOD_NR_PINS.

```
module prim_inst_unexpected (clk, data, out1);  
    input clk;  
    input data;  
    output out1;  
    dff d1 (out1, clk, data);  
endmodule
```


JasperGold Superlint Checks Reference

Lint Checks

```
primitive dff (q, clk, d);
output q;
input clk, d;
reg q;
table
    // clk d q q+
    (01) 0 : ? : 0 ;
    (01) 1 : ? : 1 ;
    (0?) 0 : 0 : 0 ;
    (0?) 1 : 1 : 1 ;
    // ignore negative edge of clk
    (?0) ? : ? : - ;
    // ignore data changes on steady clock
    ? (??) : ? : - ;
endtable
endprimitive
```

In the above example, the boldfaced code shows the instance of a primitive in the design.

[Back to Top](#)

MOD_NR_PLIF

Short Message: *PLI 1.0 function %s in module '%s' is ignored.*

Severity	Warning
Description	A PLI function encountered in an expression is not supported because it is not synthesizable.

The following code illustrates the occurrence of MOD_NR_PLIF.

```
module test () ;
reg[3:0] reg_1;
reg[3:0] reg_2;
initial
    begin:initblock1
        $tf_iputp_test(reg_1, reg_2);
    end
```

JasperGold Superlint Checks Reference

Lint Checks

```
endmodule
```

In the above code, PLI function `$tf_inputp_test` is used in the module test. PLI functions in a module are ignored.

[Back to Top](#)

MOD_NR_SYTS

Short Message: *System task '%s' in module '%s' is ignored.*

Severity	Warning
Description	<p>System tasks are generally non-synthesizable and are ignored.</p> <p>MOD_NR_SYTS will be ignored for the functions that are specified in the <code>pattern</code> parameter below.</p> <pre>params MOD_NR_SYTS {pattern=""}</pre>

The following code illustrates the occurrence of MOD_NR_SYTS.

```
module assign_var_select (in1, in2);
input in1;
input in2;
time timeStamp;
wire i;
always @(i)
begin
    timeStamp = $time;
end
endmodule
```

In the above code, there is a system task within the module. It will be ignored.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

MOD_NR_UNGN

Short Message: *Unnamed generate block found in module "%s".*

Severity	Warning
Description	The use of unnamed generate blocks might cause simulation problems. Add names for each generate block.

The following code illustrates the occurrence of MOD_NR_UNGN:

```
module test(input wire port_a, output reg port_b);
  generate
    if (1) begin: gen_a
      if (1) begin
        mod_a inst_a (port_a, port_b);
      end
    end
  endgenerate
endmodule
```

In the given code, a violation is reported as an unnamed generate block has been used in the module test.

[Back to Top](#)

OPR_NR_LOSD

Short Message: *The result of the "%s" operation between constants in expression "%s" leads to truncation of bits and loss of data in module/design-unit %s. The bits getting lost are '%s'.*

Severity	Warning
Description	The specified operation between constants leads to a truncation of bits and loss of data. There is loss of data if any of the truncated bits is determined to be 1. Remodel the design or adjust the size of the LHS operand to avoid data loss.

JasperGold Superlint Checks Reference

Lint Checks

Associated parameters	<pre>params OPR_NR_LOSD:OPR_NR_TRNB {disable_on_port_declaration="no yes"}</pre> <p>The default value for this parameter is <code>no</code>. When the value of this parameter is set to <code>no</code>, the check applies to all the entries in the design, including port declarations. When the value of this parameter is set to <code>yes</code>, the check will not apply to the port declaration.</p>
------------------------------	--

The following code illustrates the occurrence of `OPR_NR_LOSD`.

```
module mod_a(port_a,port_b);
output [2:0] port_a;
output [2:0] port_b;
wire[2:0] port_a;
wire[2:0] port_b;
    assign port_a = 4'b1111 + 4'b0111;
    assign port_b = 3'b111 * 3'b011;
endmodule
```

In the above code, `port_a` and `port_b` are 3-bit variables, while the result of the following addition and multiplication operations on the RHS is 5-bit.

```
assign port_a = 4'b1111 + 4'b0111;

assign port_b = 3'b111 * 3'b011;
```

This results in truncation of most significant bits and loss of data. To avoid data loss, remodel the design or adjust the size of `port_a` and `port_b`.

[Back to Top](#)

OPR_NR_REAL

Short Message: *Real operand used in logical comparison.*

Severity	Warning
Description	A real operand has been encountered in a logical comparison. Real values are often problematic for synthesis tools. Replace this value with an integer/bit value, which is more acceptable for synthesis tools.

The following code illustrates the occurrence of `OPR_NR_REAL`.

JasperGold Superlint Checks Reference

Lint Checks

```
library ieee;
use ieee.std_logic_1164.all;
entity TEST1 is
    port(out1,out2 :out std_logic);
end;

architecture ARCH_TEST1 of TEST1 is
    signal b,a,r1: real;
begin
    process (b)
    begin
        r1 <= b;
    end process;

    process (a, r1)
    begin
        if (a < 1.0) then
            out1 <= '1';
        end if;

        if (a >= r1) then
            out2 <= '0';
        end if;
    end process;
end ARCH_TEST1 ;
```

In the example above, a real value (1.0) is used. Use an integer instead.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

OPR_NR_TRNB

Short Message: *The result of the "%s" operation between constants in expression "%s" leads to truncation of bits but without loss of data in module/design-unit %s. The bits getting lost are '%s'.*

Severity	Warning
Description	The specified operation between constants leads to a truncation of bits. There is no loss of data if the truncated bits are determined to be zeros. Even though there is no loss of data, it is recommended that you remodel the design or adjust the size of the LHS operand to avoid bit loss.
Associated parameters	<code>params OPR_NR_LOSD:OPR_NR_TRNB</code> <code>{disable_on_port_declaration="no yes"}</code> The default value for this parameter is <code>no</code> . When the value of this parameter is set to <code>no</code> , the check applies to all the entries in the design, including port declarations. When the value of this parameter is set to <code>yes</code> , the check will not apply to the port declaration.

The following code illustrates the occurrence of OPR_NR_TRNB.

```
module mod_a(port_a,port_b);  
  output [2:0] port_a;  
  output [2:0] port_b;  
  wire[2:0] port_a;  
  wire[2:0] port_b;  
  assign port_a = 4'b0011 + 4'b0011;  
endmodule
```

In the above code, `port_a` is a 3-bit variable, while the result of the following addition operation is 0110, which is 4-bit.

```
assign port_a = 4'b0011 + 4'b0011;
```

This results in a truncation of bits. In this example, even though there is no data loss, it is recommended that you remodel the design or adjust the size of `port_a` so that there is no bit loss.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

OPR_NR_UCMP

Short Message: *Unequal length operands in equality operator encountered (padding produces incorrect result) in module/design-unit %s. LHS operand is %d bits, RHS operand is %d bit.*

Severity	Warning
Description	Operands of different widths are padded by simulators in order to perform comparisons of equal width operands. This is required to produce a boolean comparison result. In the situation identified, most of the significant bits of the larger operand are non-zero such that when the smaller operand is left-padded with zeros, the comparison operation will always be evaluated to FALSE for equality, TRUE for inequality. Modify this expression to use comparison operands of equal widths so that the desired padding of the operands is used to generate the expected comparison result.
Associated parameters	warn_unequal_length=yes no The default value of this parameter is <code>yes</code> . When <code>warn_unequal_length</code> is set to <code>yes</code> , this check is reported.

The following code illustrates the occurrence of OPR_NR_UCMP.

```
module uneq_length_cmp_op (a, b, c, d, out1, out2);
input [3:0] a;
input [1:0] b;
input [1:0] c;
input [4:0] d;
output out1;
output out2;
reg out1;
reg out2;
always @(a or b)
begin if (a < b) out1 = 1'b1;
      if (a == b) out2 = 1'b1;
      if (b == c) out2 = 1'b1;
      if (a <= c) out1 = 1'b0;
      if (d > c) out2 = 1'b0;
      if (d >= a) out1 = 1'b0;
      if (c != d) out2 = 1'b1;
end
```

JasperGold Superlint Checks Reference

Lint Checks

```
end  
endmodule
```

In the above code, `a` is defined as 4-bit while `b` as 2-bit. The boldfaced line of code shows a comparison between `a` and `b` which are of unequal size.

[Back to Top](#)

OPR_NR_UEOP

Short Message: *Unequal length operand in bit/arithmetic operator %s in module/design-unit %s. LHS operand '%s' is %s bits, RHS operand '%s' is %s bits.*

Severity	Warning
Description	<p>Use of unequal length operands in arithmetic and bit-wise operations might lead to unintended results. This is because operands of different widths are padded by simulators to generate a result with a width as defined by the sizing rules of the HDL language. To avoid unintended results, do not use unequal length operands in arithmetic and bit-wise operations.</p> <p>By default, a violation is reported whenever bit-wise or arithmetic (addition or subtraction) operation is performed between unequal length operands. However, you can control the behavior of this check for addition and subtraction operations using the following parameters in the default rules file located at <install_dir>/etc/res/rtltds/rules/superlint.def:</p> <pre>params OPR_NR_UEOP {ignore_addition="no" "yes"} params OPR_NR_UEOP {ignore_subtraction="no" "yes"}</pre> <p>The default value of these parameters is <code>no</code>. As a result, the check is reported if addition and subtraction operation is performed between operands of different lengths. When the value is set to <code>yes</code>, the check is ignored for respective operators.</p>

The following code illustrates the occurrence of OPR_NR_UEOP.

```
module mod_a(port_a,port_b,port_c,port_d);  
input [3:0] port_a;  
input [1:0] port_b;  
output [3:0] port_c;
```


JasperGold Superlint Checks Reference

Lint Checks

```
output [3:0] port_d;
reg [3:0] port_c;
reg [3:0] port_d;
always @(port_a or port_b)
begin
    port_c = port_a & port_b; //operation between unequal length operands
    port_d = port_a | port_b; //operation between unequal length operands
end
endmodule
```

In the above code, bit-wise operation is performed between operands of different lengths. As a result, a violation is reported. Remodel the design to use equal length operands in bit-wise operations.

[Back to Top](#)

OPR_NR_UREL

Short Message: *Unequal length operands in relational operator (padding produces incorrect result) in module/design-unit %s -- LHS operand is %d bits, RHS operand is %d bits.*

Severity	Warning
Description	Operands of different widths are padded by simulators in order to perform comparisons of equal width operands. This is required to produce a boolean comparison result. In the situation identified, the most significant bits of the larger operand could be non-zero such that when the smaller operand is left-padded with zeros, the comparison operation will always be evaluated to <code>false</code> . Modify this expression to use comparison operands of equal widths so that the desired padding of the operands is used to generate the expected comparison result.
Parameter associated	warn_unequal_length=yes no The default value of this parameter is <code>yes</code> . When <code>warn_unequal_length</code> is set to <code>yes</code> , this check is reported.

The following code illustrates the occurrence of OPR_NR_UREL.

```
module uneq_length_cmp_op (a, b, c, d, out1, out2);
input [3:0] a;
```

JasperGold Superlint Checks Reference

Lint Checks

```
input [1:0] b;
input [1:0] c;
input [4:0] d;
output out1;
output out2;
reg out1;
reg out2;
always @(a or b)
begin if (a < b) out1 = 1'b1;
    if (a == b) out2 = 1'b1;
    if (b == c) out2 = 1'b1;
    if (a <= c) out1 = 1'b0;
    if (d > c) out2 = 1'b0;
    if (d >= a) out1 = 1'b0;
    if (c != d) out2 = 1'b1;
end
endmodule
```

In the above code, `a` is of 4 bits while `b` is of 2 bits. A 2-bit variable is being compared to a 4-bit variable. This is a case of size mismatch. You need to re-declare variables of equal sizes.

[Back to Top](#)

OTP_NO_FDRV

Short Message: *Output/inout '%s' is not fully driven in the module '%s'.*

Severity	Warning
Description	The specified primary output is not fully driven in the specified module. The primary output of the design should always be driven completely. Undriven bits of primary output might be because of some error in design modeling or because the bit is not required in the design. The specified bit of primary port should be used if required, and if not required, it should be removed from the design.

JasperGold Superlint Checks Reference

Lint Checks

Associated parameter

params OTP_NO_FDRV {top_only="no" | "yes"}

When this parameter is set to `top_only=yes`, the tool checks for OTP_NO_FDRV for TOP level modules only. When this parameter is set to `no`, the tool checks for OTP_NO_FDRV for all the modules. The default is `no`.

The following code illustrates the occurrence of OTP_NO_FDRV:

```
module test(output reg [2:0] a);
  reg reset;
  always @*
    a[0+:1] = reset;
endmodule
```

In the above code, output port `a` is not driven completely. In this case, `a[2:1]` is undriven.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

OTP_NR_ASYA

Short Message: *Output port '%s' is assigned asynchronously.*

Severity	Warning
Description	The output port in the design is assigned asynchronously. Output port assignments should be made through a clocked assignment.
Associated parameters	<p>This check is controlled by the following parameters in the default rules file:</p> <pre>params OTP_NR_ASYA {physical_block_only= "no" "yes"}</pre> <p>The default value of this parameter is <code>no</code>, and in this case, <code>OTP_NR_ASYA</code> is reported for all blocks. When the value of this parameter is set to <code>yes</code>, the check is reported only for physical blocks.</p> <p>This parameter defines the conditions to treat an instance as logic.</p> <pre>params OTP_NR_ASYA {register_output_ports="relax" "strict"}</pre> <p>When the value of this parameter is set to the default value <code>relax</code>, then the check is not issued if the output port is registered inside an instance. When the value of <code>register_output_ports</code> is set to <code>strict</code>, then <code>OTP_NR_ASYA</code> is issued if the output port is not registered before being connected to an instance.</p>

The following code illustrates the occurrence of `OTP_NR_ASYA`.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity OperatorType is
  generic (sept : integer := 7 ;
    trois : integer := 3);
  port (sort : out unsigned (sept downto 0);
    in1 : in unsigned (sept downto 0);
    ena1 : in unsigned (trois downto 0);
    ena2 : in unsigned (trois downto 0));
end OperatorType;
```

JasperGold Superlint Checks Reference

Lint Checks

```
architecture rtl_oper of OperatorType is
begin
  p0:process ( in1, ena1, ena2)
  begin
    if ( in1 = ena1 & ena2 ) then
      sort <= ena1 & ena2;
    else
      sort <= ena1 & ena2;
    end if;

    if ((in1 = ena1) and (in1 = ena2)) then
      sort <= ena1 & ena2;
    elsif ( in1 = ena1 & ena2 ) then
      sort <= ena1 & ena2;
    end if;

    case ( in1 = ena1 & ena2 ) is
      when true => sort <= "0000";
      when false => sort <= "1111";
    end case;
  end process p0;
end rtl_oper;
```

In the above code, output port `sort` is assigned without any clocked assignment. To avoid this warning, consider making an output port assignment through a clocked assignment.

[Back to Top](#)

OTP_NR_TSUP

Short Message: *The output/inout '%s' is tied to supply0/supply1.*

Severity	Warning
Description	An output/inout was found to be driven by a constant supply0/supply1 value. Review this assignment to be sure that this was the intention in this design.

The following code illustrates the occurrence of OTP_NR_TSUP.

```
module output_tied_to_supply (a, out1, out2, out3, out4);
```

JasperGold Superlint Checks Reference

Lint Checks

```
input a;
output out1;
output out2;
inout out3;
inout out4;
supply0 vss;
supply1 vdd;
    assign out1 = vdd;
    assign out2 = vss;
    assign out3 = vdd;
    assign out4 = vss;
endmodule
```

In the above mentioned example, the output `out1` and the inout `out3` are tied to constant `supply1` whereas the output `out2` and the inout `out4` are tied to constant `supply0`.

[Back to Top](#)

OTP_NR_UDRV

Short Message: *Primary output/inout '%s' is not driven in the module '%s'.*

Severity	Warning
Description	The specified primary output is not driven in the specified module, or all assignments to it have been ignored. The primary output of the design should always be driven. Undriven primary output might be because of some error in design modeling or because the port is not required in the design. The primary port should be used if required, and if not required, it should be removed from the design.
Associated parameter	<pre>params OTP_NR_UDRV {top_only="no" "yes"}</pre> <p>When this parameter is set to <code>top_only=yes</code>, the tool checks for <code>OTP_NR_UDRV</code> for TOP level modules only. When this parameter is set to <code>no</code>, the tool checks for <code>OTP_NR_UDRV</code> for all the modules. The default is <code>no</code>.</p>

The following code illustrates the occurrence of `OTP_NR_UDRV`.

```
module mod_a(clk,rst,port_a,port_b,port_c,port_d);
input clk,rst,port_a;
```

JasperGold Superlint Checks Reference

Lint Checks

```
input [0:1] port_b;
output [0:1] port_c;
output port_d;
reg [0:1] port_c;
always @ (clk or rst or port_a)
begin
    if (rst == 1'b1)
        port_c = 2'b00;
    else if (port_a)
        port_c = 2'b01;
    else if (clk)
        port_c = port_b;
end
endmodule
```

In the above mentioned example, the primary output `port_d` is not driven in the module `mod_a`. Use this output if required; otherwise, remove it from the design.

[Back to Top](#)

OTP_UC_INST

Short Message: *Port '%s' (which is being used as an output) of entity/module '%s' is being driven inside the design, but not connected (either partially or completely) in its instance '%s'.*

Severity	Warning
Description	The specified output port is unconnected in its instance. Unconnected outputs are ones in which formal ports are driven inside the module/architecture, but corresponding actual ports are either missing or connected to some unconnected local signal. Failure to make a connection could be an oversight. To avoid potential problems with back-end tools, design must not contain unconnected outputs to instance. In the case of a black-boxed module, it is assumed that formal ports are being driven inside the module.

JasperGold Superlint Checks Reference

Lint Checks

Associated parameters	ignore_port_with_no_load=yes no
	The default value for this parameter is <code>no</code> . When the value of this parameter is set to <code>no</code> , Superlint reports <code>OTP_UC_INST</code> for output ports that have no load. When the value of this parameter is set to <code>yes</code> , Superlint does not report <code>OTP_UC_INST</code> for output ports that have no load.
	ignore_explicitly_unconnected_port=no yes
	The default value for this parameter is <code>no</code> . When the value of this parameter is set to <code>no</code> , <code>OTP_UC_INST</code> is reported for explicitly unconnected output ports. When the value of this parameter is set to <code>yes</code> , <code>OTP_UC_INST</code> is not reported for explicitly unconnected output ports.

The following code illustrates the occurrence of `OTP_UC_INST`.

```
module mod_a(port_a, port_b);
input port_a;
output port_b;
wire wir_a;
  mod_b inst_a(.port_a(port_a), .port_b(), .port_c(wir_a));
endmodule

module mod_b(port_a, port_b, port_c);
input port_a;
output port_b, port_c;
wire wir_b;
  assign port_c = port_a;
  assign port_b = port_a & wir_b;
endmodule
```

In the above code, a violation is reported for the following ports:

`port_b` of instance `inst_a` because formal port `port_b` is driven inside module `mod_b`, but the corresponding actual port is missing.

`port_c` of instance `inst_a` because formal port `port_c` is driven inside module `mod_b`, but the corresponding actual port is connected to an unconnected wire `wir_a`.

Remodel the design to avoid this violation.

JasperGold Superlint Checks Reference

Lint Checks

For the given code, if the value of parameter `ignore_explicitly_unconnected_port` is set to `yes`, then `OTP_UC_INST` is not reported for `port_b` of instance `inst_a` because `port_b` is explicitly left unconnected.

[Back to Top](#)

PAR_MS_SDAS

Short Message: *Parameter '%s' has a size mismatch between its declaration and its value assigned.*

Severity	Warning
Description	Parameter value overridden at the time of instantiation has a size mismatch between module instantiation and declaration. This can lead to padding or truncation of some bits. This should be avoided to prevent ambiguous results.

The following code illustrates the occurrence of PAR_MS_SDAS.

```
module mod_a(port_a, port_b);
    input port_a;
    output port_b;
    wire wir_a;
    mod_b #(.param_a (4'b1111)) inst_b(port_a, port_b, port_c);
endmodule

module mod_b(port_a, port_b, port_c);
    input port_a;
    output port_b, port_c;
    parameter param_a = 2'b0;
endmodule
```

In the above code example, parameter `a` is assigned a 4-bit value at the time of instantiation, which is more than the actual size of the parameter that is 2. Avoid such usage to prevent ambiguous results.

The following modified code eliminates this issue:

```
module mod_a(port_a, port_b);
    input port_a;
    output port_b;
```

JasperGold Superlint Checks Reference

Lint Checks

```
wire wir_a;
mod_b #(.param_a(2'b11)) inst_b(port_a, port_b, port_c);
endmodule
```

```
module mod_b(port_a, port_b, port_c);
  input port_a;
  output port_b, port_c;
  parameter param_a = 2'b0;
endmodule
```

[Back to Top](#)

PRO_NR_WAIT

Short Message: *Sensitivity lists should be used instead of wait statements.*

Severity	Warning
Description	In the design wait statements have been used. To synchronize simulation events and facilitate synthesis, use sensitivity lists instead of wait statements.

The following code illustrates the occurrence of PRO_NR_WAIT.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Stimulisand2 is
  port (out1 :in std_logic;
        in1  :out std_logic;
        in2  :out std_logic);
end Stimulisand2;

architecture rtl_simul of Stimulisand2 is
begin
  stimulis:process
  begin
    in1<='0';
    in2<='0';
```

JasperGold Superlint Checks Reference

Lint Checks

```
in1<='0';
wait for 30 ns;
in2<='1';
wait for 30 ns;
in1<='1';
wait for 30 ns;
in2<='1';
wait for 30 ns;
end process stimulus;
end rtl_simul;
```

In the above code, wait statements are used within the process, `stimulus`.

[Back to Top](#)

PRT_NR_DFRG

Short Message: *Port '%s' with range (%d to %d) is re-declared with a different range (%d to %d).*

Severity	Warning
Description	The input/output/inout port declaration in a Verilog module is inconsistent with a subsequent wire or register declaration. Verilog simulation allows for this difference and used the later declaration as the final declaration of the variable; however, for consistency, the port and wire or register should be declared with the same range.

The following code illustrates the occurrence of PRT_NR_DFRG.

```
module port_redecl_w_diff_range (c, out1);
  input [1:0] c;
  output [3:0] out1;
  reg [1:0] out1;
  always @(c)
    out1 = c;
endmodule
```

In the above code, `out1` is declared twice with different ranges, first as a 4-bit range and then as a 2-bit range. The boldfaced lines of code clearly illustrate this problem. Redefine `out1` either as 4-bit or as 2-bit to overcome the problem.

JasperGold Superlint Checks Reference

Lint Checks

[Back to Top](#)

PRT_NR_IOPT

Short Message: *Port '%s' is of type inout.*

Severity	Warning
Description	<p>Ports of type inout are bidirectional and might lead to contention problems. To avoid contention problems, use ports of type input or output. Avoiding bi-directional ports also eases synthesis and test insertion. By default, this check is reported for all the inout ports in the design. However, you can control the behavior of this check using the following parameter in the default rules file:</p> <pre>params PRT_NR_IOPT {iopnta_control=all top_only sub_only}</pre> <p>The default value for this parameter is <code>all</code>.</p> <p>When the value of this parameter is set to <code>top_only</code>, the check applies to inout ports of only the top-level module/design unit in the design.</p> <p>When the value of this parameter is set to <code>sub_only</code>, the check applies to inout ports of all modules/design-units other than the top-level module/design-unit in the design.</p>

The following code illustrates the occurrence of PRT_NR_IOPT.

```
module mod_a (port_a, port_b, port_c);
  input port_a, port_b;
  inout port_c;
  assign port_c = port_a & port_b;
endmodule
```

In the above code, the port `port_c` is of type `inout`. As a result, a violation is reported. To avoid this violation, change the type of this port to `output`.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

PRT_UC_INST

Short Message: *%s port '%s' defined in design-unit '%s' is not connected in its instance '%s'.*

Severity	Warning
Description	<p>The number of actual arguments found in the specified module instantiation does not match the number of input/output/inout formal ports found in the definition of the module. Presence of unconnected ports leads to the possibility of undefined states inside the units and also potential problems with the backend tools. Unconnected ports might also result in unexpected simulation/synthesis results.</p> <p>By default Superlint reports this message for unconnected output ports connected by name. Set the following parameter to <code>yes</code> to suppress flagging of this message for all the unconnected output ports connected by name.</p> <pre>params PRT_UC_INST allow_explicitly_unconnected=no yes</pre>

The following code illustrates the occurrence of PRT_UC_INST.

```
module neg_CUVWSP_unconn_port (sum, carry, in0, in1, in2, in3);
output [11:0] sum;
input [11:0] carry;
input [11:0] in0;
input [11:0] in1;
input [11:0] in2;
input [11:0] in3;
addmod top_adder 2( .sum (sum[11:0]), .carry (carry[11:0]), .cout (), .in0
(in0[11:0]), .in1 (in1[11:0]),
                    .in2 (in2[11:0]), .in3 (in3[11:0]), .cin (1'b0) );
endmodule

module addmod (cout, carry, sum, in3, in2, in1, in0, cin);
output [11:0] sum;
output cout;
input [11:0] carry;
input [11:0] in0;
input [11:0] in1;
```

JasperGold Superlint Checks Reference

Lint Checks

```
input [11:0] in2;
input [11:0] in3;
input cin;
    assign cout = 1'b1;
    assign sum = in0 + in1 + in2 + in3 + carry;
endmodule
```

In the above code, the port `cout` of the module `addmod` is not connected to an actual port expression in its instance `top_adder_2`.

[Back to Top](#)

REG_NO_READ

Short Message: *Local register variable '%s' is not read, but is assigned at least once in %s '%s'.*

Severity	Warning
Description	The specified register is not connected (unread); however, it is assigned to some object in the specified module. To avoid redundant code, prevent undefined states, and prevent any potential errors from being missed, any unread register should be properly read/used or, if not required, the register should be removed from the design description. This rule honors the synthesis off/on pragmas. This implies that a register is considered unread when it is read inside synthesis off/on pragma and its declaration is outside the pragma.

The following code illustrates the occurrence of `REG_NO_READ`.

```
module unread_reg (in1, out1);
input in1;
output out1;
reg out1;
reg g1;
always @(in1)
begin
    g1 <= in1;
    out1 <= in1;
end
endmodule
```

JasperGold Superlint Checks Reference

Lint Checks

In the above code, register `g1` is assigned but is not being read. If `g1` is not required, it should be removed from the HDL.

[Back to Top](#)

REG_NR_MBNT

Short Message: *The notifier in timing check '%s' cannot be greater than 1 bit.*

Severity	Warning
Description	The notifier used in any timing check cannot be of size greater than 1 bit. A notifier is always a 1-bit register according to Verilog coding. When a timing violation occurs, the notifier will "toggle" from 0 to 1 and vice versa. This toggling is not possible on a multi-bit register. Hence, it is illegal to have a multi-bit notifier in Verilog.

The following code illustrates the occurrence of REG_NR_MBNT.

```
module top (input data1, data2, clk1, clk2, output q);
  logic [1:0]notif;
  specify
    specparam tsetup = 7, tskew = 7, thold = 7, twidth = 7, tperiod = 10;
    $setup(data1, posedge data2, tsetup, notif);
    $skew(posedge clk1, posedge clk2, tskew, notif);
    $setuphold(posedge data2, data1, tsetup, thold, notif);
    $width(posedge data2, twidth, thold, notif);
    $period(posedge clk, tperiod, notif);
  endspecify
endmodule
```

In the system timing checks, notifier `notif` (last argument) is not of size 1.

[Back to Top](#)


JasperGold Superlint Checks Reference

Lint Checks

REG_NR_RDBA

Short Message: *Register '%s', assigned using blocking assignment, is being read before getting assigned.*

Severity	Warning
Description	If a register is assigned using a blocking assignment, it must always be assigned before being read. A read before a blocking assignment might infer an extraneous data storage element (latch in a combinational process and flip-flop in a sequential process).



Important

This check is available for Verilog designs only.

The following code illustrates the occurrence of REG_NR_RDBA.

```
module test(input sel,output reg [1:0] out);
  reg [1:0] count0to3;
  always @(sel)
    begin
      out <= count0to3;
      count0to3 = count0to3 + 2'd1;
    end
endmodule
```

In the above code, register `count0to3` is getting assigned using a blocking assignment, but is being read before getting assigned, which might lead to inference of an extra latch.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

REG_NR_UASR

Short Message: *Local register variable '%s' is unassigned, but is read at least once in %s %s.*

Severity	Warning
Description	The specified register has been read but no assignment is made to the register. Such usage leads to redundant code, undefined states, and errors. To avoid this, assign registers properly. If unassigned registers are not used, remove them. This rule honors the synthesis off/on pragmas. This implies that a register is considered unassigned when assignment is made inside synthesis off/on pragma and its declaration is outside the pragma.

The following code illustrates the occurrence of REG_NR_UASR.

```
module unassigned_reg (in1, out1);
input in1;
output out1;
reg out1;
reg g1;
always @(in1 or g1)
    out1 <= in1 & g1;
endmodule
```

In the above example, register `g1` is unassigned but is being read in an expression

[Back to Top](#)

RST_IS_NFST

Short Message: *Reset '%s' of flip-flop '%s' is not derived from first signal from reset order.*

Severity	Warning
Description	Reset signal of the flip-flop must be the first signal provided by the reset order from design information.

JasperGold Superlint Checks Reference

Lint Checks

The following code illustrates the occurrence of RST_IS_NFST.

Add the following lines to the input Tcl file:

```
config_rtltds -reset -set_order {{rst} {rst_f}}

module test(clk,rst,rst_f,port_a,in);
  input clk;
  input rst_f;
  input rst;
  output reg port_a;
  input in;

  always @(posedge clk or negedge rst_f)
  begin
    if (!rst_f)
      port_a = 2'b00;
    else
      port_a = in;
    end
  endmodule
```

In the above example, `port_a` reset is driven by signal `rst_f`, which is not the first signal declared by reset ordering.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

RST_XC_LDTH

Short Message: *The set/reset '%s' drives a combinational logic. Depth '%d' exceeded at '%s'.*

Severity	Warning
Description	The depth of combinational logic driven by the reset exceeds the threshold depth specified in the Superlint rules file. Longer logic paths result in longer path delays and, therefore, timing closure can become difficult.
Parameter associated	<p>The threshold value of depth can be controlled by the following parameter:</p> <pre>params RST_XC_LDTH {reset_logic_depth_threshold=0}</pre> <p>By default, the value of this parameter is 0.</p> <p>The depth of the combinational logic is calculated on the basis of the weight assigned to different logic gates using the parameters present in the Superlint rules file.</p>

The following code illustrates the occurrence of RST_XC_LDTH.

```
module mod_a (in_a, in_b,rst,clk, out_a, out_b);
input in_a, in_b,clk,rst;
output out_a, out_b;
reg out_a;
assign out_b = rst & in_b;
always @(posedge clk or negedge rst)
    if(!rst)
        out_a <= 1'b0;
    else
        out_a <= in_a;
endmodule
```

The Superlint rules file includes:

```
params RST_XC_LDTH {reset_logic_depth_threshold=1}
```

The above code reports a violation for the reset `rst` that drives the combinational logic and has logic depth more than the threshold value. To avoid this violation, remodel your design and ensure that reset `rst` is supplied only to the reset pin of a flip-flop.

JasperGold Superlint Checks Reference

Lint Checks

[Back to Top](#)

SEQ_NR_BLKA

Short Message: *Blocking assignment encountered in a sequential block.*

Severity	Warning
Description	Use of blocking assignments in sequential always blocks in Verilog, makes the execution of the statements dependent on the scheduling of events by the simulator. This leads to simulation race conditions. To avoid such situations, only non-blocking assignments should be used inside sequential always blocks. The non-blocking assignments are all scheduled after the continuous assignments and blocking assignments. From a users perspective, the execution of two non-blocking assignments happen in parallel. This leads to a definite behavior in simulation and avoids any simulation race conditions.

The following code illustrates the occurrence of SEQ_NR_BLKA.

```
module block_assgn (clk, q, d);
  input clk, d;
  output q;
  reg q;
  always @(posedge clk)
    q = d;
endmodule
```

In the above code, blocking assignment is used in the assignment `q = d`.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

SIG_NO_ASIG

Short Message: *Signal '%s' is unassigned, but is read at least once in %s '%s'.*

Severity	Warning
Description	The specified variable has been read but no assignment is made to it. To avoid redundant code, prevent undefined states, and to avoid confusion, any unassigned signal should be either properly assigned or should be removed from the design description. This rule honors the synthesis off/on pragmas. This implies that a variable is considered unassigned when assignment is made inside synthesis off/on pragma and its declaration is outside the pragma.

The following code illustrates the occurrence of SIG_NO_ASIG.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Aliases is
generic (s1 : integer := 9;
        s3 : integer := 7;
        s2 : integer := 1);
port (sort : out unsigned(s1 downto 0);
      ena : in unsigned(s2 downto 0) );
end Aliases;

architecture rtl_sigvar of Aliases is
signal MyInstructionWord:unsigned(s3 downto 0);
alias InstructionId:unsigned (s2 downto 0) is MyInstructionWord (7 downto 6);
--WARNING
alias AddressScheme :unsigned (s2 downto 0) is MyInstructionWord (5 downto 4);
--WARNING
alias LeftOperand :unsigned (s2 downto 0) is MyInstructionWord (3 downto 2); -
--WARNING
alias RightOperand :unsigned (s2 downto 0) is MyInstructionWord (s2 downto 0);
--WARNING
begin
p0:process
begin
    InstructionId <="01";
```

JasperGold Superlint Checks Reference

Lint Checks

```
AddressScheme <="10";
LeftOperand <="00";
RightOperand <="11";
end process p0;
beh1: sort<=ena & InstructionId & AddressScheme & LeftOperand & RightOperand;
end rtl_sigvar;
```

In the above code, signal `MyInstructionWord` is read but not assigned.

[Back to Top](#)

SIG_NO_READ

Short Message: *Signal '%s' is not read, but assigned at least once in %s '%s'.*

Severity	Warning
Description	The specified signal is unconnected (unread) but is assigned some object in the specified scope. To avoid redundant code, prevent undefined states, and to ensure any potential errors are caught, any unread signal should be properly read or removed from the design description. This rule honors the synthesis off/on pragmas, which implies that a signal is considered unread when it is read inside synthesis off/on pragma and its declaration is outside the pragma.

The following code illustrates the occurrence of `SIG_NO_READ`.

```
Library IEEE;
use IEEE.std_logic_1164.all;

entity ent_ABC is end;
architecture rtl of ent_ABC is
  signal s_out : std_logic ;
begin
  s_out<= '0';
end rtl;
```

In the above code signal `s_out` is assigned but not read.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

SIG_NO_USED

Short Message: *Signal '%s' defined in %s '%s' is unused (neither read nor assigned).*

Severity	Warning
Description	The specified local signal is defined, but no usage of the signal was found in the specified scope. To avoid redundant code and ensure any potential errors are caught, any unused signal should be removed from the design description. This rule honors the synthesis off/on pragmas. This implies that a signal is considered unused when its usage is inside synthesis off/on pragma and its declaration is outside the pragma.

The following code illustrates the occurrence of SIG_NO_USED.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity TEST1 is
end entity TEST1;

architecture ARCH_TEST1 of TEST1 is
signal DA, DB, DZ : bit_vector(3 downto 0);
signal GLOBAL : std_logic_vector(7 downto 0) := "01010101";
alias OPCODE : std_logic_vector(2 downto 0) is GLOBAL(7 downto 5);
alias MICRO_OP : std_logic_vector(1 downto 0) is OPCODE(2 downto 1);
alias TEENY : std_logic_vector(1 downto 1) is MICRO_OP(1 downto 1);
begin
process
variable result : bit;
begin
if (OPCODE = "010") then
gate3 : DZ(3) <= DA(3) nand DB(3);
elsif (TEENY = MICRO_OP) then
result := DA(0);
TEENY <= MICRO_OP;
end if;
end process;
```

JasperGold Superlint Checks Reference

Lint Checks

```
end architecture ARCH_TEST1;
```

In the above code, signal GLOBAL is defined but not used. To avoid this warning, remove the signal declaration or use it.

[Back to Top](#)

SIG_NR_IDRG

Short Message: *Index range of integer signal '%s' is '%d'.*

Severity	Warning
Description	Good coding practice suggests that range of VHDL signals declared as integer should be constrained either to positive or natural values.

The following code illustrates the occurrence of SIG_NR_IDRG.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity SignalsIntegerType is
end entity SignalsIntegerType;
signal etat_KO: integer range -4 to 0;
signal etat_OK: integer range 3 downto 0;
architecture rtl_enum of SignalsIntegerType is
begin
    etat_KO<=etat_OK;
end rtl_enum;
```

In the above code, signal etat_KO is of type integer and is of range -4 to 0.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

TSK_NO_USED

Short Message: *%s '%s' defined in %s '%s' is unused.*

Severity	Warning
Description	The specified task/procedure is defined, but no invocation of the task/procedure is found in the HDL description of the design. To avoid redundant code and confusion, remove any unused task/procedure from the design description.

The following code illustrates the occurrence of TSK_NO_USED.

```
module unused_task (in1, out1);
  input in1;
  output out1;
  reg out1;
  task task1;
    output o1;
    input in1;
    begin
      o1 = in1;
    end
  endtask

  always @(in1)
    out1 = in1;
endmodule
```

In the above code, task `task1` is not used.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

TSK_NR_ASGV

Short Message: *Task '%s' in module '%s' assigns a value to global variable '%s'.*

Severity	Warning
Description	As a good coding practice, a task should not modify the variables that are not declared inside the task. This will lead to unpredictable results in the design.

The following code illustrates the occurrence of TSK_NR_ASGV.

```
module task_sets_global_var(in1, in2, out1, out2);
  input in1;
  input in2;
  output out1;
  output out2;
  reg out1;
  reg g1;
  task task1;
    output o1;
    input in1;
    input in2;
    begin g1 = in1;
      o1 = in2;
    end
  endtask
  always @(in1 or in2) task1 (out1, in1, in2);
  assign out2 = g1;
endmodule
```

In the above example, `in1` is assigned to global variable `g1`.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

TSK_NR_ESDE

Short Message: *Edge specified for data event '%s' in system task '%s'.*

Severity	Warning
Description	If an edge is specified for the data event in system tasks such as \$setup, \$hold, \$skew, \$recovery, \$setuphold, and so on, signal transition is limited. To avoid this issue, do not specify an edge for the data event.

The following code illustrates the occurrence of TSK_NR_ESDE.

```
module mod_a( port_a, clk, rst, port_b );
input port_a, clk, rst;
output port_b;
reg port_b;
reg reg_a;
always @ ( posedge clk or negedge rst )
begin
    if ( !rst )
        port_b <= 1'b0;
    else
        port_b <= port_a;
end
specify
    $setup(posedge port_a, posedge clk, 5, reg_a);
endspecify
endmodule
```

In the above code, `posedge` is specified for data event `port_a` in system task `$setup`. As a result, a violation is reported. Remodel the design such that an edge is not specified for the data event.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

TSK_NR_UGLV

Short Message: *Task '%s' in module '%s' uses global variable '%s'.*

Severity	Warning
Description	The specified task references a variable that is not supplied as a task input or defined local to the task. This might lead to unpredictable change in the value of the variable. This global variable should be supplied as a task input.

The following code illustrates the occurrence of TSK_NR_UGLV.

```
module glob_var_in_task (in1, in2, out1);
  input in1;
  input in2;
  output out1;
  reg out1;
  task task1;
    output o1;
    input in;
    begin
      o1 = in & in2;
    end
  endtask
  always @(in1)
    task1 (out1, in1);
endmodule
```

In the above example, a global variable, `in2`, is used in the task `task1`.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

VAR_NO_ASIG

Short Message: *Variable '%s' is unassigned, but is read at least once in %s '%s'.*

Severity	Warning
Description	The specified variable has been read but no assignment is made to it. This rule honors the synthesis off/on pragmas. This implies that a variable is considered unassigned when assignment is made inside synthesis off/on pragma and its declaration is outside the pragma.

The following code illustrates the occurrence of VAR_NO_ASIG.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity IdentifierLength is
generic (size : positive := 3);
port (sort : out unsigned (size downto 0);
      ena1 : in unsigned (size downto 0);
      ena2 : in unsigned (size downto 0));
end IdentifierLength;
architecture rtl_length of IdentifierLength is
signal entierentierentKO : std_logic; -- too long identifier
signal entierentierenOK : std_logic;
signal eKO : std_logic; -- too short identifier
begin
p0:process (ena1, ena2)
  variable variablevariablKO : std_logic; -- too long identifier
  variable variablevariabOK : std_logic;
  variable vKO : std_logic; -- too short identifier
begin
  sort <= ena1 + ena2;
  entierentierentKO <= variablevariablKO;
  entierentierenOK <= variablevariabOK;
  eKO <= vKO;
end process p0;
end rtl_length;
```

In the above code, `variablevariablKO`, `variablevariabOK`, and `vKO` are read but not assigned a value. To avoid this warning, assign a value to the variable.

JasperGold Superlint Checks Reference

Lint Checks

[Back to Top](#)

VAR_NO_EVTR

Short Message: *Event variable '%s' is never triggered.*

Severity	Warning
Description	Presence of unused event variable is not desired. In addition, usage of event variable is not a good design practice. They make the debugging process more tedious.

The following code illustrates the occurrence of VAR_NO_EVTR.

```
module event_not_triggered (d, q);
    input d;
    output q;
    event ev;
    assign q = d;
endmodule
```

The event `ev` defined in the above code is never triggered in the design. You should either remove the event if it is of no use, or make use of it.

[Back to Top](#)

VAR_NO_INTL

Short Message: *Variable '%s' is not initialized before being incremented/decremented in the for loop.*

Severity	Warning
Description	An index variable which is step-incremented/step-decremented in a for loop should have a valid value at the time of initialization of the for loop. If such a variable is incremented or decremented in the loop without proper initialization, unpredictable behavior is observed.

The following code illustrates the occurrence of VAR_NO_INTL.

JasperGold Superlint Checks Reference

Lint Checks

```
module test_top;
reg [15:0] data = 1;
reg [2:1] lpidx;
initial
begin : abc
    for(; lpidx <= 15; lpidx++)
    begin
        data[lpidx] = data[lpidx - 1] + 1;
    end
end
endmodule
```

In the above code example, register `lpidx` is not initialized to any value before being used as a loop index variable or before getting incremented. Initialize `lpidx` to 0 or 1 or some other value (using a constant or variable) depending on the loop count to avoid this problem.

[Back to Top](#)

VAR_NO_READ

Short Message: *Variable '%s' is not read, but assigned at least once in %s '%s'.*

Severity	Warning
Description	The specified variable has been read but no assignment is made to it. To avoid redundant code, prevent undefined states, and to avoid confusion, any unread variable should be either properly used/read or should be removed from the design description. This rule honors the synthesis off/on pragmas. This implies that a variable is considered unused when its usage is inside synthesis off/on pragma and its declaration is outside the pragma.

The following code illustrates the occurrence of VAR_NO_READ.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity variable_ex is
    port (
        i_clk    : in std_logic;
```

JasperGold Superlint Checks Reference

Lint Checks

```
    o_done  : out std_logic
  );
end variable_ex;

architecture rtl_ex of variable_ex is

  signal r_Done      : std_logic := '0';
  signal r_Var_Copy1 : std_logic_vector (5 downto 0) := "000000";
  signal r_Var_Copy2 : std_logic_vector (5 downto 0) := "000000";
  signal r_Var_CountV : std_logic_vector (5 downto 0) := "000000";

begin

  EX_VAR : process (i_clk)
    variable v_CountX : std_logic_vector(5 downto 0) := "000000";
    variable v_Count  : std_logic_vector(5 downto 0) := "000000";
    variable v_CountV : std_logic_vector(5 downto 0) := "000000";

  begin
    if rising_edge(i_clk) then
      v_Count := v_CountX + 1;
      v_CountV := r_Var_CountV;
      r_Var_Copy1 <= v_CountX;

      if v_CountX = 5 then
        r_Done <= '1';
        v_CountX := 0;
      else
        r_Done <= '0';
      end if;

      r_Var_Copy2 <= v_CountX;

    end if;
  end process EX_VAR;

  o_done <= r_Done;

end rtl_ex;
```


JasperGold Superlint Checks Reference

Lint Checks

In the above code, variable `v_CountV` is assigned a value but not read. To avoid this warning, read the variable `v_CountV` or remove it from the design.

[Back to Top](#)

VAR_NO_USED

Short Message: *Variable '%s' defined in %s '%s' is unused (neither read nor assigned).*

Severity	Warning
Description	The specified local variable is defined but not used anywhere in the design description. To avoid redundant code, prevent undefined states, and to avoid confusion, any unused variable should be removed from the design description. This rule honors the synthesis off/on pragmas. This implies that a variable is considered unused when its usage is inside synthesis off/on pragma and its declaration is outside the pragma

The following code illustrates the occurrence of VAR_NO_USED.

```
library ieee;
use ieee.std_logic_1164.all;

entity decal is
  port ( clk: in std_logic;
        rst : in std_logic;
        ena : in std_logic;
        data:in std_logic ;
        datain :in std_logic_vector (3 downto 0);
        dataout :out std_logic_vector (3 downto 0);
        synprt:out std_logic
        );
end decal;

architecture rtl of Decal is
  signal reg2,data1,data2,rstnck : std_logic ; -- NOTE : only one declaration per
  line
  signal reg1 :std_logic ; signal bara : std_logic ; -- NOTE : only one
  declaration per line
```

JasperGold Superlint Checks Reference

Lint Checks

```
begin
My_proc : process(clk,rst)
  variable gclk,bidon1,biii : std_logic ; -- NOTE : only one declaration per
line
  begin
    if rst = '0' then
      reg2 <= '0';
    else if rising_edge(clk) then
      reg2 <= data2;
    end if;
  end if;
end process My_proc;
end rtl;
```

In the above code, variable `gclk` is defined but not used. To avoid this warning, remove the variable from the design description or use it.

[Back to Top](#)

VAR_NR_INDL

Short Message: *A variable/signal '%s' in an RTL description is initialized in its declaration.*

Severity	Warning
Description	The specified RTL variable/signal should not be initialized in its declaration. Initialization should be done via a reset signal.

The following code illustrates the occurrence of VAR_NR_INDL.

```
module top;
  reg [1:0] count = 2'b0;
endmodule
```

In the above example, the highlighted code shows a `reg` declaration with initialization.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

VAR_NR_PRC

Short Message: Procedure '%s' in design unit '%s' uses global variable '%s'.

Severity	Warning
Description	The specified procedure references a variable that was not supplied as an input or defined local to the procedure. It is recommended that this global variable is supplied as a procedure input.

The following code illustrates the occurrence of VAR_NR_PRC.

```
Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity TEST1 is
port ( in1 : in bit ;
      in2 : in bit ;
      out1 : out bit;
      out2 : out bit
      ) ;
end TEST1 ;

architecture ARCH_TEST1 of TEST1 is
signal g2: bit;
signal g2: bit;
signal g3: bit;
signal g4: bit;
procedure t1(signal in1, in2: in bit;
signal o1: out bit) is
variable f1: bit;
variable f2: bit;
variable f3: bit;
variable f4: bit;
variable ftn_val: bit;
begin
case (g1) is
when '0' => f1 := in1;
```

JasperGold Superlint Checks Reference

Lint Checks

```
    when '1' => f1 := in2;
end case;

case (in1) is
    -- when g2 => f2 := in2;
    when '0' => f2 := in2;
    when others => f2 := not in2;
end case;

if (g3 = '1') then
    f3 := in1;
else
    f3 := in2;
end if;

f4 := g4;
o1 <= f1 and f2 and f3 and f4;
end t1;

begin
    --g1 <= g2;
end ARCH_TEST1;
```

In the above code, signal `g1` is not declared as a input or defined local to the procedure `t1` but is used inside the procedure. Supply this global variable as a procedure input.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

WIR_NO_READ

Short Message: Wire '%s' defined in %s '%s' does not drive any object, but is assigned at least once.

Severity	Warning
Description	<p>The specified wire is unconnected (unread) but is assigned some object. To avoid redundant code and prevent undefined states, any unread wire should be properly read/used or, if not required, then the reg should be removed from the design description. This rule honors the synthesis off/on pragmas. This implies that a wire is considered unread when it is read inside synthesis off/on pragma and its declaration is outside the pragma.</p> <p>By default, this check is applied to all signals. However, you can control the behavior of this check to ignore signals with specific names by modifying the following parameter in the default rules file:</p> <pre>params WIR_NO_READ {pattern=""}</pre>

The following code illustrates the occurrence of WIR_NO_READ.

```
module mod_a();  
  wire wir_a;  
  reg reg_a;  
  assign wir_a = 1'b0;  
endmodule
```

In the above code example, `wir_a` is being assigned but not read anywhere. Read or connect this wire to prevent undefined conditions.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

WIR_NO_USED

Short Message: *Wire '%s' defined in %s '%s' is unused (neither read nor assigned).*

Severity	Warning
Description	<p>The specified wire is defined, but no usage of the wire was found in the HDL model. To avoid redundant code and prevent undefined states, any unused wire should be properly used or, if not required, the wire should be removed from the design description. This rule honors the synthesis off/on pragmas. This implies that a wire is considered unused when its usage is done inside synthesis off/on pragma and its declaration is outside the pragma.</p> <p>By default, this check is applied to all signals. However, you can control the behavior of this check to ignore signals with specific names by modifying the following parameter in the default rules file:</p> <pre>params WIR_NO_USED {pattern=""}</pre>

The following code illustrates the occurrence of WIR_NO_USED.

```
module unused_wire (in1, out1);
  input in1;
  output out1;
  reg out1;
  wire g1;
  always @(in1)
    out1 <= in1;
endmodule
```

In the above code, wire `g1` is neither assigned nor being read anywhere. If `g1` is not required, it should be removed from the HDL.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

WIR_NR_UASR

Short Message: *Wire '%s' defined in %s '%s' is unassigned, but drives at least an object.*

Severity	Warning
Description	<p>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.</p> <p>To avoid redundant code, prevent undefined states, and ensure any potential errors are caught, any unassigned wire should be properly assigned or, if not required, then the wire should be removed from the design description. This rule honors the synthesis off/on pragmas. This implies that a wire is considered unassigned when assignment is made inside synthesis off/on pragma and its declaration is outside the pragma.</p> <p>By default, this check is applied to all signals. However, you can control the behavior of this check to ignore signals with specific names by modifying the following parameter in the default rules file:</p> <pre>params WIR_NR_UASR {pattern=""}</pre>

The following code illustrates the occurrence of WIR_NR_UASR.

```
module mod_a()  
  wire wir_a;  
  reg_a;  
  always@(posedge clk)  
    reg_a = wir_a;  
endmodule
```

In the above code, the wire `wir_a` is used only to drive `reg_a` and is not being assigned anywhere. Assign this wire to prevent undefined conditions.

[Back to Top](#)

SIM_SYNT

The checks in this category report the coding styles in the RTL description which can lead to a mismatch between the pre-synthesis simulation and post-synthesis simulation results.

JasperGold Superlint Checks Reference

Lint Checks

These checks help reduce the time consumed in debugging the differences in simulation results obtained before and after synthesis.

The SIM_SYNTH category includes the following rules:

- ALW_NR_MSLV on page 168
- ALW_NR_UNUV on page 169
- CST_NO_BWID on page 170
- CST_NR_MSBX on page 171
- CST_NR_MSBZ on page 172
- DLY_NR_XZVL on page 172
- MOD_NR_SYXZ on page 173
- SIG_NR_NDCL on page 174

ALW_NR_MSLV

Short Message: *Identifier '%s' appearing in the sensitivity list is modified inside the block.*

Severity	Warning
Description	An assignment to a variable that appears in the always/process block sensitivity list has been encountered within the always/process body. This may result in an inefficient simulation model that continually evaluates this always/process block. It is also possible that this model will synthesize a latch/flip-flop for this variable, which may not be the intent of the design.

The following code illustrates the occurrence of ALW_NR_MSLV.

```
module sl_var_modified_in_blk (b, c, out1);
  input b;
  input [1:0] c;
  output out1;
  reg out1;
  reg a;
  always @(a or b or c)
```


JasperGold Superlint Checks Reference

Lint Checks

```
begin
    out1 = a | b;
    a = c;
end
endmodule
```

The highlighted code indicates that the variable `c` in the sensitivity list is assigned to variable `a` in the always block. This may result in simulation and synthesis mismatches.

[Back to Top](#)

ALW_NR_UNUV

Short Message: *Variable '%s' appearing in the sensitivity list is not used in the '%s' block.*

Severity	Warning
Description	A variable found in the sensitivity list of an always/process block does not appear in a right-hand side expression or conditional select expression within the block. This can result in an inefficient simulation model as the always/process block will get triggered at the change of the variable that is not required. This can also result in a synthesized flip-flop/latch model having a complicated clock/enable signal.

The following code illustrates the occurrence of ALW_NR_UNUV.

```
module top (in1, in2, out1, out2);
    input in1;
    input in2;
    output out1;
    output out2;
    reg out2;
    always @(in1 or in2)
        begin
            out2 = 1'b0;
        end
    assign out1 = in1 & in2;
endmodule
```

JasperGold Superlint Checks Reference

Lint Checks

In the above mentioned example, the `always` block has `in1` and `in2` in the sensitivity list, but these signals are not read within the `always` block. Remodel the design to avoid this issue.

[Back to Top](#)

CST_NO_BWID

Short Message: *Bit width not specified for constant '%s'.*

Severity	Warning
Description	<p>If the bit width is not specified for a constant, it is assumed to be 32 bits. This might lead to an increase in memory usage. For optimum memory usage, specify the bit width for all constants in the design. By default, a violation is not reported for constants with a bit width less than the value specified using the following parameters in the default rules file:</p> <pre>params CST_NO_BWID {bit_width_allowed_bin_system=5}</pre> <p>This parameter controls the behavior of this check when the binary number system is used to define constants. The default value of this parameter is 5, which indicates that the check is reported for binary constants of five bits or more.</p> <pre>params CST_NO_BWID {bit_width_allowed_other_system=5}</pre> <p>This parameter controls the behavior of this check when a number system other than binary is used to define constants. The default value of this parameter is 5, which indicates that the check is reported for constants of five digits or more (defined using a number system other than binary).</p>

The following code illustrates the occurrence of CST_NO_BWID.

```
module mod_a(port_a);  
    output port_a;  
    assign port_a = 'b0000000;  
endmodule
```

JasperGold Superlint Checks Reference

Lint Checks

In the above code, a violation is reported for constant 'b000000 because it is a 6-bit constant and its width is not specified. To avoid this violation, specify the bit width for 'b000000 as shown below:

```
parameter param_b = 6'b000000;
```

[Back to Top](#)

CST_NR_MSBX

Short Message: *Extension of 'x' bits in a constant.*

Severity	Warning
Description	The most significant bit of a constant was determined to be x. The width determined for this constant as a value to be assigned or as an operand to an expression was determined to be larger than the constant's width. The constant value will be left-padded with an appropriate number of x bits..

The following code illustrates the occurrence of CST_NR_MSBX.

```
module mod_a (port_a, port_b);  
  output [3:0] port_a;  
  output [7:0] port_b;  
  assign port_a = 4'bx1;  
  assign port_b = 4'bxx;  
endmodule
```

In the above code, the constant values assigned to port_a and port_b are being extended by x bits.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

CST_NR_MSBZ

Short Message: *Extension of 'z' bits in a constant.*

Severity	Warning
Description	The most significant bit of a constant was determined to be <i>z</i> . The width determined for this constant as a value to be assigned or as an operand to an expression was determined to be larger than the constant's width. The constant value will be left-padded with an appropriate number of <i>z</i> bits..

The following code illustrates the occurrence of CST_NR_MSBZ.

```
module mod_a (port_a, port_b);  
  output [3:0] port_a;  
  output [7:0] port_b;  
  assign port_a = 4'bz1;  
  assign port_b = 4'b?1;  
endmodule
```

In the above code, the constant values assigned to `port_a` and `port_b` are being extended by *z* and *?* bits, respectively.

[Back to Top](#)

DLY_NR_XZVL

Short Message: *Delay value contains an x/z.*

Severity	Warning
Description	In general, simulation time delays are not used in static verification and hence are ignored. Some synthesis tools support only integer delays that are multiples of the specified clock cycle for a design. Delay expressions consisting of <i>x/z</i> bits are usually unsupported by synthesis tools.

The following code illustrates the occurrence of DLY_NR_XZVL.

JasperGold Superlint Checks Reference

Lint Checks

```
module testcase (clk, sel1, sel2);
input clk;
output [3:0] sel1;
output [3:0] sel2;
reg [3:0] sel1;
reg [3:0] sel2;
parameter delay_x = 2'bx1;
parameter delay_z = 2'bz1;
always @(posedge clk)
begin
    #delay_x sel1<= 4'b0000;
    #delay_z sel2<= 4'b0010;
end
endmodule
```

In the above mentioned example, assignments to `sel1` and `sel2` use delay values that contain `x` and `z`, respectively. Remodel the design to avoid this warning.

[Back to Top](#)

MOD_NR_SYXZ

Short Message: *Synthesizing 'x'/'z' values in module '%s'.*

Severity	Warning
Description	<p>The <code>x</code> and <code>z</code> values are not propagated in literals and expressions. They are converted to 1 and 0, respectively, for cycle simulation. This can cause synthesis simulation mismatch.</p> <p>The check does not get reported on the default branch of the <code>case</code> statement and the last <code>else</code> branch of the <code>if-else</code> statement when all the possible combinations have been covered by the <code>if-else</code> block.</p>

The following example illustrates this problem.

```
module test(sel,port_a,port_b,port_c,port_d,port_e,out);
input sel,port_a,port_b,port_c,port_d;
output port_e, out;
reg port_e, out;
```

JasperGold Superlint Checks Reference

Lint Checks

```
always @(port_a or port_b or port_c or port_d)
begin
casez(port_d)
  2'b00: port_e = port_a;
  2'b01: port_e = port_b;
  2'b10: port_e = port_c;
  2'b11: port_e = 4'b00xx;
  default: port_e = 4'bx;
endcase
end
always @(sel)
begin
  if(sel == 1'b1)
    out <= 1'b1;
  else if(sel <= 1'b0)
    out <= 1'b0;
  else
    out <= 1'bx;
end
endmodule
```

In the above example, a violation is not reported for the default branch of the `case` statement and the last `else` statement of the `if-else` block.

[Back to Top](#)

SIG_NR_NDCL

Short Message: *Signal '%s' is declared as '%s'. Use of '%s' can lead to simulation/synthesis.*

Severity	Error
Description	The modeling information of the specified signal is within its net declaration. The use of advanced net declarations to model logic contention is not supported in synthesis. Such usage leads to mismatch in the design behavior observed in pre-synthesis simulation and the behavior after. Only <code>wire</code> and <code>tri</code> should be used for net declaration. Avoid <code>wand</code> , <code>triand</code> , <code>wor</code> , <code>trior</code> , <code>triereg</code> , <code>tri0</code> , and <code>tri1</code> .

JasperGold Superlint Checks Reference

Lint Checks

The following code illustrates the occurrence of SIG_NR_NDCL.

```
module wire_types (in1, in2, in3, out1);
    input in1;
    input in2;
    input in3;
    output out1;
    wire w1;
    wand w2;
    wor w3;
    tri w4;
    triand w5;
    trior w6;
    tri0 w7;
    tril w8;
    trireg w9;
    supply0 s0;
    supply1 s1;
    assign w1 = in1;
    assign w2 = in2;
    assign w2 = in3;
    assign out1 = w2;
endmodule
```

In the above mentioned example, w2, w3, w5, w6, w7, w8, and w9 are declared as wand, wor, triand, trior, tri0, tril, and trireg, respectively. Remodel the design to avoid this error.

[Back to Top](#)

SYNTHESIS

This category consists of checks for any unsynthesizable constructs in the design. These rules help you achieve good synthesis results and highlight differences between simulation and synthesis. For example, initializing the signal in the declaration in the VHDL code or using initial statement to initialize the signal in Verilog code can result in unpredictable synthesis behavior.

The SYNTHESIS category includes the following rules:

- [ALW_IC_SENL](#) on page 178
- [ALW_NO_COMB](#) on page 179

JasperGold Superlint Checks Reference

Lint Checks

- ALW_NO_ETRG on page 179
- ALW_NO_EVTS on page 180
- ALW_NO_FFLP on page 181
- ALW_NO_LATH on page 182
- ALW_NR_MCLK on page 182
- ALW_NR_MXCK on page 183
- ALW_NR_TCST on page 184
- ASG_NR_NBCB on page 185
- ASG_NR_SUPN on page 186
- CAS_NR_EVLX on page 187
- CLK_IS_NSYT on page 188
- CLK_NR_DDBD on page 189
- CLK_NR_EDGE on page 190
- CND_NR_CMXZ on page 190
- FLP_NR_ASMX on page 191
- FLP_NR_MBCK on page 194
- FNC_NR_CREC on page 194
- IDX_NR_ORNG on page 195
- INP_NR_ASGN on page 196
- LAT_NR_BLAS on page 197
- LAT_NR_MXCB on page 197
- LOP_NR_FCND on page 198
- LOP_NR_GLID on page 199
- LOP_NR_INFL on page 201
- LOP_NR_SRLG on page 202
- MOD_NR_ALAS on page 203
- MOD_NR_ASLD on page 204

JasperGold Superlint Checks Reference

Lint Checks

- MOD_NR_CNDO on page 205
- MOD_NR_DSBC on page 206
- MOD_NR_EVRP on page 207
- MOD_NR_FINB on page 208
- MOD_NR_FKJN on page 209
- MOD_NR_FORE on page 209
- MOD_NR_FREL on page 210
- MOD_NR_IFSM on page 211
- MOD_NR_INIB on page 212
- MOD_NR_LDLY on page 213
- MOD_NR_NSLP on page 214
- MOD_NR_USWC on page 215
- MOD_NS_ADAS on page 216
- MOD_NS_DCSP on page 217
- MOD_NS_GTIN on page 218
- REG_NR_MNBA on page 219
- RST_IS_CPLX on page 220
- RST_NR_ASRO on page 221
- SIG_NO_HIER on page 222
- SIG_NR_MDRV on page 223
- TSK_NR_CLKE on page 224
- VAR_NO_COMR on page 225
- VAR_NR_OUTR on page 226
- VAR_NR_REAL on page 227
- VAR_NR_TIME on page 227

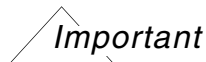
JasperGold Superlint Checks Reference

Lint Checks

ALW_IC_SENL

Short Message: *Sensitivity list incomplete in module %s, missing signal(s): %s.*

Severity	Warning
Description	The specified variable appearing on the right-hand side of an assignment, or in if/case conditional statements, is not found in the sensitivity list of the corresponding always/process block. This may cause mismatch between pre-synthesis and post-synthesis simulation results. This is because, during simulation, the always block gets triggered only when the signal specified in the sensitivity list changes. However, during synthesis, the sensitivity list is assumed to be complete.



This check is available for Verilog designs only.

The following code illustrates the occurrence of ALW_IC_SENL.

```
module test1 (a, b, c, out1);
  input a;
  input b;
  input [1:0] c;
  output out1;
  reg out1;
  always @(a or b)
    case (c)
      2'b00: out1 = a | b;
      2'b01: out1 = a & b;
      default: out1 = ~(a & b);
    endcase
endmodule
```

In the above code, `c` is used in the always block but it is not declared in the sensitivity list.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

ALW_NO_COMB

Short Message: *The node '%s' models a %s in an 'always_comb' block.*

Severity	Warning
Description	An <code>always_comb</code> block indicates the intent to model combinational logic. However, the logic inferred for the specified node does not match the design intent.

The following code illustrates the occurrence of ALW_NO_COMB.

```
module ALW_NO_COMB( input [2:0] in1, input en, output reg [3:0] out1 );
  always_comb
  begin
    if(en)
      out1 = in1;
    end
  endmodule
```

In the above example, the use of `always_comb` block indicates an intent to model combinational logic. However, a latch is inferred for `out1`.

[Back to Top](#)

ALW_NO_ETRG

Short Message: *Always block with no event trigger at the start in module '%s'.*

Severity	Warning
Description	An <code>always</code> block that does not have an event trigger at the start of the block cannot be synthesized. Remodeling of the HDL source will be required.

The following code illustrates the occurrence of ALW_NO_ETRG.

```
module test(d,q);
  input d;
  output q;
```

JasperGold Superlint Checks Reference

Lint Checks

```
reg q;
always
begin
    q <= d;
end
endmodule
```

In the above mentioned example, the `always` block does not have an event trigger at its start. This may cause a difference in simulation and synthesis results. Specify the signal `d` in the sensitivity list of the `always` block.

[Back to Top](#)

ALW_NO_EVTS

Short Message: *%s block with no event trigger at the start in %s %s.*

Severity	Error
Description	Always/process blocks with no event trigger at the start are not synthesizable and therefore are not supported. To avoid this warning, remodel the design.

The following code illustrates the occurrence of `ALW_NO_EVTS`.

```
module test(in_a);
input in_a;
reg out_a;
reg clk;
always
begin
    out_a <= in_a;
end
endmodule
```

In the above code, the `always` block has no event trigger. As a result, it is not synthesizable. Remodel the design. To avoid this warning, you can modify the `always` block as follows:

```
module test(in_a);
input in_a;
reg out_a;
reg clk;
always @(*)
begin
    out_a <= in_a;
end
endmodule
```

JasperGold Superlint Checks Reference

Lint Checks

```
always @(in_a)
begin
    out_a <= in_a;
end
endmodule
```

[Back to Top](#)

ALW_NO_FFLP

Short Message: *The node '%s' models a %s in an 'always_ff' block.*

Severity	Warning
Description	An <code>always_ff</code> indicates the intent to model a flip-flop. However, the logic inferred for the specified node does not match the design intent.

The following code illustrates the occurrence of `ALW_NO_FFLP`.

```
module noflop( input [3:0] in1,
               input en,
               output reg [3:0] out1
               );

always_ff@(en)
begin
    if(en)
        out1 = in1;
end
endmodule
```

In the above example, the use of `always_ff` block indicates an intent to model a flip-flop. However, a latch is inferred for `out1`.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

ALW_NO_LATH

Short Message: *The node '%s' models a %s in an 'always_latch' block.*

Severity	Warning
Description	An <code>always_latch</code> block indicates the intent to model latched logic. However, the logic inferred for the specified node does not match the design intent.

The following code illustrates the occurrence of ALW_NO_LATH.

```
module ALW_NO_LATH( input [3:0] in1, output reg [3:0] out1 );
  always_latch
  begin
    out1<= in1;
  end
endmodule
```

In the above example, the use of `always_latch` block indicates an intent to model latched logic. However, a wire is inferred for `out1`.

[Back to Top](#)

ALW_NR_MCLK

Short Message: *In module %s node %s has multiple clocks specified.*

Severity	Error
Description	The module/design-unit has an <code>always</code> block with multiple clock signals. For synthesizing a sequential element, an <code>always</code> block should have a unique clock signal.

The following code illustrates the occurrence of ALW_NR_MCLK.

```
module test(q, d, clk1, clk2);
  input d;
  input clk1, clk2;
  output q;
```

JasperGold Superlint Checks Reference

Lint Checks

```
reg q;
always @( posedge clk1 or posedge clk2 )
begin
    q = d;
end
endmodule
```

In the above code, two clock signals `clk1` and `clk2` are specified for the flip-flop `q`.

[Back to Top](#)

ALW_NR_MXCK

Short Message: *Always block has both level and edge sensitive nodes in its sensitivity list.*

Severity	Error
Description	The <code>always</code> 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. The semantics of the design may differ from simulation semantics.

The following code illustrates the occurrence of `ALW_NR_MXCK`.

```
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
end
endmodule
```

In the above code, the `always` block has level sensitive `clk` and edge sensitive `rst1`.

[Back to Top](#)

ALW_NR_TCST

Short Message: *The specified event expression cannot be synthesized.*

Severity	Error
Description	Event expressions that trigger on a constant cannot be synthesized. To avoid this error, modify the event expression such that it does not trigger on a constant. Associated parameter includes the following: <code>params ALW_NR_TCST {hide_verific_messages="no"}</code>

The following code illustrates the occurrence of ALW_NR_TCST.

```
module top;
  reg [1:0] pstate2;
  reg [7:0] clk2;
  reg [1:0] pstate3;
  always@(posedge clk2[8])
    pstate2 = 2'b00;
  always@(posedge 1)
    pstate3 = 2'b00;
endmodule
```

In the above code, ALW_NR_TCST triggers on both the highlighted lines.

■ `always@ (posedge clk2[8])`

In the above line, the event expression triggers on bit select [8] of `clk2`, which is outside the defined range of `clk2`. This bit select expression translates to do not care and hence the event expression cannot be synthesized.

■ `always@ (posedge 1)`

In the above line, the event expression triggers on constant 1. As a result, this event expression cannot be synthesized.

To avoid this error, modify the event expression such that it does not trigger on a constant.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

ASG_NR_NBCB

Short Message: *Non-blocking assignment encountered in a combinational block.*

Severity	Warning
Description	Non-blocking assignments are used in simulation to enforce an execution order. Non-blocking assignments assign a value to a variable after all blocking assignments are evaluated in a particular time frame. Combinational blocks are assumed to evaluate all logical/arithmetic operations concurrently. The presence of non-blocking statement in a combinational always block can lead to differences in the pre-synthesis and post-synthesis simulation results.

The following code illustrates the occurrence of ASG_NR_NBCB.

```
module nonblk_asgn_in_comb_blk (in1, in2, out1);
input in1;
input in2;
output out1;
reg out1;

always @(in1 or in2)
    out1 <= in1 & in2;
endmodule
```

In the given code, the non-blocking assignment `out1` is used in a combinational block. This might result in simulation and synthesis mismatches.

Note: If any bit of a vector is not fully assigned (for example, the latch is inferred), Superlint does not issue this warning. See the following example:

```
logic [3:0]out
always@(a or b) // no warning here as whole out is not fully assigned
if(a)
out[1:0] <= in1; // latch on out[3:0]
else
out <= in2;
```

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

ASG_NR_SUPN

Short Message: *Assignment to a supply0/supply1 net %s in module %s ignored.*

Severity	Warning
Description	As per the synthesis semantics, assignment to a supply0/supply1 nets are ignored as their values are fixed by definition. Remodel the design.

The following code illustrates the occurrence of ASG_NR_SUPN.

```
module assign_to_supply_net (in1, in2, out1);
input in1;
input in2;
output [3:0] out1;
reg [3:0] out1;
supply0 s0;
supply1 s1;
supply0 p0;
supply1 p1;
assign s0 = in1;
assign s1 = in2;
buf (p0, in1);
buf (p1, in2);
always @(s0 or s1 or p0 or p1)
    out1 = {s0,s1,p0,p1};
endmodule
```

In the above code, assignment to supply `s0` and `s1` are made. These assignments are ignored because the values for supply are fixed.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

CAS_NR_EVLX

Short Message: *In %s '%s', case item expressions evaluating to 'x/z/?' are ignored.*

Severity	Error
Description	For simulation, a case/casez item expression containing an x, z, or ? will be ignored. Comparisons to the case expression will never match. Remodel the design.

The following code illustrates the occurrence of CAS_NR_EVLX.

```
module dc_in_case_tag (a, b, c, sel, out1);
input a;
input b;
input c;
input [3:0] sel;
output out1;
reg out1;
always @(a or b or c or sel)
begin
    case (sel)
        4'b0010: out1 = a;
        4'b??01: out1 = b;
        4'b011?: out1 = c;
        4'b0x11: out1 = b;
        default: out1 = 1'b1;
    endcase
end
endmodule
```

In the above code, the highlighted line shows a `case` item which has a tag expression value containing an `x`. Comparison to this tag expressions are always false. Modify the tag expression to use a list of binary values.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

CLK_IS_NSYT

Short Message: *Unable to synthesize clock for signal '%s' in module '%s'.*

Severity	Error
Description	The tool issues this check when it is unable to determine the clock for a specific flip-flop and elaboration or synthesis cannot proceed.

The following code illustrates the occurrence of CLK_IS_NSYT.

```
Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity TEST1 is
port ( clock1 : in std_logic;
      clock2 : in std_logic;
      reset : in boolean;
      input : in std_logic;
      output : out std_logic
    ) ;
end TEST1 ;

architecture ARCH_TEST1 of TEST1 is
begin
process (clock1, reset)
begin
    if (clock1'event and clock1 = '1') then
        output <= input;
    elsif (reset) then
        output <= '0';
    end if;
end process;
end ARCH_TEST1 ;
```

In the above code, synthesis is unable to detect the clock signal for the given block.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

CLK_NR_DDBD

Short Message: *In module/design-unit %s, clock signal %s is driving data on both edges.*

Severity	Warning
Description	Clock signal with wrong polarity has been detected in the RTL. Data is being driven at both edges of <code>clock</code> . This may be because of an error in specifying an edge in the sensitivity list.

The following code illustrates the occurrence of CLK_NR_DDBD.

```
module test(clk, q,d, rst1, rst2);
input clk, d, rst1, rst2;
output q;
reg q;
always @(clk)
begin
    if (clk)
        q <= d;
    else
        q <= ~d ;
end
endmodule
```

In the above code, data is driven at both edges of signal `clk`. The boldfaced `if/else` block illustrates this.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

CLK_NR_EDGE

Short Message: *In module/design-unit %s clock signal %s for node %s does not drive any data. Wrong polarity specified.*

Severity	Warning
Description	Clock signal with wrong polarity is detected in the RTL. It is not driving any data on any edge of the clock. This may be due incorrect edge specification while modeling the design behavior.

The following code illustrates the occurrence of CLK_NR_EDGE.

```
module test(clock, load, d , dl, q);
input clock, load, d , dl;
output q;
reg q;
always @(posedge clock)
    if (~clock)
        q = d;
endmodule
```

In the above code, signal `clock` is not driving any data from `d` to `q`.

[Back to Top](#)

CND_NR_CMXZ

Short Message: *In module '%s', conditional expression containing 'x'/'z' is statically evaluated to false.*

Severity	Warning
Description	As per synthesis semantics, conditional or if statements with condition expression having x or z are statically evaluated to false. This interpretation differs from simulation semantics.

The following code illustrates the occurrence of CND_NR_CMXZ.

```
module condasgn_xz_cmp (a, b, sel, sel2, out1, out2, out3, out4);
```

JasperGold Superlint Checks Reference

Lint Checks

```
input a;
input b;
input sel;
input sel2;
output out1;
output out2;
output out3;
output out4;

assign out1 = (sel == 1'bx) ? a : b;
assign out2 = (sel2 == 1'bz) ? a : b;
assign out3 = (1'bx) ? a : b;
assign out4 = (1'bz) ? a : b;

endmodule
```

The boldfaced code indicates the presence of **x** and **z** in a conditional expression. Synthesis tools always evaluate it to false. It is recommended that you always use exact values in conditional expressions to avoid synthesis mismatches.

[Back to Top](#)

FLP_NR_ASMX

Short Message: *In the specified always/process block, descriptions of flip-flops with and without asynchronous set/reset are mixed. Flip-flops without asynchronous set/reset are: %s.*

Severity	Warning
Description	In the specified always/process block, flip-flops with asynchronous set/reset, as well as those without asynchronous set/reset, are mixed. This might be due to an omission in specifying an asynchronous set/reset. Some synthesis tools might not be able to handle such a description appropriately. If the description of asynchronous reset is omitted unintentionally, add the appropriate description. If the omission is intentional, describe the flip-flops with and without asynchronous set/reset in separate <code>always</code> blocks.

The following example illustrates this problem.

```
module mod_a(clk, rst, port_a, port_b, port_c, port_d);
```

JasperGold Superlint Checks Reference

Lint Checks

```
input clk;
input rst;
input port_a;
input port_b;
output port_c,port_d;
reg port_c;
reg port_d;
always @(posedge clk or negedge rst)
  if (rst==1'b0)
    port_c <= 1'b0;
  else
    begin
      port_c <= port_a;
      port_d <= port_b;
    end
endmodule
```

In the above code, an asynchronous reset is specified for flip-flop `port_c` but no asynchronous set/reset is specified for flip-flop `port_d`. To rectify this problem, specify an asynchronous set/reset for `port_d`, or describe the flip-flops with and without asynchronous set/reset in separate `always` blocks as shown below:

Specify an asynchronous set/reset for `port_d` .

```
module mod_a (clk, rst, port_a,port_b, port_c,port_d);
input clk;
input rst;
input port_a;
input port_b;
output port_c;
output port_d;
reg port_c;
reg port_d;
always @(posedge clk or negedge rst)
  if(rst== 1'b0 )
    begin
      port_c<= 1'b0;
      port_d<= 1'b0;
    end
  else
    begin
      port_c <=port_a;
```


JasperGold Superlint Checks Reference

Lint Checks

```
        port_d <=port_b;
    end
endmodule
```

Describe the flip-flops with and without asynchronous set/reset in separate always blocks.

```
module mod_a (clk, rst, port_a, port_b, port_c, port_d);
    input clk;
    input rst;
    input port_a;
    input port_b;
    output port_c;
    output port_d;
    reg port_c;
    reg port_d;
    always @ (posedge clk or negedge rst)
        if(rst == 1'b0)
            port_c <=1'b0;
        else
            begin
                port_c <= port_a;
            end
    always @ (posedge clk)
        port_d <= port_b;
endmodule
```

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

FLP_NR_MBCK

Short Message: *In %s %s, multi-bits '%s' used as clock for flip-flop '%s'.*

Severity	Warning
Description	The specified flip-flop uses a multi-bit clock. Clock edge will be determined as per language specifications, which states that the edge transitions will be detected on the least significant bit of the clock.

The following code illustrates the occurrence of FLP_NR_MBCK.

```
module test(q, d, clk);
  input d;
  input [1:8]clk;
  output q;
  reg q;
  always @( posedge clk )
    q = d;
endmodule
```

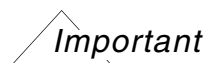
In the above code, signal `clk` is a multiple bit signal `[1:8]`.

[Back to Top](#)

FNC_NR_CREC

Short Message: *Function '%s' is called recursively in %s '%s'.*

Severity	Warning
Description	The specified function is called recursively. Some synthesis tools do not support recursion, so you might have problems synthesizing designs that include recursive functions. To avoid this issue, use a simple loop instead of calling a function recursively.



This check is available for Verilog designs only.

JasperGold Superlint Checks Reference

Lint Checks

The following code illustrates the occurrence of FNC_NR_CREC.

```
module test(a, b, x);
  input a;
  input b;
  output x;
  function func;
    input a;
    input b;
    begin
      if ( a != b )
        func = a & b;
      else //Recursive call of the function
        func = func(~a, b);
    end
  endfunction
  assign x = func(a, b);
endmodule
```

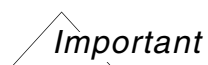
A violation is reported for the above code because function `func` is called recursively. To avoid this violation, use a simple loop instead of calling the function recursively.

[Back to Top](#)

IDX_NR_ORNG

Short Message: *Variable index/range selection of '%s' is potentially outside of the defined range.*

Severity	Warning
Description	A bit/part select reference in an expression is found to have a variable index specification that is potentially outside of the defined range of the variable. This can lead to unexpected simulation and/or synthesis results. Change this reference so that the index/subrange falls within the defined range.



This check is available for Verilog designs only.

JasperGold Superlint Checks Reference

Lint Checks

The following code illustrates the occurrence of `IDX_NR_ORNG`.

```
module top (count, d, q1);
  output q1;
  input [2:0] d;
  input [2:0] count;
  assign q1 = d[count];
endmodule
```

In the above code, `count` is used as a variable index into `d`, which has a range 2 down to 0. The size of `count` is 3 bits; hence, it can take values from 0 to 7. If `count` takes any values from 3 to 7 at run time, the index will be out of the defined range of `d`.

[Back to Top](#)

INP_NR_ASGN

Short Message: *Primary input port %s of module %s may be driven inside the module.*

Severity	Warning
Description	The specified primary input may be driven from inside the module. This port will be treated as an inout port. If this is not intentional, remodel the design.

The following code illustrates the occurrence of `INP_NR_ASGN`.

```
module tes(a, out1);
  input a;
  output out1;
  reg out1;
  assign a = 1'b1;
  always @(a) out1 = a;
endmodule
```

In the above example, the input signal `a` is being driven inside the module.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

LAT_NR_BLAS

Short Message: *In module/design-unit %s, latch is assigned by blocking assignments.*

Severity	Warning
Description	The design has one or more latches that are coded using a blocking assignment.

The following code illustrates the occurrence of LAT_NR_BLAS.

```
module Dlat(data, en, Q);
  input data, en;
  output Q;
  reg Q;
  always @(en or data)
    if (en)
      Q = data;
endmodule
```

In the above code, latch Q is coded using a blocking assignment.

[Back to Top](#)

LAT_NR_MXCB

Short Message: *The latches '%s' in the process/always block are mixed with combinational logic.*

Severity	Warning
Description	An always block has both latch and combinational logic. This can lead to simulation problems.

The following code illustrates the occurrence of LAT_NR_MXCB.

```
module test();
  wire wir_a,wir_b;
  reg sig_a;
```

JasperGold Superlint Checks Reference

Lint Checks

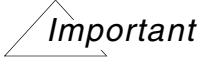
```
reg reg_a, reg_b;
always @(sig_a or wir_a or wir_b) begin
  if (sig_a == 2'b00) begin
    reg_a = wir_a & wir_b;
    reg_b = 1'b0;
  end
  else if (sig_a == 2'b01) begin
    reg_a = 1'b0;
    reg_b = wir_a | wir_b;
  end
  else if (sig_a == 2'b10) begin
    reg_a = 1'b0;
  end
  else begin
    reg_a = 1'b1;
    reg_b = 1'b0;
  end
end
endmodule
```

In the given code, `reg_b` is not assigned in all branches and is synthesized into a latch; `reg_a` is assigned in all branches and is modeled as combinational logic. So, both latch and combinational logic are in the same `always` block. To avoid this violation, assign `reg_b` in all branches.

[Back to Top](#)

LOP_NR_FCND

Short Message: *Loop condition is false*

Severity	Warning
Description	The design includes one or more loops that will never be triggered. This modeling style might not have been intentional. Verify the HDL for correctness.
<div> Important</div> <p>This check is available for Verilog designs only.</p>	

JasperGold Superlint Checks Reference

Lint Checks

The following code illustrates the occurrence of LOP_NR_FCND.

```
module test (out, in);
  output [3:0] out;
  input [3:0] in;
  reg [3:0] out;
  integer i;
  always begin
    for ( i=2; i<1; i=i+1 ) begin
      out = i;
    end
  end
endmodule
```

In the above code, the variable `i` is initialized to 2 and will never be less than 1. Thus, the loop will never be triggered.

[Back to Top](#)

LOP_NR_GLID

Short Message: *The loop variable '%s' is used in multiple always blocks.*

Severity	Warning
Description	The specified global variable is used to iterate loops in multiple <code>always</code> blocks. These <code>always</code> blocks can execute simultaneously during simulation, causing incorrect results to be generated. To avoid such issues, use different variables to iterate loops in different <code>always</code> blocks.

The following code illustrates the occurrence of LOP_NR_GLID.

```
module top(input [3:0] in1, in2, output reg [3:0] out1, out2);
  integer i;
  always @*
  begin
    for(i = 0; i <= 3; i = i+1)
      out1[i] = in1[i];
  end
end
```

JasperGold Superlint Checks Reference

Lint Checks

```
always @*
begin
    for(i = 0; i <= 3; i = i+1)
        out2[i] = in2[i];
    end
endmodule
```

In the above code, integer `i` is used as the loop variable to iterate for loops in two `always` blocks. If both the `always` blocks execute in parallel during simulation, incorrect results might be generated. To avoid this violation, use unique global variables or use local variables as shown below:

Use unique global variables to iterate loops in `always` blocks.

```
integer i, j;
always @*
begin
    for(i = 0; i <= 3; i = i+1)
        out1[i] = in1[i];
    end

always @*
begin
    for(j = 0; j <= 3; j = j+1)
        out2[j] = in2[j];
    end
```

Use local variables to iterate loops in `always` block.

```
always @*
begin : B1
    integer i;
    for(i = 0; i <= 3; i = i+1)
        out1[i] = in1[i];
    end

always @*
begin : B2
    integer i;
    for(i = 0; i <= 3; i = i+1)
        out2[i] = in2[i];
    end
```

[Back to Top](#)


JasperGold Superlint Checks Reference

Lint Checks

LOP_NR_INFL

Short Message: *%s '%s' contains a possibly infinite loop.*

Severity	Warning
Description	The tool has detected a potential non-terminating loop. Synthesis semantics require that a loop in an HDL design must terminate.



This check is available for Verilog designs only.

The following example illustrates this problem:

```
Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
entity TEST1 is
port ( clk : in bit ;
      c: in bit_vector(15 downto 0);
      q : out bit_vector(15 downto 0)
      ) ;
end TEST1 ;

architecture ARCH_TEST1 of TEST1 is
signal a: bit_vector(15 downto 0);
begin
process (clk, c)
variable i: integer;
begin
if ((clk = '1') and not clk'stable) then
  while (TRUE) loop
    if (c(i) = '1') then
      q(i) <= c(i);
    end if;
  end loop;
end if;
end process;
end ARCH_TEST1;
```

JasperGold Superlint Checks Reference

Lint Checks

The boldfaced code in the above example points to the non-terminating loop in the design.

[Back to Top](#)

LOP_NR_SRLG

Short Message: *Flip-flop '%s' has reset/set and logic section in the same '%s' loop.*

Severity	Warning
Description	If the set/reset and logic sections of a flip-flop are described in the same for/while/do-while loop, logic synthesis cannot be performed. To avoid this issue, describe set/reset and logic sections in separate for/while/do-while loops.

The following code illustrates the occurrence of LOP_NR_SRLG.

```
module mod_a (port_a, port_b, clk, reset);
input [3:0] port_a;
output [3:0] port_b;
input clk, reset;
reg [3:0] port_b;
reg [2:0] reg_a;
always @ (posedge clk or negedge reset)
begin
    for ( reg_a = 0; reg_a <= 3; reg_a = reg_a + 2'b01 )
        if ( !reset )
            port_b[reg_a] <= 1'b0; // reset section
        else
            port_b[reg_a] <= port_a[reg_a]; // logic section
    end
endmodule
```

In the above code, the reset and logic sections of the flip-flop are described in one for loop. As a result, logic synthesis cannot be performed. To avoid this issue, describe the reset and logic sections of the flip-flop in separate for/while loops.

The following modified code eliminates this issue.

```
module mod_a (port_a, port_b, clk, reset);
input [3:0] port_a;
output [3:0] port_b;
```

JasperGold Superlint Checks Reference

Lint Checks

```
input clk, reset;
reg [3:0] port_b;
reg [3:0] reg_a;
always @ (posedge clk or negedge reset)
begin
    if ( !reset)
    begin
        for ( reg_a = 0; reg_a <= 3; reg_a = reg_a + 2'b01)
            port_b[reg_a] <= 1'b0; // reset section
        end
    else
    begin
        for ( reg_a = 0; reg_a <= 3; reg_a = reg_a + 2'b01)
            port_b[reg_a] <= port_a[reg_a]; // logic section
        end
    end
endmodule
```

[Back to Top](#)

MOD_NR_ALAS

Short Message: *Aliased modules are not supported by default. Module %s has duplicate input/inout ports which has effect of aliasing (shorting) two nets.*

Severity	Error
Description	<p>In Verilog, you can declare modules with duplicate input/inout ports. For example,</p> <pre>module alias_module(in1, in1); input in1; endmodule</pre> <p>Instantiating <code>alias_module</code> with two nets has an effect of physically shorting these two nets. For example, instantiation</p> <pre>alias_module i1(w1, w2);</pre> <p>shorts <code>w1</code> and <code>w2</code>.</p>

The following code illustrates the occurrence of OTP_UC_INST.

```
module test1 (a, a);
```

JasperGold Superlint Checks Reference

Lint Checks

```
input a;
reg b;
reg [1:0] c;
//output out1;
reg out1;
always @(a or b or c)
    case (c)
        2'b00:
            out1 = a | b;
        2'b01:
            out1 = a & b;
        default:
            out1 = ~(a & b);
    endcase
endmodule
```

In the above code, module `test1` has aliased inputs.

[Back to Top](#)

MOD_NR_AS LD

Short Message: *In module %s, asynchronous load is inferred for node %s. A synchronous reset may also be generated, if there is an error in the polarity of the reset signal.*

Severity	Warning
Description	As per synthesis semantics, Superlint interprets the <code>always</code> block assigning to the node as implying asynchronous load functionality. This may result in behavior that does not match pre-synthesis simulation behavior.
Parameter associated	params MOD_NR_AS LD {hierarchical = "yes" "no"} The default value for this parameter is <code>no</code> . When the value is set to <code>no</code> , Superlint checks for asynchronous load within the module where the <code>always</code> block is defined only. When the value is set to <code>yes</code> , Superlint checks for asynchronous load in the complete design.

JasperGold Superlint Checks Reference

Lint Checks

The following code illustrates the occurrence of MOD_NR_AS LD.

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

In the above code, a `posedge` of `rst` is used in the `always` block, but in the `if` block, `!rst` is used, which is an asynchronous load on `q`.

[Back to Top](#)

MOD_NR_CNDO

Short Message: *In module %s, %s comparison treated as %s.*

Severity	Warning
Description	As per synthesis semantics, the operators <code>===</code> and <code>!==</code> are treated as <code>==</code> and <code>!=</code> , respectively. This interpretation differs from simulation semantics.

The following code illustrates the occurrence of MOD_NR_CNDO.

```
module MOD_NR_CNDO (a, b, sel, sel2, out1, out2);
input a;
input b;
input sel;
input sel2;
output out1;
output reg out2;
    assign out1 = (sel === 1'bx) ? a : b;
    //assign out2 = (sel2 !== 1'bz) ? a : b;
always@*
```

JasperGold Superlint Checks Reference

Lint Checks

```
if(sel2 === 1'bz)
    out2 = a;
else
    out2 = b;
endmodule
```

The boldfaced code indicates the presence of conditional operators. Synthesis and simulation tools cannot always handle these operators efficiently, resulting in unpredictable tool behavior. Avoid conditional operators to prevent synthesis and simulation mismatches.

[Back to Top](#)

MOD_NR_DSBC

Short Message: *Module %s contains unsupported disable construct.*

Severity	Error
Description	The design includes <code>disable</code> constructs. These constructs are not synthesizable and hence are not supported. To avoid this error, remodel the design.

The following code illustrates the occurrence of MOD_NR_DSBC.

```
module test;
integer n,i,I;
reg a,b,c;
wire clk;
always
begin : break
    for (i = 0; i < n; i = i+1)
        begin : continue
            @clk if (a == 0)
                disable continue;
            @clk if (a == b)
                disable break;
        end

    for (I = 2; 1; I = I + 1)
        begin: loop2
            //c[I] = a[I] | b[I];
```

JasperGold Superlint Checks Reference

Lint Checks

```
        if (I == 3)
            disable loop2;
        end
    end
endmodule
```

In the above code, `disable` construct is used and it is not supported.

[Back to Top](#)

MOD_NR_EVRP

Short Message: *%s %s contains unsupported repeat event specification.*

Severity	Error
Description	The design includes Verilog <code>repeat</code> event expressions that are non-synthesizable. To avoid this error, remodel the design.

The following code illustrates the occurrence of MOD_NR_EVRP.

```
module neg_repeat_event (clk, num, data, q);
    input clk;
    input num;
    input data;
    output q;
    reg q;
    always @(clk or num or data)
        begin
            q = repeat(num) @(clk) data;
        end
endmodule
```

In the above code, `repeat` event expression is used in the `always` block. Remodel the design to avoid this error.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

MOD_NR_FINB

Short Message: *Module '%s' has a final block, which is ignored by synthesis tools.*

Severity	Warning
Description	Final blocks are ignored by synthesis tools. This can cause a pre-synthesis and post-synthesis simulation mismatch.

The following code illustrates the occurrence of MOD_NR_FINB.

```
module MOD_NR_FINB ( output reg out1, input in1);
  initial
  begin out1 = 1'b0;
    #10s $finish;
  end

  always @(in1)
  begin
    out1 = #1fs ~in1;
  end

  final
  begin
    $display("\nout1 = %b", out1);
    out1 = 1'b0;
  end
endmodule
```

In the above example, the code written inside the `final` block is ignored as per synthesis semantics.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

MOD_NR_FKJN

Short Message: *%s %s contains non-synthesizable fork-join constructs.*

Severity	Error
Description	The design includes fork-join constructs. Since fork-join constructs cannot be synthesized, they are not supported. To avoid this error, remodel the design.

The following code illustrates the occurrence of MOD_NR_FKJN.

```
module neg_fork_join (clk, cout);
input clk;
output cout;
reg cout;
always @(posedge clk)
begin
    fork cout = 1'b0;
    cout = 1'b1;
    join
end
endmodule
```

In the above code, fork-join construct is used. The fork-join constructs cannot be synthesized and hence are not supported. Remodel the design to avoid this error.

[Back to Top](#)

MOD_NR_FORE

Short Message: *Module %s contains unsupported forever construct.*

Severity	Error
Description	The design includes Verilog <code>forever</code> loops, which are non-synthesizable. Remodel the design to avoid this error.

The following code illustrates the occurrence of MOD_NR_FORE.

JasperGold Superlint Checks Reference

Lint Checks

```
module forever_statement (d, q);
input d;
output q;
reg q;
reg clk;
top t1();
always@*
begin clk = 1'b0;
  forever clk = ~clk;
end

always @(posedge clk)
begin: b1
  q = d;
end
endmodule
```

In the above code, forever loop is used. Verilog forever loops are non-synthesizable and hence not supported. Remodel the design to avoid this error.

[Back to Top](#)

MOD_NR_FREL

Short Message: *%s %s contains non-synthesizable force/release constructs.*

Severity	Error
Description	Synthesis semantics does not permit the use of <code>force/release</code> constructs for asynchronous force modeling. To avoid this error, remodel your design.

The following code illustrates the occurrence of MOD_NR_FREL.

```
module force_release (clk, cin, cout);
input clk;
input cin;
output cout;
reg cout;
always @ (posedge clk)
begin
```

JasperGold Superlint Checks Reference

Lint Checks

```
    force cout = 1'b1;
    cout = cin;
    release cout;
end
endmodule
```

In the above code, `force` and `release` constructs are used. Synthesis semantics does not permit the use of these constructs for asynchronous force modeling. To avoid this error, remodel the design.

[Back to Top](#)

MOD_NR_IFSM

Short Message: *%s %s contains implicit finite-state machine.*

Severity	Warning
Description	The design includes implicit FSMs (multiple event guards in a process). Implicit FSMs cannot be synthesized. It is recommended that you remodel your design.

The following code illustrates the occurrence of MOD_NR_IFSM.

```
module mod_impl_fsm (clk, out1);
input clk;
output [3:0] out1;
reg [3:0] out1;
always @(posedge clk)
begin
    out1 = 4'b0001;
    @(posedge clk)
    out1 = 4'b0010;
    @(posedge clk)
    out1 = 4'b0011;
    @(posedge clk)
    out1 = 4'b0100;
end
endmodule
```

JasperGold Superlint Checks Reference

Lint Checks

In the above code, the sensitivity list of the `always` block contains an `event` expression. There are additional `event` control statements within the `always` block, resulting in an implicit FSM. To avoid this error, remodel the design.

[Back to Top](#)

MOD_NR_INIB

Short Message: *Module '%s' has an initial block, which is being ignored.*

Severity	Warning
Description	The design includes initial blocks. Initial blocks are ignored by synthesis tools. This can cause a pre-synthesis and post-synthesis simulation mismatch.

The following code illustrates the occurrence of MOD_NR_INIB.

```
module test (clk,data_in,data_out);  
input clk;  
input data_in;  
output data_out;  
reg data_out;  
initial  
begin  
    data_out = 1'b0;  
end  
always @(posedge clk)  
begin  
    data_out = data_in;  
end  
endmodule
```

The boldfaced lines show the use of `initial` block in the design. Initial blocks are ignored by synthesis tools. This results in simulation mismatches between the behavioral and gate-level models.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

MOD_NR_LDLY

Short Message: *Lumped delay in %s '%s' is ignored.*

Severity	Warning
Description	<p>Simulation time delays are ignored in synthesis and static verification. Therefore, a violation is reported if delays are found in the design. Though synthesis tools ignore delay values, specifying a delay in flip-flop data input can be meaningful in RTL simulation.</p> <p>By default, all delays in the design are reported. However, you can allow delay specification in flip-flop data input, using the following parameter in the default rules file:</p> <pre>params MOD_NR_LDLY {allow_delay_in_ff_data="no"}</pre> <p>The default value for this parameter is <code>no</code>. When the value of this parameter is set to <code>no</code>, a violation is reported for all delays in the design. When the value of this parameter is set to <code>yes</code>, a violation is reported for all delays except for flip-flop data input.</p> <p>You can also allow delay specification in asynchronous reset/set of the flip-flop using the following parameter in the default rules file:</p> <pre>params MOD_NR_LDLY {allow_delay_in_ff_reset="no"}</pre> <p>The default value for this parameter is <code>no</code>. When the value of this parameter is set to <code>no</code>, a violation is reported for all delays in the design. When the value of this parameter is set to <code>yes</code>, a violation is reported for all delays except for asynchronous reset/set of flip-flop.</p> <p>If both the parameters are set to <code>yes</code>, delays are allowed in data as well as asynchronous reset/set of the flip-flop.</p>

The following code illustrates the occurrence of MOD_NR_LDLY .

```
module mod_a (clk, rst, port_a, port_b);
  input clk;
  input rst;
  input port_a;
  output port_b;
  reg port_b;
  always @ ( posedge clk or negedge rst)
    if (!rst)
      port_b <= #1 1'b0;
```

JasperGold Superlint Checks Reference

Lint Checks

```
else
    port_b<= #1 port_a;
endmodule
```

In the above code, a flip-flop with an asynchronous reset is modeled. This model has a delay in the reset control, as well as in flip-flop data input. By default, a violation is reported for both delays. If the value of parameter `allow_delay_in_ff_data` is set to `yes`, the delay in the reset control is reported, but the delay in data input is not reported. If the value of parameter `allow_delay_in_ff_reset` is also set to `yes`, the delay in the reset control is also not reported.

[Back to Top](#)

MOD_NR_NSLP

Short Message: *%s %s contains non-static loop.*

Severity	Error
Description	The tool has 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.

The following code illustrates the occurrence of MOD_NR_NSLP.

```
module neg_MOD_NR_NSLP (a, b, c);
input [2:0] a, b;
output [2:0] c;
reg [2:0] c;
parameter p1 = 3;
parameter p2 = 1'b0;
always @(a or b)
begin: P integer I;
    reg [1:0] N;
    N = 3;
    for (I = 0; I < N; I = I + 1)
        c[I] = a[I] & b[I];
    N = a[1:0];
    N = p1;

    for (I = 0; I < N; I = I + 1)
```

JasperGold Superlint Checks Reference

Lint Checks

```
    c[I] = a[I] & b[I];
    N = b[1:0];
    N = {1'b1, 1'b1};

    for (I = 0; I < N; I = I + 1)
        c[I] = a[I] & b[I];
        N = b[2:1];
        N = {1'b1, p2};

    for (I = 0; I < N; I = I + 1)
        c[I] = a[I] & b[I];
        N = a[2:1];

    for (I = 0; I < N; I = I + 1)
        c[I] = a[I] & b[I];
end
endmodule
```

In the above code, the boldfaced `for` loop is non-static because `N` gets a value from `a`, which is a variable.

[Back to Top](#)

MOD_NR_USWC

Short Message: *%s %s contains non-synthesizable wait construct.*

Severity	Error
Description	The design includes Verilog <code>wait</code> event controls, which are currently not supported. You need to remodel the design.

The following code illustrates the occurrence of MOD_NR_USWC.

```
module neg_wait (enable, a, b, c, d);
input enable;
input b;
input d;
output a;
output c;
```

JasperGold Superlint Checks Reference

Lint Checks

```
reg a;
reg c;
always @(enable or b or d)
begin
    wait (!enable) #10 a = b;
    #10 c = d;
end
endmodule
```

In the above code, a `wait` event control is used for the assignment `a = b`. Verilog `wait` event controls are currently not supported. Remodel the design to avoid this error.

[Back to Top](#)

MOD_NS_ADAS

Short Message: *Module %s has non-synthesizable assign/deassign statements.*

Severity	Warning
Description	The design includes procedural continuous assignments. Procedural continuous assignments are not synthesizable and hence are not supported.

The following code illustrates the occurrence of MOD_NS_ADAS.

```
module counter(cnt,clk,reset);
input reset,clk;
output [3:0] cnt;
dff dff1(a,b,c,d);
mine mine1(e,f);
reg [3:0] cnt_tem;
always @ (reset)
begin
    if (reset)
        assign cnt_tem = 4'b0000;
    else
        deassign cnt_tem;
end

always @ (posedge clk)
```


JasperGold Superlint Checks Reference

Lint Checks

```
    cnt_tem[3:0] = cnt_tem[3:0] + 4'd1;
    assign cnt = cnt_tem;
endmodule
```

```
module mine (outp , inp);
input inp;
output outp;
endmodule
```

In the above code, unsupported `assign` and `deassign` statements are found in the `always` block.

[Back to Top](#)

MOD_NS_DCSP

Short Message: *Disconnect specification in design unit '%s' not supported.*

Severity	Warning
Description	Disconnect specifications are not supported. They are not supported even by synthesis tools. Remodel your design.

The following code illustrates the occurrence of MOD_NS_DCSP.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity EX_DISCONNECT is
end entity EX_DISCONNECT;

architecture A1 of EX_DISCONNECT is

    signal Foo : STD_LOGIC bus;
    disconnect Foo: STD_LOGIC after 10 NS; -- then goes to 'Z'
    signal D : STD_LOGIC;

begin

    Logic: block (D = '1') is -- block executes while D = '1'
```

JasperGold Superlint Checks Reference

Lint Checks

```
begin
    Foo <= guarded D; -- Foo remains '1' for up to 10 NS after D /= '1' then
    goes to 'Z'
end block Logic;
end architecture A1;
```

In the above code `disconnect` is used, which is not supported.

[Back to Top](#)

MOD_NS_GTIN

Short Message: *'%s' not supported.*

Severity	Error
Description	The specified gate instance is not supported. To avoid this error, remodel the design.

The following code illustrates the occurrence of MOD_NS_GTIN.

```
module neg_prim_gate (d, ctrl1, ctrl2, out1, inout1, inout2);
input d;
input ctrl1;
input ctrl2;
output [0:7] out1;
inout [0:5] inout1;
inout [0:5] inout2;

/* MOS switches */
cmos (out1[0], d, ctrl1, ctrl2);
nmos (out1[1], d, ctrl1);
pmos (out1[2], d, ctrl1);
rcmos (out1[3], d, ctrl1, ctrl2);
rnmos (out1[4], d, ctrl1);
rpms (out1[5], d, ctrl1);

/* bidirectional pass switches */
tran(inout1[0], inout1[0]);
rtran(inout1[1], inout1[1]);
tranif1(inout1[2], inout2[2], ctrl1);
```

JasperGold Superlint Checks Reference

Lint Checks

```
rtranif1(inout1[3], inout2[3], ctrl2);
tranif0(inout1[4], inout2[4], ctrl1);
rtranif0(inout1[5], inout2[5], ctrl2);

/* pullup/pulldown */
pullup(out1[6]);
pulldown(out1[7]);
endmodule
```

In the above code, the gate instances, `tran`, `tranif1`, `rtranif1`, and `tranif0` are not supported. Remodel the design to avoid this error.

[Back to Top](#)

REG_NR_MNBA

Short Message: *In module '%s', register '%s' has multiple non-blocking assignments %s.*

Severity	Warning
Description	<p>The specified register has multiple non-blocking assignments in the same block. This may result in behavior that does not match pre-synthesis simulation behavior.</p> <p>By default, this check will not consider the default non-blocking assignment to a register as reset and report <code>REG_NR_MNBA</code> for the next non-blocking assignment to the same register. To avoid this check, set the value of the following parameter in the default rules file to <code>yes</code>.</p> <pre>params REG_NR_MNBA {mulnba_reset_mode="no"}</pre>

The following code illustrates the occurrence of `REG_NR_MNBA`.

```
module REG_NR_MNBA(q, d1, d2);
input d1, d2;
output q;
reg q;
always @(d1 or d2)
begin
    q <= d1;
end
```

JasperGold Superlint Checks Reference

Lint Checks


```
#10;  
    q <= d2;  
end  
endmodule
```

In the above code, register `q` has multiple non-blocking assignments. In this scenario, the simulation and synthesis behavior may differ.

[Back to Top](#)

RST_IS_CPLX

Short Message: *In module/design-unit %s for flip-flop %s reset is an expression.*

Severity	Warning
Description	The module/design-unit has a complex expression reset driving a flip-flop. Synthesizability requires that the reset be a single variable name defined in the RTL.
<div> Important</div> <p>This check is available for Verilog designs only.</p>	

The following code illustrates the occurrence of RST_IS_CPLX.

```
library ieee;  
use ieee.std_logic_1164.all;  
  
entity rstexp is  
    port (clk : in std_logic;  
          rst : in std_logic;  
          rstr : in std_logic;  
          data : in std_logic;  
          q : out std_logic);  
end rstexp;  
  
architecture rtl of rstexp is  
begin  
    process (rst, rstr, clk)  
    begin
```

JasperGold Superlint Checks Reference

Lint Checks

```
if (rst and rstr) = '1' then
    q <= '0' ;
elsif
    rising_edge (clk) then
        q <= data;
    end if;
end process;
end rtl;
```

In the above code, a complex reset expression, which is a function of reset forces `rst` and `rstr`, is inferred for flip-flop `q`. Remodel your design.

[Back to Top](#)

RST_NR_ASRO

Short Message: *Logic operator other than '!' or '~' used to describe asynchronous set/reset of flip-flop '%s'.*

Severity	Warning
Description	If a logic operator other than ! or ~ is used to describe an asynchronous set/reset, logic synthesis cannot be performed. To avoid this issue, do not use a logic operator other than ! or ~ to describe an asynchronous set/reset.

The following code illustrates the occurrence of RST_NR_ASRO.

```
module mod_a(clk, rst, port_a, port_b)
    input clk, rst, port_a;
    output port_b;
    reg port_b;
    always @ ( posedge clk or negedge rst )
    begin
        if (^rst )
            port_b <= 1'b0;
        else
            port_b <= port_a;
        end
    endmodule
```

JasperGold Superlint Checks Reference

Lint Checks

In the above code, the reset of flip-flop `port_b` is described using operator `^`. As a result, logic synthesis cannot be performed. To avoid this issue, replace the bitwise operator `^` with a logic operator (`!` or `~`).

[Back to Top](#)

SIG_NO_HIER

Short Message: *Module '%s' has unsynthesizable hierarchical reference.*

Severity	Error
Description	Out-of-module references are not synthesizable. Remodel the design to remove any out-of-module references.

The following code illustrates the occurrence of SIG_NO_HIER.

```
module top(input i1,i2,clk,rst, output reg o1,output o2);
test t1(i1,i2,o2);
always@(posedge clk)
    if( !rst )
        o1 <= 1'b0;
    else
        o1 <= t1.i1;
endmodule

module test(input i1,i2,output o1);
    assign o1 = i1 & i2;
endmodule
```

In the above code, the boldfaced portion has an out-of-module reference, which is not supported. Remodel the design to remove any out-of-module references.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

SIG_NR_MDRV

Short Message: *Node '%s' in design-unit '%s' is being driven at edges of more than one signal.*

Severity	Warning
Description	As per the synthesis semantics, data driven at more than one edge of same signal or same edge of more than one signal is not supported.

The following code illustrates the occurrence of SIG_NR_MDRV.

```
Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity TEST1 is
port ( clk : in bit ;
      d : in bit ;
      rst1 : in bit ;
      rst2 : in bit ;
      q : out bit
      ) ;
end TEST1 ;

architecture ARCH_TEST1 of TEST1 is
begin
process (clk, rst1)
begin
  if (((clk = '1') and (not clk'stable)) or
      ((rst1 = '1') and (not rst1'stable))) then
    if (rst1 = '1') then
      q <= '0';
    else
      q <= d;
    end if;
  end if;
end process;
end ARCH_TEST1 ;
```

In the above code, node `q` is being driven at edges of more than one signal.

JasperGold Superlint Checks Reference

Lint Checks

[Back to Top](#)

TSK_NR_CLKE

Short Message: *'%s of signal '%s' used in task '%s'*

Severity	Warning
Description	A task block cannot be synthesized if it includes a signal's edge description.

The following code illustrates the occurrence of TSK_NR_CLKE.

```
module mod_a(clk, in0, out0);
  input clk, in0;
  output out0;
  reg out0;
  task task_a;
  begin
    @(posedge clk)
      begin
        out0 <= in0;
      end
  end
endtask
endmodule
```

In the above code, `posedge` of signal `clk` is described in `task_a`. Remodel the design to avoid this violation.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

VAR_NO_COMR

Short Message: *Variable index/range selection of '%s' is too small to access its defined range completely.*

Severity	Warning
Description	A bit/part select reference in an expression is found to have a variable index specification that is too small to access its defined range completely.

The following code illustrates the occurrence of VAR_NO_COMR.

```
module assign_var_select (in1, in2, out1);
  input in1;
  input in2;
  output [3:0] out1;
  reg [3:0] out1;
  always @(in1 or in2)
  begin
    out1[in1] = in2;
  end
endmodule
```

In the above code, `in1` is used as a variable index into `out1`, which has a range 3 down to 0. The size of `in1` is 1 bit, hence it can take values from 0 to 1 only. Therefore, index selection of `out1` is too small to access its complete range.


[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

VAR_NR_OUTR

Short Message: *Bit/part select '%s' is outside of the defined range %s.*

Severity	Warning
Description	A bit/part select reference in an expression has an index specification outside the defined range of the variable. This can lead to unexpected simulation and/or synthesis results. Modify this reference so that the index/subrange falls within the defined range.
<div> Important</div> <p>This check is available for Verilog designs only.</p>	

The following code illustrates the occurrence of VAR_NR_OUTR.

```
module test1 (a, b, out1);
input [3:0] a;
input [3:0] b;
output out1;
wire [3:0] a;
reg [2:0] out1;
wire [2:-1] q;
wire [2:-4] w;
always @(a or b)
    out1 = q[4] & b[3]; // xmelab warn at q[4], VAR_NR_OUTR
    assign a[3] = out1[3] ; // xmelab warn at out1[3], VAR_NR_OUTR
    //assign w[1:-7] = 2'b00; //xmelab errs at last line if this uncommented
    //assign q[4] = 1; // xmelab err at q[4], same wire q at lhs
    assign q = w[1+:3]; // xmelab err at a[4]
endmodule
```

In the above code, `q[4]`, `out1[3]`, and `w[3:1]` are used, which are outside the defined range.

[Back to Top](#)

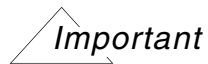
JasperGold Superlint Checks Reference

Lint Checks

VAR_NR_REAL

Short Message: *Real variable %s is used in %s %s. Real variables are not synthesizable.*

Severity	Error
Description	Real variables are not supported in synthesis models. They are converted to 32-bit integers when encountered in static verification tool flows.



This check is available for Verilog designs only.

The following code illustrates the occurrence of VAR_NR_REAL.

```
module real_in_cmp (b, out1);
input [1:0] b;
output out1;
reg out1;
    real r1;
always @(b)
    r1 = b;
endmodule
```

The above code displays the use of a `real` variable. These variables are not supported.

[Back to Top](#)

VAR_NR_TIME

Short Message: *Time variable %s is used in %s %s. Time variables are not synthesizable.*

Severity	Error
Description	Time variables are not supported in synthesis models. They are converted to 32-bit integers when encountered in static verification tool flows.

JasperGold Superlint Checks Reference

Lint Checks

The following code illustrates the occurrence of VAR_NR_TIME.

```
module convert_time_to_int (out1);
    output out1;
    wire [31:0] out1;
    time t1;
    initial
    begin
        t1 = 90;
    end
    assign out1 = t1;
endmodule
```

The code in the above example shows the use of a `time` variable. Remodel the design and avoid using `time` variables.

[Back to Top](#)

STRUCTURAL

This category performs checks for functional correctness, testability, and reusability over the entire design hierarchy and includes the following rules:

- [CLK_IS_MCDM](#) on page 229
- [CLK_NO_HGHI](#) on page 230
- [CLK_NO_INPT](#) on page 231
- [CLK_NS_EDMX](#) on page 232
- [CMB_NR_TLIO](#) on page 233
- [FLP_NO_ASRT](#) on page 234
- [FLP_NR_ASRT](#) on page 235
- [FLP_NR_ENCT](#) on page 236
- [INS_NR_INPR](#) on page 237
- [LAT_IS_FLSE](#) on page 238
- [MOD_IS_SYAS](#) on page 239
- [MOD_NO_IPRG](#) on page 240

JasperGold Superlint Checks Reference

Lint Checks

- MOD_NS_DCLK on page 241
- MOD_NS_GLGC on page 242
- RST_IS_DCMB on page 243
- RST_IS_DFLP on page 244
- RST_IS_DLAT on page 245
- RST_NO_HGHI on page 246
- RST_NR_MULT on page 247
- RST_NR_PENA on page 248
- SIG_IS_INTB on page 249
- SIG_IS_MDRV on page 250

CLK_IS_MCDM

Short Message: *Clock domain crossing is occurring between signals '%h' and '%h'. Perform CDC verification for more details.*

Severity	Info
Description	Clock domain crossing is occurring between signals '%h' and '%h'. Perform CDC verification for more details.

The following code illustrates the occurrence of CLK_IS_MCDM.

```
module clk_2_cross ( clock1, clock2, rst_n, data_in, data_out);
input clock1;
input clock2;
input rst_n;
output data_out;
input data_in;
reg data_out_meta;
reg [1:0] data_out_reg;
assign data_out = data_out_reg[1];
always @ (posedge clock1)
begin
    data_out_meta <= data_in;
```

JasperGold Superlint Checks Reference

Lint Checks

```
end
always @ (posedge clock2 or negedge rst_n)
begin
    if (! rst_n)
        data_out_reg <= 'b0;
    else
        data_out_reg <= {data_out_reg[0], data_out_meta};
    end
end
endmodule
```

[Back to Top](#)

CLK_NO_HGHI

Short Message: *The clock generation logic for clock '%s' is not at the same or a higher hierarchical level as the module/design-unit to which the clock applies.*

Severity	Warning
Description	The clock generation logic should be at the same level as the module to which the clock applies or at a higher level in the hierarchy. This eases clock network synthesis and improves the portability of the code to a different clocking scheme.

The following code illustrates the occurrence of CLK_NO_HGHI.

```
module mod_a(clk,rst,in2,in1,out1);
input clk, rst, in2;
input in1;
output reg out1;
wire clk_2;
always @(posedge clk_2 or negedge rst)
begin: block
    if(!rst)
        out1 = 0;
    else
        out1 = in1;
    end
end

mod_b inst_b(clk, in2, clk_2);
```

JasperGold Superlint Checks Reference

Lint Checks

```
endmodule

module mod_b(clk_1, in1, clk_2);
input clk_1, in1;
output clk_2;
    mod_c inst_c (clk_1, in1, clk_2);
endmodule
```

In the above code, the module `mod_a` contains a flip-flop `out1`, which is clocked by signal `clk_2`. The signal `clk_2` is the output `clk_2` of instance `inst_b` of module `mod_b`. In module `mod_b`, signal `clk_2` is the output of instance `inst_c` of clock generation module `mod_c`. The module `mod_c` lies at a lower hierarchical level than the flip-flop `out1` that uses the clock. To avoid this warning, instantiate the clock generation module `mod_c` in the module `mod_a`, or move the flip-flop `out1` to module `mod_b`.

[Back to Top](#)

CLK_NO_INPT

Short Message: *In design-unit/module '%m', clock signal '%l' for flip-flop '%l' is not an input.*

Severity	Warning
Description	The module/design-unit has a clock signal that is not an input, or the clock signal has been converted into an inout because of global interconnections. Synthesizability requires that the clock signal is an input to the module/design-unit.

The following code illustrates the occurrence of `CLK_NO_INPT`.

```
module test(q, d);
input d;
output q;
reg q, clk;
always @(posedge clk)
    q = d;
endmodule
```

In the above code, signal `clk`, which is the clock signal for flip-flop `q`, is not declared as input for module `test`.

JasperGold Superlint Checks Reference

Lint Checks

[Back to Top](#)

CLK_NS_EDMX

Short Message: *Both edges of clock signal '%s' used, positive edge at '%s' and negative edge at '%s'.*

Severity	Warning
Description	If both the edges of a clock are used in a module, there might be issues during scan insertion and clock tree synthesis. To avoid such issues, describe the different edges of the clock in separate modules.

The following code illustrates the occurrence of CLK_NS_EDMX.

```
module mod_a(clk, in0, out0, out1);
  input clk, in0;
  output out0, out1;
  reg out0, out1;
  always @(posedge clk)
    begin
      out0 <= in0;
    end

  always @(negedge clk)
    begin
      out1 <= !in0;
    end
endmodule
```

In the above code, both positive and negative edges of clock signal `clk` are used in the same module. As a result, a violation is reported. To avoid this issue, describe the different edges of the clock in separate modules.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

CMB_NR_TLIO

Short Message: *Combinational path detected between top-level input '%s' and top-level output '%s'.*

Severity	Warning
Description	As per coding guidelines, there should be no combinational path between top-level inputs and top-level outputs. There should be some sequential element, such as flip-flop, present in such path.

The following code illustrates the occurrence of CMB_NR_TLIO.

```
module top(input in1, in2, clk, rst, output out1, out2);
    reg out1;
    reg out2;
    DFF ffl(in1, clk, rst, out1);
    assign out2 = out1 & in2;
endmodule

module DFF(input in, clk, rst, output out);
    reg out;
    always@(posedge clk or negedge rst)
        if(!rst)
            out = 0;
        else
            out = in;
endmodule
```

In the above example, there is no sequential element between the primary output `out2` and primary input `in2`. To avoid this violation, add some flip-flop in the path between these two endpoints.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

FLP_NO_ASRT

Short Message: *Flip-flop '%s' does not have any asynchronous set or reset.*

Severity	Warning
Description	Flip-flops must have asynchronous set or reset. If a flip-flop does not have any asynchronous set or reset, then it is not controllable from primary inputs.

The following code illustrates the occurrence of FLP_NO_ASRT.

```
module mod_a(input rst, input var_a, output reg port_a);
  reg clk;
  always@(posedge clk)
  begin
    if(!rst)
      port_a <= 1'b0;
    else
      begin
        port_a <= var_a;
      end
    end
  end
endmodule
```

In the above HDL code, the flip-flop `port_a` does not have any asynchronous reset `rst`.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

FLP_NR_ASRT

Short Message: *Flip-flop '%s' has both asynchronous set and reset signals.*

Severity	Warning
Description	If a flip-flop uses both asynchronous set and reset, there might be differences in RTL- and gate-level simulation results. To avoid this issue, it is recommended that you use either asynchronous set or reset.

The following code illustrates the occurrence of FLP_NR_ASRT.

```
module mod_a(clk,port_a,reset,set,port_b);
input clk;
input reset,set;
input port_a;
output port_b;
reg port_b;
always @ ( posedge clk or negedge reset or posedge set )
    if(!reset)
        port_b <= 1'b0;
    else if(set)
        port_b <= 1'b1;
    else
        port_b <= port_a;
endmodule
```

In the above code, *both* asynchronous set and reset are used. Remodel the design so that *either* asynchronous set or reset is used.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

FLP_NR_ENCT

Short Message: *Signal '%h' driving enable pin of flip-flop '%h' is permanently enabled.*

Severity	Warning
Description	Enable pin of flip-flop is constant. This might cause a functional issue in the design as enable value never changes.

The following code illustrates the occurrence of FLP_NR_ENCT.

```
module test(clk,rst,port_f,port_e,var_a);
input clk,rst,port_f,port_e;
output var_a;
wire en;
reg var_a;
assign en = 1'b0;
always@(posedge clk or negedge rst)
begin
if (!rst)
var_a <= 1'b0;
else if(en)
var_a <= port_f;
else
var_a <= port_e;
end
endmodule
```

In the given example, the flip-flop `var_a` has its enable `en` as permanently constant. This can cause a glitch as `var_a` will never be assigned the value `port_f`. To avoid this violation, remodel the design.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

INS_NR_INPR

Short Message: *Input port '%h' of instance '%i' is not a register.*

Severity	Warning
Description	The specified input of an internal module is neither a register nor directly driving a register. For each block of the hierarchical design, register all input signals from the block.

The following code illustrates the occurrence of INS_NR_INPR.

```
module top(port_a,port_b,clk,rst,sel,port_c);
    output port_b;
    output port_c;
    input port_a;
    input[1:0] sel;
    input clk,rst;
    reg port_b;
    reg port_c;
    bot b1(port_a,port_b,clk,rst,sel,port_c);
endmodule
```

```
module bot(port_a,port_b,clk,rst,sel,port_c);
    output port_b;
    output port_c;
    input port_a;
    input[1:0] sel;
    input clk,rst;
    reg port_b;
    reg port_c;
    always@(clk && port_a)
    begin
        if (clk == 1)
            port_b = port_a;
    case(sel)
        2'b00: port_c = port_a;
        2'b11: port_c = 1'b1;
        default: port_c = 0;
    endcase
endmodule
```

JasperGold Superlint Checks Reference

Lint Checks

```
end endmodule
```

[Back to Top](#)

LAT_IS_FLSE

Short Message: *Latch '%s' is feeding latch '%s' having same enable %s.*

Severity	Warning
Description	A race condition may exist when two latches on the same enable are back-to-back in the same data path. This might cause unpredictable behavior.

The following code illustrates the occurrence of LAT_IS_FLSE.

```
module test (din, enable, reset1,reset2,sel,dout);
input din,enable,reset1,reset2;
input sel;
output dout;
reg dout;
reg data_inter;
always @(enable or reset1)
begin if (reset1 == 1'b1)
    data_inter = 1'b0;
else if (enable)
    data_inter = din;
end

always @(enable or reset2 or sel)
begin if (reset2 == 1'b1)
    dout = 1'b0;
else if (enable)
    dout = data_inter && sel;
end
endmodule
```

In the above code, latch `data_inter` feeds latch `dout` and both have the same enable.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

MOD_IS_SYAS

Short Message: *The module/design-unit '%s' contains synchronous as well as asynchronous logic. Asynchronous logic is present at '%s' and Synchronous logic at '%s'.*

Severity	Warning
Description	<p>Asynchronous and synchronous logic should be partitioned in separate modules. Such partitioning eases clock network synthesis and improves the portability of the code to a different clocking scheme. The behavior of this check can be controlled through the following parameter in the rules file:</p> <pre>params MOD_IS_SYAS { asynchronous_reset_is_synchronous = yes no }</pre> <p>By default, this parameter is set to <code>yes</code> and flip-flops with asynchronous set or reset are considered as synchronous logic while recommending partitioning synchronous and asynchronous logic in separate modules. When it is set to <code>no</code>, flip-flops with asynchronous set or reset are considered as asynchronous logic.</p>

The following code illustrates the occurrence of MOD_IS_SYAS.

```
module mod_a (in1,in2,clk,out1,out2);
input in1,in2;
input clk;
output reg out1, out2;
always @(posedge clk)
begin
    if(in1 == 1'b0)
        out1 = 1'b0;
    else
        out1 = in2;
end

always @( posedge in1 or negedge in2)
begin
    if ( in2 == 1'b0 )
        out2 = 1'b0;
```

JasperGold Superlint Checks Reference

Lint Checks

```
    else
        out2 = 1'b1;
    end
endmodule
```

In the above code, flip-flop `out1` has only synchronous control, while flip-flop `out2` has an asynchronous reset. To avoid this warning, code the two flip-flops in separate modules. Synchronous logic with asynchronous set or reset control is considered asynchronous.

[Back to Top](#)

MOD_NO_IPRG

Short Message: *Input port '%h' of top-level module is not a register.*

Severity	Warning
Description	The specified input of the top-level module is neither a register nor directly driving a register. For each block of the hierarchical design, register all input signals from the block. This check is issued only for top-level inputs.
Parameter associated	<p>The following parameter controls the behavior of MOD_NO_IPRG. The value given in the parameter is used to define the maximum number of levels of combinational logic that should be present before a flip-flop is encountered. The default value is 0.</p> <pre>params MOD_NO_IPRG {levels_of_combinational_logic="0"}</pre>

The following code illustrates the occurrence of MOD_NO_IPRG.

```
module mod_a(port_a,port_b,clk,sel,port_c);
    output port_b;
    output port_c;
    input port_a;
    input[1:0] sel;
    input clk;
    reg port_b;
    reg port_c;
    always@(clk && port_a)
begin
```


JasperGold Superlint Checks Reference

Lint Checks

```
    if (clk == 1)
        port_b = port_a;
    case(sel)
        2'b00: port_c = port_a;
        2'b11: port_c = 1'b1;
        default: port_c = 0;
    endcase
end
endmodule
```

In the given code, change the input of the top level module to `register` and register the input signals to increase controllability.

[Back to Top](#)

MOD_NS_DCLK

Short Message: *In instance '%i', clocks belong to following different clock domains: %g.*

Severity	Warning
Description	Separate clock domains should be partitioned into separate modules. This eases clock network synthesis and test strategy generation, limits exceptions to coding standards to a small module, and improves the portability of the code to a different clocking scheme.

The following code illustrates the occurrence of MOD_NS_DCLK.

```
module mod_a (clk_1, clk_2, rst, in1, in2, out1, out2);
    input clk_1, clk_2, rst;
    input [1:0] in1, in2;
    output reg [1:0] out1, out2;
    always @(posedge clk_1)
        begin
            if (!rst)
                out1 <= 2'b00;
            else
                out1 <= in1;
        end
end
```

JasperGold Superlint Checks Reference

Lint Checks

```
always @(posedge clk_2)
begin
    if (!rst)
        out2 <= 2'b00;
    else
        out2 <= in2;
end
endmodule
```

In module `mod_a`, clocks `clk_1` and `clk_2` of flip-flops `out1` and `out2` define different clock domains. To avoid this warning, place the flip-flops `out1` and `out2` in separate modules.

[Back to Top](#)

MOD_NS_GLGC

Short Message: *Glue logic inferred in top-level module/design-unit '%s'.*

Severity	Warning
Description	<p>Use of glue logic in the top-level module makes the floor plan problematic. As a result, glue logic should be avoided in the top-level module. Avoiding glue logic helps you enhance synthesis results, improves run time, and enables simpler synthesis strategies to meet timing.</p> <p>By default, this check is reported for all simple assignment statements in the design. To ignore this check for simple assignment statements, set the value of the following parameter in the default rules file to <code>yes</code>.</p> <pre>params ATLGLC {ignore_simple_asgnt_stmt="no"}</pre>

The following code illustrates the occurrence of MOD_NS_GLGC.

```
module mod_a(port_a,port_b,port_c);input port_a;
input port_b;
output port_c;
wire wire_a;
    assign wire_a = port_a & port_b;
    mod_b inst_b(wire_a,port_c);
endmodule
```

JasperGold Superlint Checks Reference

Lint Checks

```
module mod_b(port_a,port_b);  
input port_a;  
output port_b;  
endmodule
```

In the above code, the top-level module `mod_a` has glue logic (`assign wire_a = port_a & port_b`) due to which a violation is reported. To eliminate this issue, remove the glue logic from the top-level module.

[Back to Top](#)

RST_IS_DCMB

Short Message: *Signal '%h' driving reset of flip-flop '%h' is generated from combinational logic.*

Severity	Error
Description	The presence of combinational logic in the path of set/reset might lead to glitches, resulting in malfunctioning of the design. Such usage should be avoided.
Parameter associated	<pre>params RST_IS_DCMB {treat_multi_driven_as_combinational_logic=yes no}</pre> <p>The default value of this parameter is yes.</p> <p>When the value is set to <code>yes</code>, multi-driven signals are treated as combinational logic and therefore <code>RST_IS_DCMB</code> is reported.</p> <p>When the value is set to <code>no</code>, multi-driven signals are not treated as combinational logic. Instead, Superlint traces the drivers of the multi-driven signals and checks if a combinational logic drives the reset. If combinational logic drives the reset, then <code>RST_IS_DCMB</code> is reported.</p>

The following code illustrates the occurrence of `RST_IS_DCMB`.

```
module mod_a(q, data, clk, rst_a, rst_b);  
output q;  
input data, clk, rst_a, rst_b;  
reg q;  
wire reset;  
assign reset = rst_a | rst_b;  
always @(posedge clk or negedge reset)
```

JasperGold Superlint Checks Reference

Lint Checks

```
begin
  if (!reset)
    q <= 1'b0;
  else
    q <= data;
  end
endmodule
```

In the above code, flip-flop `q` has the combinational logic `rst_a | rst_b`, which is driving the reset. The signals `rst_a` and `rst_b` can be internal signals or primary inputs. This combinational logic might lead to a glitch during gate-level simulation.

[Back to Top](#)

RST_IS_DFLP

Short Message: *Signal '%h' driving reset of flip-flop '%h' is generated from flip-flop.*

Severity	Warning
Description	<p>The specified flip-flop's set/reset is generated through a flip-flop. An internally generated set/reset results in timing and testability issues. Therefore, avoid using an internally generated set/reset.</p> <p>RMM section 5.4.6 also recommends avoiding internally generated set/reset.</p>

The following code illustrates the occurrence of `RST_IS_DFLP`.

```
module mod_a(port_a,rst_a,clk, port_c);
  input port_a;
  input clk;
  input rst_a;
  output port_c;
  reg port_c;
  reg rst;
  always@(posedge clk)
    rst <= ~rst_a;
  always@(posedge clk or negedge rst)
  begin
    if (!rst)
```

JasperGold Superlint Checks Reference

Lint Checks

```
        port_c <= 1'b0;
    else
        port_c <= port_a;
    end
endmodule
```

In the above code, reset `rst` of flip-flop `port_c` is generated through a flip-flop. As a result, a violation is reported. Remodel the design.

[Back to Top](#)

RST_IS_DLAT

Short Message: *Signal '%h' driving reset of flip-flop '%h' is generated from latch.*

Severity	Warning
Description	<p>The specified flip-flop's set/reset is generated through a latch. An internally generated set/reset results in timing and testability issues. Therefore, avoid using an internally generated set/reset.</p> <p>RMM section 5.4.6 also recommends avoiding internally generated set/reset.</p>

The following code illustrates the occurrence of RST_IS_DLAT.

```
module mod_a(port_a,en,rst_a,clk, port_c);
    input port_a;
    input clk;
    input en;
    input rst_a;
    output port_c;
    reg port_c;
    reg rst;
    always@*
        if(en)
            rst <= ~rst_a;
    always@(posedge clk or negedge rst)
        begin
            if (!rst)
                port_c <= 1'b0;
```

JasperGold Superlint Checks Reference

Lint Checks

```
        else
            port_c <= port_a;
        end
    endmodule
```

In the above code, reset `rst` of flip-flop `port_c` is generated through a latch. As a result, a violation is reported. Remodel the design.

[Back to Top](#)

RST_NO_HGHI

Short Message: *The reset generation logic for clock '%s' is not at the same or a higher hierarchical level as the module/design-unit to which the clock applies.*

Severity	Warning
Description	The reset generation logic should be at the same or higher hierarchical level as the module to which the reset applies. This increases the portability of the code to different reset schemes.

The following code illustrates the occurrence of `RST_NO_HGHI`.

```
module mod_a(clk,rst,port_a,port_b,port_c);
    input clk, rst, port_a;
    input [5:0] port_b;
    output reg [5:0] port_c;
    wire rst_b;
    mod_b inst_b(rst, port_a, rst_b);
    always @(posedge clk or negedge rst_b)
        begin: block
            if(!rst_b)
                port_c = 0;
            else
                port_c = port_b;
            end
        endmodule

    module mod_b(rst_a, port_a, rst_b);
        input rst_a, port_a;
        output rst_b;
```

JasperGold Superlint Checks Reference

Lint Checks

```
mod_c inst_c (rst_a, port_a, rst_b);
endmodule
```

```
module mod_c(rst_a, port_a, rst_b);
input rst_a, port_a;
output rst_b;
assign rst_b = port_a & rst_a;
endmodule
```

In the above code, the reset generation logic for flip-flop `port_c` is at a lower hierarchical level than module `mod_a` to which the reset applies. As a result, the tool reports a violation. To avoid this violation, instantiate the reset generation module `mod_c` in module `mod_a`, or move the flip-flop `port_c` to module `mod_b`.

[Back to Top](#)

RST_NR_MULT

Short Message: *Flip-flop '%s' contains multiple asynchronous set/reset inputs.*

Severity	Warning
Description	As a good coding practice, use only one asynchronous set and reset signal per flip-flop.

The following code illustrates the occurrence of `RST_NR_MULT`.

```
module top(input rst1, rst2, din, clk, output out);
reg out;
always @(posedge clk or posedge rst1 or posedge rst2)
begin
    if (rst1)
        out <= 1'b0;
    else if (rst2)
        out <= 1'b0;
    else
        out <= din;
    end
endmodule
```

JasperGold Superlint Checks Reference

Lint Checks

In the above code, a violation is reported for the flop `out` as it is controlled by two resets, `rst1` and `rst2`.

[Back to Top](#)

RST_NR_PENA

Short Message: *Signal '%h' driving reset of flip-flop '%h' is permanently enabled.*

Severity	Error
Description	Set/reset pin is permanently enabled. This can cause a glitch in the design as the intended value might never be assigned to the output variable, which might not be detected in simulation.

The following code illustrates the occurrence of RST_NR_PENA.

```
module test(port_f,clk,rst,var_a);
input port_f,clk,rst;
output var_a;
reg var_a;
assign rst = 1'b0;
always@(posedge clk or negedge rst)
begin
if (!rst)
var_a <= 1'b0;
else
var_a <= port_f;
end
endmodule
```

In the above code, the flip-flop `var_a` has its reset permanently enabled. This can cause a glitch as `var_a` will never be assigned the intended value `port_f`. To avoid this violation, remodel the design.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

SIG_IS_INTB

Short Message: *Signal '%l' defined in module/design-unit '%m', is inferred as a tri-state buffer.*

Severity	Warning
Description	The signal defined in the module/design-unit is inferred as a tri-state buffer. This can drive the output to a high impedance state, which in turn can cause issues in timing analysis. As per Reuse Methodology Manual (RMM) conventions [Section - 3.5.2], avoid using tri-state buffers.

The following code illustrates the occurrence of SIG_IS_INTB.

```
module top (input port_a,en,output port_b);
  reg port_b;
  always @(en);
  case (en);
    1'b0 : port_b = port_a;
    1'b1 : port_b = 1'bz;
  endcase
endmodule
```

In the above code, `port_b` is inferred as a tri-state buffer.

[Back to Top](#)

JasperGold Superlint Checks Reference

Lint Checks

SIG_IS_MDRV

Short Message: *Signal/register '%l' has multiple drivers.*

Severity	Error
Description	The specified signal/register has multiple drivers that can be active simultaneously. This might lead to signal/register having undefined/unexpected value and can also result in difference in simulation and synthesis behavior.
Parameter associated	<pre>params SIG_IS_MDRV {ignore_z_drivers="yes" "no"}</pre> <p>The default value of this parameter is yes.</p> <p>When the value is set to <code>yes</code>, the check does not consider <code>z</code> as a driver. As a result, if a signal has two drivers and one of the drivers is <code>z</code>, the check is not reported.</p> <p>When the value is set to <code>no</code>, the check considers <code>z</code> as a driver. As a result, if a signal has two drivers and one of the drivers is <code>z</code>, the check is reported.</p>

The following code illustrates the occurrence of SIG_IS_MDRV.

```
module test (a, b, out1,out2);
input a,b;
output out1,out2;
reg out1, out2, temp1, clk;
always begin
    #20 clk=~clk;
end

always @(negedge clk) begin
    out1= a;
    temp1=b;
end

always @(posedge clk) begin
    out1=b;
    out2=temp1;
end
endmodule
```

In the above code, `out1` has multiple drivers `a` and `b`.

[Back to Top](#)

RACES

Race conditions occur when two events take place at the same instant of simulation time. They can cause multiple simulation results to differ, depending on the event the simulator executed first. When a model contains event-order dependent code, a race condition can cause unexpected results between successive clock cycles and mismatches between the behaviors of different simulators. This category consists of simulation race condition checks as follows:

- [REG_NR_RWRC](#) on page 251
- [REG_NR_TRRC](#) on page 252
- [REG_NR_WWRC](#) on page 253

REG_NR_RWRC

Short Message: *A read-write race exists for signal: '%s' at line no : %d and %d.*

Severity	Warning
Description	A read-write race condition occurs when a register is read in one <code>always</code> block and written in another <code>always</code> block at the same clocking condition. This causes problems in any zero-delay verification environment. Depending on the order in which the <code>always</code> blocks are scheduled, the simulation results differ. The results might also vary with different simulators.
Parameter associated	<pre>params REG_NR_RWRC {enable_check_on_combinational_logic="no"}</pre> <p>The default for this parameter is <code>no</code>. When you set this parameter to <code>yes</code>, Superlint checks the combinational logic for read-write race conditions.</p>

The following code illustrates the occurrence of `REG_NR_RWRC`.

```
module test (a, b, c, clk, sel, out1);
```

JasperGold Superlint Checks Reference

Lint Checks

```
input a;
input b;
input c;
input clk;
input [3:0] sel;
output out1;
reg out1;
reg tmp1;
reg tmp2;
    always @(posedge clk) begin
        tmp1 = a;
    end
    always @(posedge clk) begin
        tmp2 = tmp1;
    end
    always @(negedge clk) begin
        out1 = tmp1 ^ tmp2 & c;
    end
endmodule
```

In the above example, `tmp1` is being written and read at the positive edge of the clock in two different `always` blocks. Depending on the simulator algorithm, you might get different results in different simulators. To avoid this warning, remodel your design.

[Back to Top](#)

REG_NR_TRRC

Short Message: *A trigger-propagation race exists between '%s' and '%s'.*

Severity	Warning
Description	A trigger-propagation race condition occurs when a register's trigger effect propagates through two different paths. This might cause event-ordering problems in any zero-delay verification environment.

The following code illustrates the occurrence of `REG_NR_TRRC`.

```
module test ();
    reg var_b, var_c;
    reg var_d;
```

JasperGold Superlint Checks Reference

Lint Checks

```
reg var_a;
always @(var_c)
begin
    var_b= 1;
    var_b= var_a;
end
always @(var_b)
begin
    var_c= 0;
    var_c= var_d;
end
endmodule
```

There is a trigger propagation race between var_c and var_b.

[Back to Top](#)

REG_NR_WWRC

Short Message: *'%s' is written in more than one process.*

Severity	Warning
Description	A write-write race condition occurs when a register is written in more than one always block at the same clocking condition. This causes problems in any zero-delay verification environment. Depending on the order in which the always blocks are scheduled, the simulation results differ. The results might also vary with different simulators.

The following code illustrates the occurrence of REG_NR_WWRC.

```
module test (a, b, c, clk, sel, out1);
input a;
input b;
input c;
input clk;
input [3:0] sel;
output out1;
reg out1;
reg tmp1;
reg tmp2;
```

JasperGold Superlint Checks Reference

Lint Checks

```
always @(posedge clk) begin
    tmp1 = a;
end
always @(posedge clk) begin
    tmp1 = b;
    tmp2 = tmp1;
end
always @(negedge clk) begin
    out1 = tmp1 ^ tmp2 & c;
end
endmodule
```

In this example, `tmp` is being written at the positive edges of clocks in two different `always` blocks. The value of `tmp1` for `out1 = tmp1 ^ tmp2 & c` could either be `a` or `b`. You get either of these two results depending on the simulator scheduling the algorithm. To avoid this warning, remodel your design.

[Back to Top](#)

DFT Checks

The DFT domain includes the following categories:

- INTEGRATION on page 256
- DFT_FUNCTIONAL on page 259
- DFT_SHIFT_CAPTURE on page 292

INTEGRATION

Use INTEGRATION checks to verify imported block-level constraints.

The INTEGRATION category includes the following rules:

- [CON_IS_VRFD](#) on page 256
- [CON_NO_VRFD](#) on page 257
- [SIG_MS_WIDT](#) on page 258

CON_IS_VRFD

Short Message: *Constraint '%s' for block '%s' has the same value that was used during block verification.*

Severity	Info
Description	A constraint specified for a black-boxed module has been verified by using design constraints.

The following code illustrates the occurrence of CON_IS_VRFD:

```
module top(port_a);
    input port_a;
    wire sign_a
    assign sig_a = port_a;
    bot b1(.port_a(sig_a));
endmodule

Required DFT constraints in input Tcl file:
config_rtltds -signal -constant {{port_a 1}} -mode functional
```

In the above example, `b1.port_a` is assigned the value 1 using design constraints. If the `bot` block was verified using the same value and then the database was saved using the `check_superlint -save` command, the tool issues CON_IS_VRFD.

Note: In the above example, this check would be issued only when module `bot` is black boxed and its database is loaded using the `check_superlint -import` command.

[Back to Top](#)

JasperGold Superlint Checks Reference

DFT Checks

CON_NO_VRFD

Short Message: *Constraint '%s' for block '%s' either has a different value than used during block verification or is not constant.*

Severity	Error
Description	A constraint specified for a black-boxed module has not been verified using design constraints.

The following code illustrates the occurrence of CON_NO_VRFD:

```
module top(port_a);
    input port_a;
    wire sign_a
    assign sig_a = port_a;
    bot b1(.port_a(sig_a));
endmodule

Required DFT constraints in input Tcl file:
config_rtltds -signal -constant {{port_a 1}} -mode functional
```

In the above example, `b1.port_a` is assigned the value 1 using design constraints. If the `bot` block was verified using the value 0 and then the database was saved using the `check_superlint -save` command, the tool issues CON_NO_VRFD.

Note: In the above example, this check would be issued only when module `bot` is black boxed and its database is loaded using the `check_superlint -import` command.

[Back to Top](#)

JasperGold Superlint Checks Reference

DFT Checks

SIG_MS_WIDT

Short Message: *Size mismatch between the block level constraint for signal '%s' and the signal in instance '%s'. Constraint verification might not be proper.*

Severity	Info
Description	The size of a constraint imported using <code>check_superlint -import</code> does not match the size in its instantiation. This might lead to incorrect verification, and other constraint verification messages might not be enough to catch this mismatch.

The following code illustrates the occurrence of SIG_MS_WIDT:

```
BLOCK:
module bot(port_a);
    parameter param_a = 1'b0;
    input [param_a:0] port_a;
endmodule;

SOC:
module top(port_a);
    input port_a;
    wire sign_a
    assign sig_a = port_a;
    bot #(1'b1) b1(.port_a(sig_a));
endmodule
```

In the above example, when the block database is saved using `check_superlint -save`, the size of signal `port_a` is 1 bit. However, when this database is imported in SOC using `check_superlint -import`, the size of the `b1.port_a` will be 2 bits.

Note: In the above example, this check would be issued only when module `bot` is black boxed and its database is loaded using the `check_superlint -import` command.

[Back to Top](#)

DFT_FUNCTIONAL

Use DFT_FUNCTIONAL checks to verify that your RTL complies with design for testability (DFT) rules and to successfully insert DFT logic in your design. You can perform DFT_FUNCTIONAL checks at the register transfer level before synthesizing your VHDL or Verilog code thus identifying testability issues early in the design cycle.

The DFT_FUNCTIONAL category includes the following rules:

- CLK_IS_ACRE on page 260
- CLK_IS_ACRL on page 261
- CLK_IS_CDLA on page 262
- CLK_IS_CDTE on page 263
- CLK_IS_DDCF on page 264
- CLK_IS_DDCL on page 265
- CLK_IS_DLAT on page 267
- CLK_IS_DRFE on page 268
- CLK_IS_DRLA on page 269
- CLK_IS_MMCK on page 270
- CLK_IS_NDPI on page 271
- FLP_IS_ASFL on page 272
- FLP_IS_CDFF on page 273
- FLP_IS_CSTD on page 274
- FLP_IS_GATC on page 275
- FLP_IS_GTCK on page 276
- FLP_IS_TNEF on page 277
- FLP_NO_CNPI on page 278
- FLP_NO_SRST on page 279
- FLP_SR_SAME on page 279
- LAT_EN_NCPI on page 280

JasperGold Superlint Checks Reference

DFT Checks

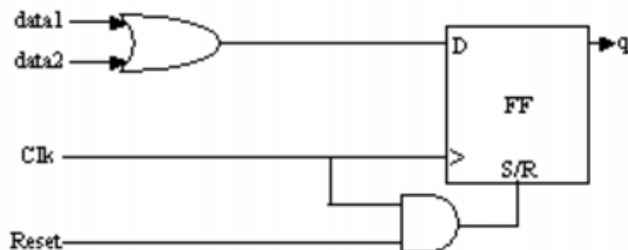
- LAT_IS_INFR on page 281
- MOD_IS_CMBL on page 283
- OTP_NO_RGTM on page 285
- RST_IS_DDAF on page 286
- RST_MX_EDGE on page 287
- RST_MX_SYAS on page 288
- TRI_NO_EPTB on page 291

CLK_IS_ACRF

Short Message: *Clock signal '%h' drives set/reset and clock pin '%h' of flip-flop '%h'.*

Severity	Warning
Description	Clock as a part of set/reset of a flip-flop leads to testability issues. This creates problems in initialization of the flip-flop and hence makes the flip-flop less controllable. This leads to low fault coverage while testing using scan chains. This condition should be either avoided or clock should be isolated in scan capture mode using the test-mode signal.

The following schematic illustrates this problem.



In the above schematic, the clock `clk` affects the set/reset of the same flip-flop. This kind of clock usage might cause testability issues, prevent successful scan chain insertion, and reduce controllability of the design.

JasperGold Superlint Checks Reference

DFT Checks

A similar check CLK_IS_CDLA exists for latches.

[Back to Top](#)

CLK_IS_ACRL

Short Message: *Clock signal '%s' drives set/reset and enable pin '%s' of latch '%s'*

Severity	Warning
Description	Clock as a part of set/reset of a latch leads to testability issues. This creates problems in initialization of the latch and hence makes the latch less controllable. This leads to low fault coverage while testing using scan chains. This condition should be either avoided or the clock should be isolated in scan capture mode using the test-mode signal.

The following code illustrates the occurrence of CLK_IS_ACRL.

```
module top (o1, o2, i);
  output o1, o2;
  input i;
  wire r, d, t;
  assign r = ~i;
  lat I1(o1, d, i, r);
  ff I2(o2, d, i, r);
  feedThru I3(d, i, t);
endmodule
```

```
module lat(o, d, e, r);
  output o;
  input d, e, r;
  reg o;
  always @(d, e, r)
    if(r)
      o <= 1'b0;
    else if(e)
      o <= d;
endmodule
```

```
module ff(o, d, c, r);
```

JasperGold Superlint Checks Reference

DFT Checks

```
output o;
input d, c, r;
reg o;
always @(posedge(c))
    if(c)
        begin
            if(r)
                o <= 1'b0;
            else
                o <= d;
        end
endmodule

module feedThru(o, i1, i2);
output o;
input i1, i2;
assign o = i1 + i2;
endmodule
```

In the above code, clock signal *i* drives set/reset and enable pin *e* of *top.I1.o*.

[Back to Top](#)

CLK_IS_CDLA

Short Message: *Clock signal '%h' drives the data pin of latch '%h'.*

Severity	Warning
Description	Clock as a part of data of a latch leads to testability issues. This makes data entering the latch unobservable. This leads to data capture issues and hence low fault coverage while testing using scan chains. This condition should either be avoided or the clock should be isolated in the scan capture mode using the test-mode signal.

The following code illustrates the occurrence of CLK_IS_CDLA.

```
module d_latch(q,d,clk,control,rst);
input control;
input d,rst;
input clk;
```

JasperGold Superlint Checks Reference

DFT Checks

```
output q;
reg q;
always @(clk or control or d)
begin
    if(control)
        q <= clk;
end
always @(posedge clk or negedge rst)
begin
    if(!rst)
        q <= 1'b 0;
    else
        q <= d;
end
endmodule
```

In the above code, `clk` is used as data pin of the latch.

[Back to Top](#)

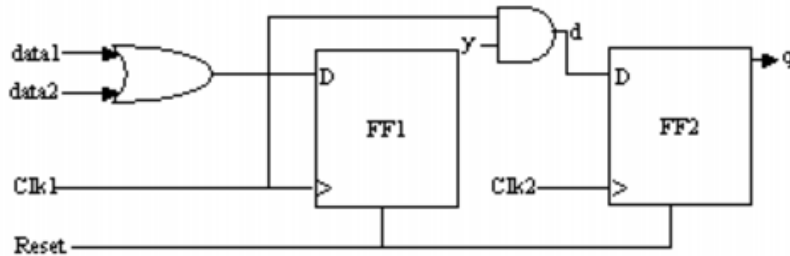
CLK_IS_CDTF

Short Message: *Clock signal '%h' drives the data pin of flip-flop '%h'.*

Severity	Warning
Description	Clock as a part of data of a flip-flop leads to testability issues. This makes data entering the flip-flop unobservable. This leads to data capture issues and hence low fault coverage while testing using scan chains. This condition should either be avoided or the clock should be isolated in the scan capture mode using the test-mode signal.

The following diagram illustrates this problem.

DFT Checks



In the above schematic diagram, the clock `clk1` affects the data of `FF2`. This kind of clock usage might cause testability issues, prevent successful scan chain insertion, and reduce controllability of the design.

A similar check `CLK_IS_CDLA` exists for latches.

[Back to Top](#)

CLK IS DDCF

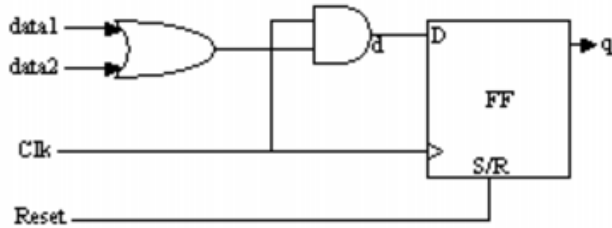
Short Message: Clock signal '%h' drives the data pin and clock pin '%h' of flip-flop '%h'.

Severity	Warning
Description	Clock as a part of data of a flip-flop leads to testability issues. This makes data entering the flip-flop unobservable. This leads to data capture issues and hence low fault coverage while testing using scan chains. This condition should either be avoided or the clock should be isolated in the scan capture mode using the test-mode signal.

The following diagram illustrates this problem.

JasperGold Superlint Checks Reference

DFT Checks



In the above schematic, the clock `clk` is driving both the clock as well as data of the same flip-flop. This kind of clock usage might cause testability issues, prevent successful scan chain insertion, and reduce controllability of the design.

A similar check `CLK_IS_DLAT` exists for latches.

[Back to Top](#)

CLK_IS_DDCL

Short Message: *Clock signal '%s' drives the data pin and enable pin '%s' of latch '%s'.*

Severity	Warning
Description	Clock as a part of data of a latch leads to testability issues. This makes data entering the latch unobservable. This leads to data capture issues and hence low fault coverage while testing using scan chains. This condition should either be avoided or the clock should be isolated in the scan capture mode using the test-mode signal.

The following code illustrates the occurrence of `CLK_IS_DDCL`.

```
module top (o1, o2, i);
  output o1, o2;
  input i;
  wire r, d, t;
  assign r = ~i;
  lat I1(o1, d, i, r);
  ff I2(o2, d, i, r);
```

JasperGold Superlint Checks Reference

DFT Checks

```
    feedThru I3(d, i, t);
endmodule
```

```
module lat(o, d, e, r);
output o;
input d, e, r;
reg o;
always @(d, e, r)
    if(r)
        o <= 1'b0;
    else if(e)
        o <= d;
endmodule
```

```
module ff(o, d, c, r);
output o;
input d, c, r;
reg o;
always @(posedge(c))
    if(c)
        begin
            if(r)
                o <= 1'b0;
            else
                o <= d;
        end
endmodule
```

```
module feedThru(o, i1, i2);
output o;
input i1, i2;
    assign o = i1 + i2;
endmodule
```

In the above code, clock signal `i` drives the pin and clock pin `e` of latch `top.I1.o`.

[Back to Top](#)

JasperGold Superlint Checks Reference

DFT Checks

CLK_IS_DLAT

Short Message: *Signal '%h' driving clock signal of flip-flop '%h' is generated from latch.*

Severity	Warning
Description	<p>The specified flip-flop's clock is generated through a latch. Internally generated clocks result in testability problems because flip-flops clocked by internally generated clocks cannot be made part of a scan chain. Thus, avoid using internally generated clocks.</p> <p>RMM section 5.4.4 also recommends avoiding internally generated clocks.</p>

The following code illustrates the occurrence of CLK_IS_DLAT.

```
module mod_a (clk, en, port_a, port_b, port_c);
input clk, en, port_a, port_b;
output reg port_c;
reg clk_a;
always @(clk or en)
begin
    if(en)
        clk_a = port_a;
end
always @(posedge clk_a)
    port_c <= port_b;
endmodule
```

In the above code, clock `clk_a` of flip-flop `port_c` is generated through a latch. As a result, a violation is reported. Remodel the design to avoid this issue.

[Back to Top](#)

JasperGold Superlint Checks Reference

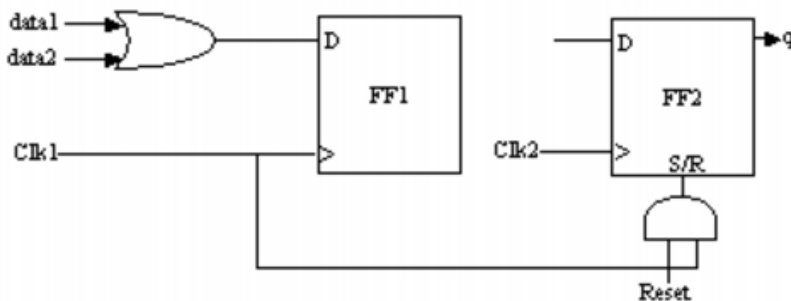
DFT Checks

CLK_IS_DRFF

Short Message: *Clock signal '%h' drives a set or reset pin of flip-flop '%h'.*

Severity	Warning
Description	Clock as a part of set/reset of a flip-flop leads to testability issues. This creates problems in initialization of the flip-flop and hence makes the flip-flop less controllable. This leads to low fault coverage while testing using scan chains. This condition should be either avoided or the clock should be isolated in the scan capture mode using the test-mode signal.

The following diagram illustrates this problem.



In this case, clock `clk1` affects the set/reset of FF2. This kind of clock use may cause testability issues, prevent successful scan chain insertion, and reduce controllability of the design.

A similar check `CLK_IS_DRLA` exists for latches.

[Back to Top](#)

CLK_IS_DRLA

Short Message: *Clock signal '%h' drives a set or reset pin of latch '%h'*

Severity	Warning
Description	Clock as a part of set/reset of a latch leads to testability issues. This creates problems in initialization of the latch and hence makes the latch less controllable. This leads to low fault coverage while testing using scan chains. This condition should be either avoided or the clock should be isolated in the scan capture mode using the test-mode signal.

The following code illustrates the occurrence of CLK_IS_DRLA.

```
module d_latch(q,d,clk,control,rst);
input control;
input d,rst;
input clk;
output q;
reg q;
always @(clk or control or d or rst)
begin
    if (rst | clk)
        q <= 1'b0;
    else if(control)
        q <= clk;
end

always @(posedge clk or negedge rst)
begin
    if(!rst)
        q <= 1'b0;
    else
        q <= d;
end
endmodule
```

In the above code, `clk` drives the reset pin of the latch `d_latch.q`.

[Back to Top](#)

CLK_IS_MMCK

Short Message: *Multiple master clocks found. Clock '%s' for flip-flop '%s' is derived from master input '%s' while the previously detected clocks were derived from '%s' for flip-flop '%s'.*

Severity	Warning
Description	Multiple clocks are being used in the design. As per RMM section 5.4, the preferred clocking structure is a single global clock. A simple clocking structure consisting of a single global clock and positive edge-triggered flip-flops is easier to understand, analyze, maintain, and synthesize.

The following code illustrates the occurrence of CLK_IS_MMCK.

```
module clk_sel (clk1, clk2, reset, din, frame_sel);
input  clk1, clk2 , reset, din;
output [1:0] frame_sel;
reg [1:0] frame_sel;
always @ ( posedge clk1  or posedge reset)
begin
    if (reset ==1)
        frame_sel[0] = 1'b0;
    else
        frame_sel[0] = din;
end
always @ ( posedge clk2  or posedge reset)
begin
    if (reset ==1)
        frame_sel[1] = 1'b0;
    else
        frame_sel[1] = din;
end
endmodule
```

In the above code, clocks `clk1` and `clk2` are used to trigger flip-flops `frame_sel[0]` and `frame_sel[1]`, respectively.

[Back to Top](#)

JasperGold Superlint Checks Reference

DFT Checks

CLK_IS_NDPI

Short Message: *Signal '%h' driving clock signal of flip-flop '%h' is not generated from any master input.*

Severity	Warning
Description	<p>The clock of the specified flip-flop is not derived from the primary input. Internally generated clocks result in testability problems because flip-flops clocked by internally generated clocks cannot be made part of a scan chain. Thus, avoid using internally generated clocks.</p> <p>RMM section 5.4.4 also recommends avoiding internally generated clocks.</p>

The following code illustrates the occurrence of CLK_IS_NDPI.

```
module mod_a (clk, rst, en, port_a, port_b);
  input clk, rst, en;
  input port_a;
  output reg port_b;
  wire clk_a;
  mod_b inst1 (clk, en, clk_a);
  always @(posedge clk_a or posedge rst)
    if (rst == 0)
      port_b = 1'b0;
    else
      port_b = port_a;
endmodule

module mod_b (clk, en, clk_b);
  input clk, en;
  output clk_b;
endmodule
```

In the above code, clock `clk_a` of flip-flop `port_b` is not driven from the primary input. As a result, a violation is reported. Remodel the design to avoid this issue.

[Back to Top](#)

JasperGold Superlint Checks Reference

DFT Checks

FLP_IS_ASFL

Short Message: *Asynchronous feedback loop detected through set/reset of flip-flop and '%s'.*

Severity	Error
Description	A feedback loop has been detected through the asynchronous set/resets of flip-flop(s) and the listed signals/wires/expressions. This check is a subset of RMM section 5.5.3, which describes how to avoid combinational feedback.

The following code illustrates the occurrence of FLP_IS_ASFL.

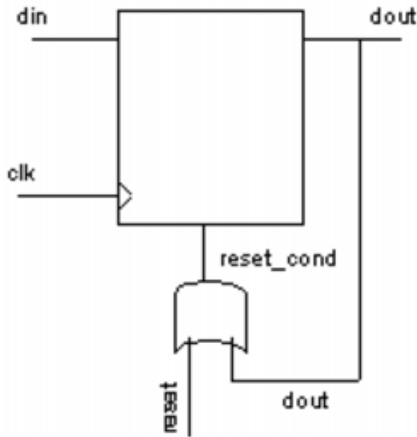
```
library ieee;
use ieee.std_logic_1164.all;
Entity dff is
  port (clk: in std_logic;
        reset: in std_logic;
        din: in std_logic;
        dout: out std_logic);
end dff;

Architecture behave of dff is
  signal reset_cond: std_logic;
  begin
    reset_cond <= dout or reset;
    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;
```

The above code will be synthesized into the following schematic.

JasperGold Superlint Checks Reference

DFT Checks



Notice that the flip-flop reset is a combination of the output as well as primary reset, resulting in a feedback loop.

[Back to Top](#)

FLP_IS_CDFF

Short Message: *Signal '%h' driving clock signal of flip-flop '%h' is generated from a flip-flop.*

Severity	Warning
Description	<p>The specified flip-flop's clock is generated through a flip-flop. Internally generated clocks result in testability problems because flip-flops clocked by internally generated clocks cannot be made part of a scan chain. Thus, avoid using internally generated clocks.</p> <p>RMM section 5.4.4 also recommends avoiding internally generated clocks.</p>

The following code illustrates the occurrence of FLP_IS_CDFF.

```
module mod_a (clk, rst, port_a, port_b, port_c);  
  input clk, rst, port_a, port_b;
```

JasperGold Superlint Checks Reference

DFT Checks

```
output reg port_c;
reg clk_a;
always @(posedge clk or posedge rst)
begin
    if(rst)
        clk_a = 1'b0;
    else
        clk_a = port_a;
    end
always @(posedge clk_a)
    port_c <= port_b;
endmodule
```

In the above code, clock `clk_a` of flip-flop `port_c` is generated through a flip-flop. Ideally, the clock for all of the flip-flops should come from the same primary input. Remodel the design to avoid this violation.

[Back to Top](#)

FLP_IS_CSTD

Short Message: *Inferred flip-flop '%s' has a constant data input.*

Severity	Warning
Description	<p>If the input value of a flip-flop is fixed, the flip-flop data cannot be tested, and therefore, fault coverage of the design is reduced. To avoid this issue, remodel the design so that there are no fixed data inputs to the flip-flop.</p> <p>The behavior of this check is controlled by the following parameters in the default Superlint rules file:</p> <pre>params FLP_IS_CSTD {output_of_latch="variable" "fixed"} params FLP_IS_CSTD {output_of_blackbox="variable" "fixed"}</pre> <p>These parameters define whether the output of a latch or a black-boxed unit (used as an input to flip-flop) should be considered fixed or variable. The default value of these parameters is variable.</p>

JasperGold Superlint Checks Reference

DFT Checks

The following code illustrates the occurrence of FLP_IS_CSTD.

```
module mod_a ( port_a, rst, port_b );
input port_a, rst;
output port_b;
reg port_b;

always @ (posedge port_a or negedge rst)
begin
    if( !rst )
        port_b <= 1'b0;
    else
        port_b <= 1'b1;
    end
endmodule
```

In the above code, the input value of flip-flop `port_b` is fixed. As a result, a violation is reported. Remodel the design to avoid this violation.

[Back to Top](#)

FLP_IS_GATC

Short Message: *Signal '%h' driving clock signal of flip-flop '%h' is generated from combinational logic.*

Severity	Info
Description	<p>The specified flip-flop's clock is gated. Improper timing of the clock gating circuit can cause a false clock or a glitch, which in turn can result in the flip-flop clocking wrong data. In addition, gated clocks can result in testability problems because flip-flops clocked by gated clocks cannot be made part of a scan chain. Therefore, avoid using gated clocks.</p> <p>RMM section 5.4.3 also recommends avoiding use of gated clocks.</p> <p>Note: The tool reports FLP_IS_GATC on gated clocks by default.</p>

The following code illustrates the occurrence of FLP_IS_GATC.

```
module mod_a (clk, rst, en, port_a, port_b);
input clk, rst, en;
```

JasperGold Superlint Checks Reference

DFT Checks

```
input port_a;
output reg port_b;
wire clk_a;
assign clk_a = en ^ clk;
always @(posedge clk_a or posedge rst)
    if (rst == 0)
        port_b = 1'b0;
    else
        port_b = port_a;
endmodule
```

In the above code, clock `clk_a` of flip-flop `port_b` is gated.

[Back to Top](#)

FLP_IS_GTCK

Short Message: *Signal '%h' driving clock signal of flip-flop '%h' is generated from combinational logic.*

Severity	Warning
Description	<p>The specified flip-flop's clock is gated. Improper timing of the clock gating circuit can cause a false clock or a glitch, which in turn can result in the flip-flop clocking wrong data. In addition, gated clocks can result in testability problems because flip-flops clocked by gated clocks cannot be made part of a scan chain. Therefore, avoid using gated clocks.</p> <p>RMM section 5.4.3 also recommends avoiding use of gated clocks.</p>
Parameter associated	<p>params FLP_IS_GTCK {identify_clock_gating_cell="yes no"}</p> <p>The default value of this parameter is no.</p> <p>By default, a clock gating cell is treated as a gated clock and the check FLP_IS_GTCK is reported. When the value of this parameter is set to yes, clock gating cells are differentiated from a gated clock and the check FLP_IS_GTCK is not reported.</p>

The following code illustrates the occurrence of FLP_IS_GTCK.

```
module mod_a (clk, rst, en, port_a, port_b);
```

JasperGold Superlint Checks Reference

DFT Checks

```
input clk, rst, en;
input port_a;
output reg port_b;
wire clk_a;
assign clk_a = en && clk;
always @(posedge clk_a or posedge rst)
    if (rst == 0)
        port_b = 1'b0;
    else
        port_b = port_a;
endmodule
```

In the above code, clock `clk_a` of flip-flop `port_b` is gated.

[Back to Top](#)

FLP_IS_TNEF

Short Message: *Flip-flop '%s' is triggered at negative-edge of clock '%s'.*

Severity	Warning
Description	Sequential elements should always be triggered on the positive edge.

The following code illustrates the occurrence of `FLP_IS_TNEF`.

```
module test (a, out1);
    inout a;
    inout out1;
    wire temp1, temp2;
    reg clk;
    mod1 m1(a,clk, temp1);
    mod1 m2(temp1,clk, out1);
    mod1 m3(out1,clk,temp2);
    mod1 m4(temp2,clk, a);
    always begin
        #20 clk=~clk;
    end
endmodule

module mod1(i1,clk, o1);
```

JasperGold Superlint Checks Reference

DFT Checks

```
input i1;
input clk;
output o1;
reg o1;
always@(negedge clk) begin
    #2 o1 = i1;
end
endmodule
```

In the above code, `o1` is triggered by `negedge` of clock `clk`.

[Back to Top](#)

FLP_NO_CNPI

Short Message: *Signal '%h' driving set/reset of flip-flop '%h' is not generated from any master input.*

Severity	Warning
Description	The asynchronous control (set/reset) of the latch/flip-flop is not directly controllable from the primary inputs. This would prevent proper scan insertion for the latch/flip-flop.

The following code illustrates the occurrence of `FLP_NO_CNPI`.

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

In the above code, `rst` is declared as output and is used as reset for register `q`. Reset should be a primary input.

JasperGold Superlint Checks Reference

DFT Checks

[Back to Top](#)

FLP_NO_SRST

Short Message: *Flip-flop '%s' does not have any set or reset.*

Severity	Warning
Description	If a flip-flop does not have any set or reset, it cannot be controlled from primary inputs. This can lead to testability issues. To avoid such issues, ensure that the flip-flop has a set or a reset.

The following code illustrates the occurrence of FLP_NO_SRST.

```
module mod_a(input port_a, input clk, output reg port_b);
  always@(posedge clk)
  begin: block_A
    port_b <= port_a;
  end
endmodule
```

In the above code, the signal `port_a` does not have any set or reset. As a result, the tool reports a violation. Remodel the design such that the flip-flop has a set or a reset.

[Back to Top](#)

FLP_SR_SAME

Short Message: *The set '%s' and reset '%s' of the flop '%s' are driven from the same signal.*

Severity	Warning
Description	Same condition is used as set and reset of the flip-flop. This will lead to set and reset being active at the same time.

The following code illustrates the occurrence of FLP_SR_SAME.

```
module top(clk, rst, in , out);
  input clk, rst, in;
```

JasperGold Superlint Checks Reference

DFT Checks

```
output out;
reg out;
assign set = rst;

always@(posedge clk or negedge rst or negedge set)
  if(!rst)
    out = 1'b0;
  else if(!set)
    out = 1'b1;
  else out = in;
endmodule
```

In the above example, the negative edge is used as both set and reset of the flop. To avoid this violation, use the right edge or right signal.

[Back to Top](#)

LAT_EN_NCPI

Short Message: *Enable of latch '%s' is not controllable from primary inputs.*

Severity	Warning
Description	The enable of latch is not directly controllable from the primary inputs. This would reduce the controllability of the latch and might create testability issues.
Parameter associated	controllable_thru_comb_logic=no yes The default value for this parameter is <code>no</code> . If the latch enable is controllable by primary inputs through some combinational logic, Superlint does not consider the latch enable as controllable and generates the LAT_EN_NCPI message. Setting a value of <code>yes</code> causes Superlint to consider the latch enable driven through combinational logic as controllable if any net in the combinational logic is controlled by a primary input.

The following code illustrates the occurrence of LAT_EN_NCPI.

```
module top(o1, o2, o3, i, d);
output o1, o2, o3;
input i, d;
```


JasperGold Superlint Checks Reference

DFT Checks

```
wire  ro1, ro2, loopClk,  loopEnable , condLoop, mo3;
    assign loopClk = i & ro1;
    assign loopEnable = i & ro2;
    assign condLoop = i & mo3;
    assign o1 = ro1;
    assign o2 = ro2;
    assign o3 = mo3;
    bot I1(ro1, ro2, mo3, loopClk, loopEnable, condLoop, d);
endmodule

module bot(ro1, ro2, mo3, loopClk, loopEnable, condLoop, d);
output ro1, ro2, mo3;
input  loopClk,  loopEnable , condLoop, d;
reg ro1, ro2;
always@(posedge loopClk)
    ro1 = d;
always @( loopEnable  or d)
    if(loopEnable)
        ro2 = d;
        assign mo3 = (condLoop) ? d : ~d;
endmodule
```

In the above code, `loopEnable` is not a primary input of module `top`. It is connected to the latch enable input of the latch `ro2`.

[Back to Top](#)

LAT_IS_INFR

Short Message: *Process/always block models a latch, or signal '%s' is not assigned a value in all branches.*

Severity	Warning
Description	Latches should be avoided in the design. Latch timing is inherently ambiguous. As a result, timing analysis of design containing latches is difficult. Over a large design, timing analysis becomes virtually impossible. Unanticipated latch may obstruct generation of clock tree or decrease fault detection rate. RMM section 5.5.2 recommends that you avoid using latches in your design.

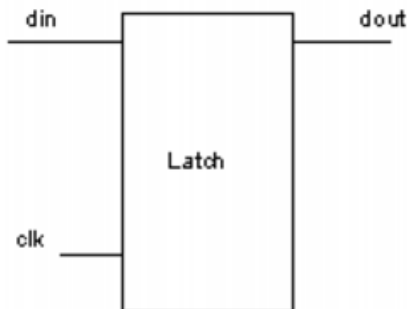
JasperGold Superlint Checks Reference

DFT Checks

Paramter associated	print_full_name=no yes The default value for this parameter is no. When the value of this parameter is set to <code>yes</code> , Superlint reports hierarchical names of the latches in the message.
----------------------------	---

The following code illustrates the occurrence of LAT_IS_INFR.

```
module test (din,dout,clk);  
  output dout;  
  input din;  
  input clk;  
  reg dout;  
  always @(clk,din) begin  if (clk == 1)  
    dout = din;  
  end  
endmodule
```



The above code and the schematic show a latch in the design. Latches should be avoided as per the RMM guidelines.

Note: This check is not flagged if this latch is used as a synchronizer.

[Back to Top](#)

JasperGold Superlint Checks Reference

DFT Checks

MOD_IS_CMBL

Short Message: *Combinational loop detected passing through: '%h'.*

Severity	Error
Description	A combinational loop has been detected involving the listed signals/wires/expressions. As per RMM section 5.5.3, you should avoid combinational feedback that is looping of combinational processes.
Parameter associated	<p>Parameters to control the behavior for MOD_IS_CMBL:</p> <p><code>disable_enumerated_loops</code>: When set to <code>no</code>, if one signal is part of multiple loops, then the tool issues messages for all loops. Setting the value to <code>yes</code> issues messages only once.</p> <p><code>loop_count_limit</code>: Controls the maximum number of messages issued. Tool stops issuing messages after the limit is reached. When the the value is 0, the will issues messages for all the loops in the design.</p> <p><code>include_latch_data</code>: Message for loops passing through data of a latch is not issued when the value of this parameter is <code>yes</code>. Setting it to <code>no</code> disables such messages.</p>

The following code illustrates the occurrence of MOD_IS_CMBL.

```
library ieee;
use ieee.std_logic_1164.all;
entity TEST is
  port ( a,b,c,d,e,f: buffer std_logic;
         o ,o2 : out std_logic
       );
end;

Architecture structural of test is
component NAND_2
  port (I0, I1: in std_logic;
        o: buffer std_logic);
end component;

component OR_2
  port (I0, I1: in std_logic;
```

JasperGold Superlint Checks Reference

DFT Checks

```
        o: buffer std_logic);
end component;

begin
    x1: NAND_2 port map (a, b,c);
    x2: OR_2 port map (c,d,b);
end structural;

library ieee;
use ieee.std_logic_1164.all;
entity nand_2 is
    port (I0, I1: in std_logic;
          o: buffer std_logic);
end;

architecture dataflow of NAND_2 is
begin
    O <= I0 nand I1;
end dataflow;

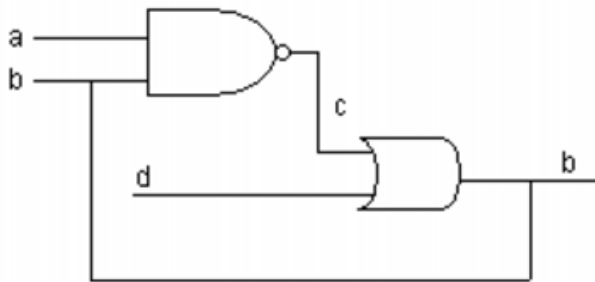
library ieee;
use ieee.std_logic_1164.all;
entity OR_2 is
    port (I0, I1: in std_logic;
          o : buffer std_logic);
end entity;

architecture dataflow of OR_2 is
begin
    O <= I0 or I1;
end dataflow;
```

In the above example, *c* is NAND of *a* and *b* and *b* is OR of *c* and *d* resulting in a combinational feedback loop between *c* and *b*. This kind of looping of combinational processes is not a good design practice. The following circuit diagram illustrates the problem:

JasperGold Superlint Checks Reference

DFT Checks



[Back to Top](#)

OTP_NO_RGTM

Short Message: *Output '%s' of top level module is not a register.*

Severity	Warning
Description	The specified output of top level module is neither a register, nor being directly driven by a register. For each block of hierarchical design, register all output signals from the block. This check is limited to top level outputs only.

The following code illustrates the occurrence of OTP_NO_RGTM.

```
module mod_a(port_a, port_b, clk, sel, port_c );
output port_b;
output port_c ;
input port_a;
input[1:0] sel;
input clk;
reg port_b;
reg port_c;
always@(clk && port_a)
begin
    if (clk == 1)
        port_b = port_a;
        case(sel)
```

JasperGold Superlint Checks Reference

DFT Checks

```
        2'b00: port_c = port_a;
        2'b11: port_c = 1'b1;
        default: port_c = 0;
    endcase
end
endmodule
```

You should change the output of the top level module to register. You should also register the output signals to increase observability.

[Back to Top](#)

RST_IS_DDAF

Short Message: Reset signal '%h' drives the data pin of %h '%h'.

Severity	Warning
Description	The specified set/reset signal drives the data pin of another flip-flop in the design. Such usage leads to timing issues. To avoid timing issues, the asynchronous set/reset signal of a flip-flop should not drive the data pin of another other flip-flop in the design.

The following code illustrates the occurrence of RST_IS_DDAF.

```
module mod_a(port_a, port_b, clk, rst, port_c);
input port_a;
input clk ;
input rst;
output port_c;
output port_b;
reg port_c;
reg port_b;
always@(posedge clk or negedge rst)
begin // signal "rst" used for reset of the FF.
    if ( rst == 1'b0 )
        port_b <= 1'b0;
    else
        port_b <= port_a;
    end
always@(posedge clk)
```

JasperGold Superlint Checks Reference

DFT Checks

```
begin // "rst" is used for operation other than reset.
    port_c <= rst;
end
endmodule
```

In the above code, reset signal `rst` of flip-flop `port_b` is also feeding flip-flop `port_c`. As a result, a violation is reported. Remodel the design to avoid this violation.

[Back to Top](#)

RST_MX_EDGE

Short Message: *Signal '%s' is used as both, active-high set/reset as well as active-low set/reset.*

Severity	Warning
Description	The specified signal is used as active high set/reset for some flip-flops/latches and as active low set/reset for some other flip-flops/latches. Both the active values of a set/reset signal should not be used on different flip-flops/latches. This causes an increase in the design area. In scan capture mode, this also leads to a reduction in the number of scan patterns.
Associated parameters	<p>This check is controlled using the following parameter in the default rules file:</p> <pre>params RST_MX_EDGE {convention_for_set_reset_style = "mixed" "asynchronous" "synchronous"}</pre> <p>The default value of this parameter is <code>mixed</code>. When the value of this parameter is set to <code>synchronous</code>, the tool reports this check for synchronous resets/sets only. When the value is set to <code>asynchronous</code>, the tool reports this check for asynchronous resets/sets only. When the value is set to <code>mixed</code>, the tool reports this check for asynchronous as well as synchronous resets/sets.</p>

The following code illustrates the occurrence of `RST_MX_EDGE`.

```
library ieee;
use ieee.std_logic_1164.all;
entity ent_a is
```

JasperGold Superlint Checks Reference

DFT Checks

```
port ( scan_enable: in std_logic;
      rst_a: in std_logic;
      clk_a: in std_logic;
      port_a: in std_logic;
      port_b: in std_logic;
      port_c: out std_logic;
      port_d: out std_logic
    );
end ent_a;
architecture arch_a of ent_a is
signal sig_b : std_logic;
begin
  proc_a :process(sig_b, clk_a)
  begin
    if(sig_b = '1') then
      port_c <= '0';
    elsif clk_a'event and clk_a = '1' then
      port_c <= port_a;
    end if;
  end process proc_a;
  proc_b :process(sig_b, clk_a)
  begin
    if(sig_b = '0') then
      port_d <= '0';
    elsif clk_a'event and clk_a = '1' then
      port_d <= port_b;
    end if;
  end process proc_b;
end arch_a;
```

In the above code, reset `sig_b` is active high reset for flip-flop `port_c` and at the same time active low reset for flip-flop `port_d`. The same signal should not be active high set/reset and active low set/reset at the same time. Remodel the design to overcome this problem.

[Back to Top](#)

RST_MX_SYAS

Short Message: *Design uses a mix of both synchronous and asynchronous set/reset, these are synchronous reset: '%h' and asynchronous reset: '%h'.*

JasperGold Superlint Checks Reference

DFT Checks

Severity	Warning
Description	The design is using a mix of synchronous and asynchronous resets/sets for latches and flip-flops. This can lead to issues in timing closure. To avoid such issues, all the resets/sets should be asynchronous or all should be synchronous.
Associated parameters	<p>This check is controlled using the following parameter in the default rules file:</p> <pre>params RST_MX_SYAS {same_reset_line = "yes" "no"}</pre> <p>The default value of this parameter is no. By default, this check reports the use of asynchronous and synchronous resets/sets in the entire design. However, you can report the use of asynchronous and synchronous reset/set for the same signal by setting the parameter value to yes.</p>

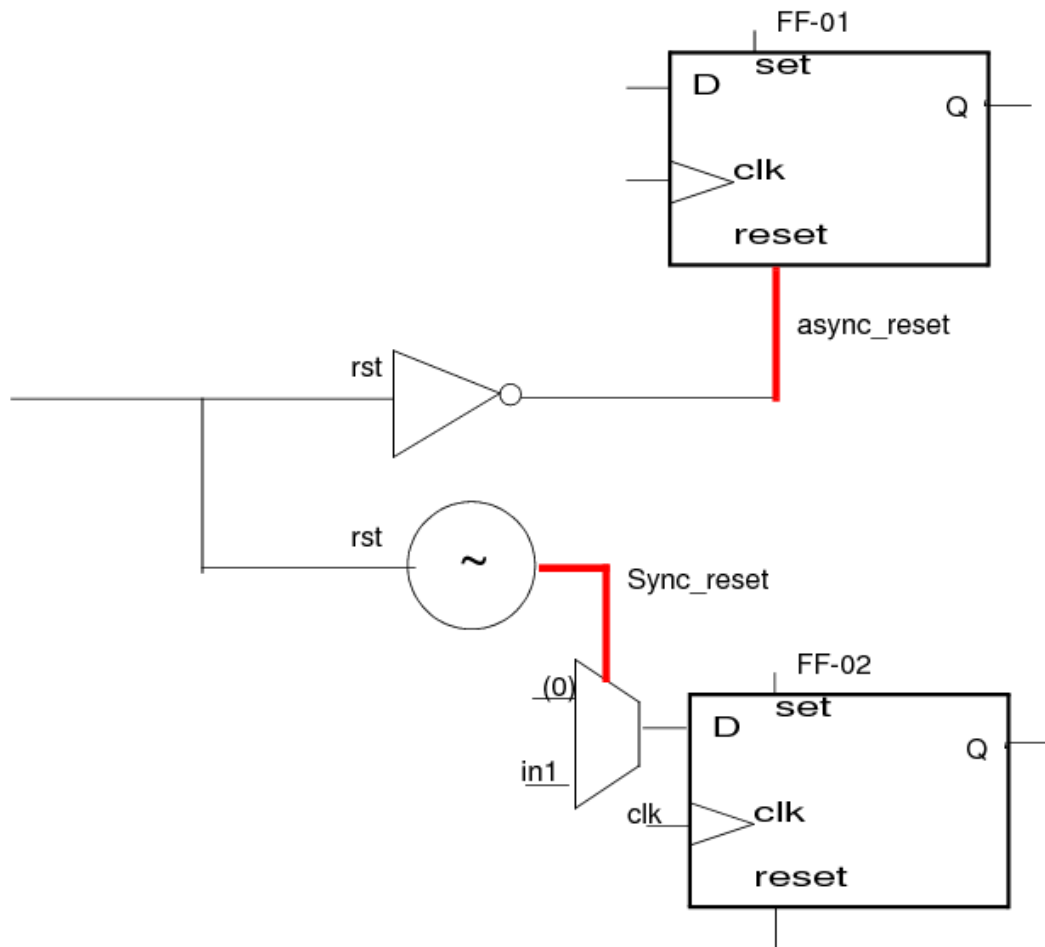
The following code illustrates the occurrence of RST_MX_SYAS.

```
module test(input clk, rst, in1, output reg out1, out2)
always @ (posedge clk or negedge rst)
    if (!rst)
        out1 <= 1'b0;
    else
        o ut1 <= in1;
always @ (posedge clk)
    if(!rst)
        out2 <= 1'b0;
    else
        out2 <= in1;
endmodule
```

In the above code, `rst` is used both as an asynchronous reset in flip-flop `out1` and as a synchronous reset in flip-flop `out2`. This problem is captured in the following schematic:

JasperGold Superlint Checks Reference

DFT Checks



[Back to Top](#)

JasperGold Superlint Checks Reference

DFT Checks

TRI_NO_EPTB

Short Message: *Signal '%h' driving enable pin of the tri-state buffer '%h' is not directly controllable by primary input(s).*

Severity	Warning
Description	<p>The enable condition of the specified tristate buffer is not directly controllable by primary inputs. This will reduce the controllability of the tristate buffer and create testability issues.</p> <p>By default, if the enable of a tristate buffer is controllable by primary inputs through some combinational logic, it is not considered controllable. However, you can change this default behavior by setting the value of the following parameter in the default rules file to <code>yes</code>:</p> <pre>params TRI_NO_EPTB {controllable_thru_comb_logic=no}</pre> <p>When the value of this parameter is set to <code>yes</code>, the enable of a tristate buffer driven through combinational logic is considered controllable if any net in the combinational logic is controlled by a primary input.</p>

The following code illustrates the occurrence of TRI_NO_EPTB.

```
module TRI_NO_EPTB(input en_a, en_b, clk, in1, output out1, out2);
  wire wir_a;
  reg reg_a;
  assign wir_a = en_a || en_b;
  always @ (posedge clk)
    reg_a = en_a;
  assign out1 = wir_a ? in1 : 1'bz;
  assign out2 = reg_a ? in1 : 1'bz;
endmodule
```

In the above code, the tristate buffer `out1` is controllable by primary inputs through combinational logic, and the tristate buffer `out2` is controlled by a flip-flop. When the value of the parameter `controllable_thru_comb_logic` is set to `no`, a violation is reported for both tristate buffers. When the value of this parameter is set to `yes`, the enable pin of `out1` is considered controllable, and a violation is reported for `out1` only.

[Back to Top](#)

DFT_SHIFT_CAPTURE

In addition to the set of DFT_FUNCTIONAL checks, Superlint enables you to perform advanced DFT analysis, which enables you to perform testability checks on the RTL even without specifying any test mode signals. In this mode, clock and reset controllability checks are done by recognizing the test circuit and clock gates automatically. Additionally clock and reset generation structures are also checked for testability.

Note that when you enable DFT_SHIFT_CAPTURE checks, the overlapping checks in the DFT_FUNCTIONAL category, if any, are not reported.

The DFT_SHIFT_CAPTURE category includes the following checks:

- CCN_IS_VRFD on page 294
- CCN_NO_VRFD on page 294
- CLK_FF_CDCD on page 295
- CLK_NC_GTEN on page 296
- CLK_NR_DSFF on page 297
- CLK_NR_RSTN on page 298
- CLK_TM_DCDL on page 299
- CLK_TM_DDFE on page 300
- CLK_TM_DDLA on page 301
- CLK_TM_DMCD on page 302
- CLK_TM_PROP on page 303
- FLP_NO_CTCL on page 304
- FLP_NO_SCAN on page 305
- FLP_NR_DBBM on page 306
- FLP_NR_UVSC on page 307
- ICG_IS_CTSC on page 308
- LAT_NO_TRTM on page 309
- MEM_NC_CKCT on page 310
- MEM_NC_INPC on page 311

JasperGold Superlint Checks Reference

DFT Checks

- MEM_NC_OTPC on page 312
- MEM_NO_FFMC on page 313
- MEM_NO_MCFE on page 315
- MEM_NR_MCLK on page 316
- MEM_SM_MCMB on page 317
- RST_NC_CTCL on page 319
- RST_NR_CKSC on page 320
- RST_NR_NCTL on page 321
- RST_NR_TCLK on page 322
- RST_TM_NCSC on page 323
- SCN_IS_VRFD on page 324
- SCN_NO_VRFD on page 325
- SIG_CM_DNCF on page 326
- SIG_CT_PROP on page 327
- SIG_NR_ICGR on page 328
- SIG_SM_DNCF on page 329
- SIG_ST_PROP on page 330
- TRI_NC_ENSS on page 331
- TRI_NO_DZSS on page 332

JasperGold Superlint Checks Reference

DFT Checks

CCN_IS_VRFD

Short Message: *In capture mode, constraint '%s' for block '%s' has the same value that was used during block verification.*

Severity	Info
Description	A constraint specified for a black-boxed module has been verified by using design constraints.

The following code illustrates the occurrence of CCN_IS_VRFD:

```
module top(port_a);
    input port_a;
    wire sign_a
    assign sig_a = port_a;
    bot b1(.port_a(sig_a));
endmodule
Required DFT constraints in input Tcl file:
config_rtltds -signal -constant {{port_a 1}} -mode capture
```

In the above example, `b1.port_a` is assigned the value 1 using design constraints. If the `bot` block was verified using the same value in capture mode and then the database was saved using the `check_superlint -save` command, the tool issues CCN_IS_VRFD.

Note: In the above example, this check would be issued only when module `bot` is black boxed and its database is loaded using the `check_superlint -import` command.

[Back to Top](#)

CCN_NO_VRFD

Short Message: *In capture mode, constraint '%s' for block '%s' either has a different value than used during block verification or is not constant'.*

Severity	Error
Description	A constraint specified for a black-boxed module has not been verified using design constraints.

JasperGold Superlint Checks Reference

DFT Checks

The following code illustrates the occurrence of CCN_NO_VRFD:

```
module top(port_a);
    input port_a;
    wire sign_a
    assign sig_a = port_a;
    bot b1(.port_a(sig_a));
endmodule
Required DFT constraints in input Tcl file:
config_rtlids -signal -constant {{port_a 1}} -mode capture
```

In the above example, `b1.port_a` is assigned the value 1 using design constraints. If the `bot` block was verified using the value 0 and then the database was saved using the `check_superlint -save` command, the tool issues `CCN_NO_VRFD` in capture mode.

Note: In the above example, this check would be issued only when module `bot` is black boxed and its database is loaded using the `check_superlint -import` command.

[Back to Top](#)

CLK_FF_CD CD

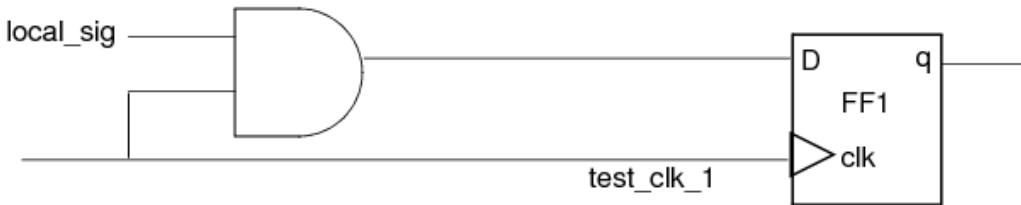
Short Message: *In shift mode, clock signal '%h' drives the data pin and clock pin %h of flip-flop '%h'.*

Severity	Error
Description	Clock as a part of data of a flop-flop leads to testability issues. This makes data entering the flip-flop unobservable. As a result, while testing using scan chains, data capture and low fault coverage issues arise. Such conditions should be avoided or the clock should be isolated in the scan capture mode using the shift-mode signal.

The following diagram illustrates this problem.

JasperGold Superlint Checks Reference

DFT Checks



In the above diagram, test clock `test_clk_1` is driving the data pin and the clock pin of flip flop FF1.

[Back to Top](#)

CLK_NC_GTEN

Short Message: *In scan mode, clock gate '%h' is not enabled.*

Severity	Warning
Description	In scan shift mode, if the clock to a sequential element is disabled, it can lead to issues in scan chain insertion. If a clock gating cell is used to pass clock to a sequential element, the enable of the gating cell must be enabled or directly controlled from top. The enable of a clock gate is the data port of the latch used to create the clock gate circuit.

The following code illustrates the occurrence of CLK_NC_GTEN.

```
module test(clk,rst,en,var_a,port_d);
input  clk,rst,en,port_d;
output var_a;
wire  clk_out;
reg   var_a,latch_out;
assign en = 1'b0;
always @en
if(!clk)
latch_out = en;
assign clk_out = clk & latch_out;
always @(posedge clk_out or negedge rst)
```


JasperGold Superlint Checks Reference

DFT Checks

```
begin
  if (!rst)
    var_a <= 1'b0;
  else
    var_a <= port_d;
  end
endmodule
```

In the above code, `clk_out` is clock to the flip-flop `var_a`, which is coming from a clock gating cell, and the enable of clock gating cell `en`, which is the data port of latch `latch_out`, is set to zero. Due to this, the flop-flop `var_a` is disabled and cannot be a part of scan chain. To avoid this violation, modify the design so that the `en` is set to one or a top-level input in scan shift mode.

[Back to Top](#)

CLK_NR_DSFF

Short Message: *In capture mode, flip-flop '%s' is driving its own clock.*

Severity	Warning
Description	The clock pin of the flip-flop is driven by the same clock. This usually happens in clock divider or generators circuits. It can be an issue if timing is not properly addressed. The clock can become glitchy or it can be missed altogether.

The following code illustrates the occurrence of CLK_NR_DSFF.

```
module test (rst_a, sig_a, port_b, port_a);
  output port_b;
  reg port_b;
  input port_a;
  input rst_a, sig_a;
  reg clk_a;
  reg sig_b;
  wire gtd_clk_a;

  always@ (clk_a)
  begin
    if (clk_a)
```

JasperGold Superlint Checks Reference

DFT Checks

```
sig_b = sig_a;
end
assign gtd_clk_a = clk_a & sig_b;
always@(posedge gtd_clk_a or negedge rst_a)
begin
if(!rst_a)
clk_a = 1'b0;
else
clk_a = #20 !clk_a;
end

always@(posedge clk_a)
port_b = port_a;

endmodule
```

In the above example, flip-flop `clk_a` is driving its own clock `gtd_clk_a`. To avoid this violation, remodel the design.

[Back to Top](#)

CLK_NR_RSTN

Short Message: *In capture mode, set/reset and clock of flip-flop '%h' are transitioning together.*

Severity	Warning
Description	<p>In scan capture mode, set/reset of the flip-flops must not transition at the same time as the clock as this can lead to testability issues. The tool issues this check in SHIFT_CAPTURE mode only.</p> <p>Use the following parameter to specify whether to test any flip-flop or reset synchronizers only:</p> <pre>params CLK_NR_RSTN {check_on="reset_sync_only" "any_ff"}</pre> <p>By default, the tool tests reset synchronizers only.</p>

The following code illustrates the occurrence of CLK_NR_RSTN.

```
module test(clk,rst_a,rst_b,var_b,var_a);
```

JasperGold Superlint Checks Reference

DFT Checks

```
input clk,var_a;
output rst_a,rst_b,var_b;
reg rst_a,rst_b,var_b;
always @(posedge clk or negedge rst_a)
if (!rst_a)
rst_a = 1'b0;
else
rst_a = 1'b1;

always @(posedge clk or negedge rst_a)
if (!rst_a)
rst_b = 1'b0;
else
rst_b = rst_a;

always @(posedge clk or negedge rst_b)
if(!rst_b)
var_b = 1'b0;
else
var_b = var_a;
endmodule
```

In the above code, the flop `var_b`, reset `rst_b`, and clock `clk` are transitioning at the same time. This can lead to testability issues.

[Back to Top](#)

CLK_TM_DCDL

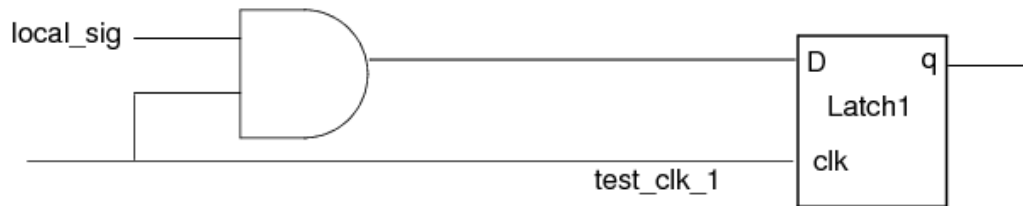
Short Message: *In shift mode, clock signal '%s' drives the data pin and enable pin %s of latch '%s'.*

Severity	Warning
Description	Clock as a part of data of a latch leads to testability issues. This makes data entering the latch unobservable. As a result, while testing using scan chains, data capture and low fault coverage issues arise. Such conditions should be avoided or the clock should be isolated in the scan capture mode using the shift-mode signal.

JasperGold Superlint Checks Reference

DFT Checks

The following diagram illustrates this problem.



In the above diagram, test clock `test_clk_1` is driving the data pin and the clock pin of latch `Latch1`.

[Back to Top](#)

CLK_TM_DDFF

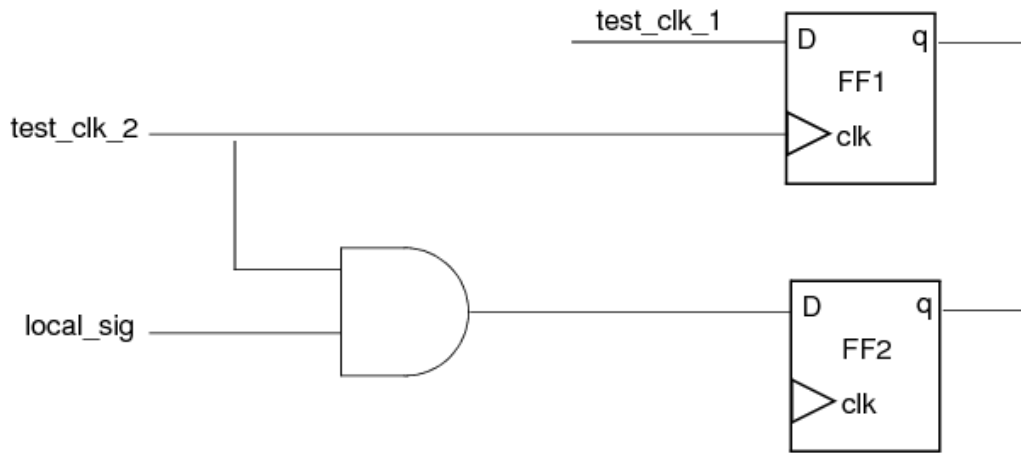
Short Message: *In shift mode, clock signal '%h' drives the data pin of flip-flop '%h'.*

Severity	Warning
Description	Clock as a part of data of a flop-flop leads to testability issues. This makes data entering the flip-flop unobservable. As a result, while testing using scan chains, data capture and low fault coverage issues arise. Such conditions should be avoided or the clock should be isolated in the scan capture mode using the shift-mode signal.

The following diagram illustrates this problem.

JasperGold Superlint Checks Reference

DFT Checks



In the above diagram, test clock `test_clk_2` is feeding data of flip flop FF2.

[Back to Top](#)

CLK_TM_DDLA

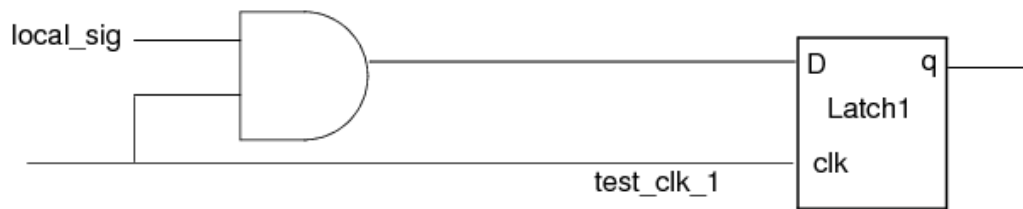
Short Message: *In shift mode, clock signal '%h' drives the data pin of latch '%h'.*

Severity	Warning
Description	Clock as a part of data of a latch leads to testability issues. This makes data entering the latch unobservable. As a result, while testing using scan chains, data capture and low fault coverage issues arise. Such conditions should be avoided or the clock should be isolated in the scan capture mode using the shift-mode signal.

The following diagram illustrates this problem.

JasperGold Superlint Checks Reference

DFT Checks



In the above diagram, test clock `test_clk_1` is feeding data of latch `Latch1`.

[Back to Top](#)

CLK_TM_DMCD

Short Message: *In capture mode, test clock '%h' drives multiple functional clocks, these are: '%g'.*

Severity	Error
Description	For testability purposes, one test clock should control flip-flops of one clock domain only in the scan capture mode. This is required for testing path delays. In addition, it helps in avoiding timing problems during scan shift and scan capture operations.
Parameter associated	params CLK_TM_DMCD {flipflop_reporting_limit=10} This parameter controls the number of flip-flops to be reported with CLK_TM_DMCD message. The default limit of this parameter is 10.

The following code illustrates the occurrence of CLK_TM_DMCD.

```
module top (clk_a, clk_b, test_clk_a, scan_enable, rst, port_h, port_i);
input clk_a, clk_b, test_clk_a, scan_enable, rst;
output port_h, port_i;
reg port_h, port_i;
wire clk, clk_c, var_a, var_b;
assign clk = (scan_enable) ? test_clk_a : clk_a;
assign clk_c = (scan_enable) ? test_clk_a : clk_b;
always@(posedge clk or negedge rst)
```

JasperGold Superlint Checks Reference

DFT Checks

```
begin : blk_a
  if (!rst)
    begin
      port_h <= 4'b0;
    end
  else
    begin
      port_h <= var_a;
    end
  end
always@(posedge clk_c or negedge rst)
begin : blk_b
  if (!rst)
    begin
      port_i <= 4'b0;
    end
  else
    begin
      port_i <= var_b;
    end
  end
end
endmodule
```

In the above code, flip-flops are in different clock domains. However, in scan capture mode, they are controlled by the same test clock, `test_clk_a`. Remodel the design so that different test clocks control flip-flops of different clock domains.

[Back to Top](#)

CLK_TM_PROP

Short Message: *In capture mode, Instance '%s' is stopping propagation of test clock '%s'.*

Severity	Info
Description	Combinational logic present in the instance has stopped forward propagation of test clock in the design. This could lead to clock controllability violations being thrown in the design as intended test clock might not reach its destination flip-flops

JasperGold Superlint Checks Reference

DFT Checks

The following code illustrates the occurrence of CLK_TM_PROP.

```
module test (test_clk, enable, port_a, port_b);
  input test_clk, enable, port_b;
  output port_a;
  reg port_a;
  wire test_clk_a;
  assign test_clk_a = test_clk & enable;
  always@(posedge test_clk_a)
    port_a = port_b;
endmodule

Required DFT constraints in input Tcl file:
config_rtltds -test_clock test_clk;
```

In the above example, due to an AND gate between `test_clk` and `enable`, the intended test clock `test_clk` is not reaching its destination flip-flop, so this instance is stopping the propagation of test clock.

[Back to Top](#)

FLP_NO_CTCL

Short Message: *In capture mode, clock signal '%h' driving flip-flop '%h' is not controlled by any test clock.*

Severity	Error
Description	For testability purposes, all the flip-flops in the design must be controllable from a test clock in the scan capture mode. This is required for testing path delays. In addition, it helps in avoiding timing problems during scan shift and scan capture operations.
Parameter associated	params FLP_NO_CTCL {flipflop_reporting_limit=10} This parameter controls the number of flip-flops to be reported with FLP_NO_CTCL message. The default limit of this parameter is 10.

The following code illustrates the occurrence of FLP_NO_CTCL.

```
module top (clk_a, test_clk_a, scan_enable, rst, port_h);
  input clk_a, test_clk_a, scan_enable, rst;
  output port_h;
  reg port_h, port_i;
```


JasperGold Superlint Checks Reference

DFT Checks

```
wire clk, var_a;
assign clk = (scan_enable) ? clk_a : test_clk_a;
always@(posedge clk or negedge rst)
begin : blk_a
  if (!rst)
    begin
      port_h <= 4'b0;
    end
  else
    begin
      port_h <= var_a;
    end
end
endmodule
```

The flip-flop in the above code is not controlled by any test clock. Remodel the design so that the flip-flop is controllable from a test clock in the scan capture mode.

[Back to Top](#)

FLP_NO_SCAN

Short Message: *In shift mode, flip-flop '%s' is not scannable.*

Severity	Error
Description	A flip-flop cannot be scanned if its asynchronous set/reset and/or clock are not controllable during scan shift mode.

[Back to Top](#)

JasperGold Superlint Checks Reference

DFT Checks

FLP_NR_DBBM

Short Message: *Data pin of the flip-flop '%s' is driven by the signal '%s', which is either undriven or output of black box module.*

Severity	Warning
Description	Data pin of flip-flop is either undriven or it is driven by a black-boxed instance. This can lead to issues in testability of the design as data is now free to take any value.

The following example illustrates the occurrence of FLP_NR_DBBM:

```
module test (clk, rst, var_a);
  input clk, rst;
  output var_a;
  reg var_a;
  wire sig_a;

  always@(posedge clk or negedge rst)
  begin
    if (!rst)
      var_a <= 1'b0;
    else
      var_a <= sig_a;
    end
  endmodule
```

In the given example, data pin of the flip-flop `var_a` is driven by a undriven wire `sig_a`. To avoid this violation, remodel the design.

[Back to Top](#)

FLP_NR_UVSC

Short Message: *In capture mode, data pin of the flip-flop '%s' is driven by the signal '%s', which is either undriven or output of black box module.*

Severity	Warning
Description	Data pin of flip-flop is either undriven or it is driven by a black-boxed instance. This can lead to issues in testability of the design as data is now free to take any value in capture mode.

The following example illustrates the occurrence of FLP_NR_UVSC:

```
module test (clk,rst,var_a);
  input clk;
  input rst;
  output reg var_a;
  wire sig_a;
  always@(posedge clk or negedge rst)
  begin
    if (!rst)
      var_a <= 1'b0;
    else
      var_a <= sig_a;
    end
  endmodule
```

In the given example, data pin of the flip-flop `var_a` is driven by undriven wire `sig_a`. To avoid this violation, remodel the design.

[Back to Top](#)

JasperGold Superlint Checks Reference

DFT Checks

ICG_IS_CTSC

Short Message: *In capture mode, clock gate '%s' is not controlled from top.*

Severity	Warning
Description	<p>In scan capture mode, if a clock gating cell is used to pass the clock to a sequential element, then the enable of the gating cell must be driven by a scannable flop or directly from top. The enable of a clock gate is the data port of the latch used to create the clock gate circuit.</p> <p>ICG_IS_CTSC is not issued when enable is constant 1; however, the following parameter can be used to issue the check when enable is constant 1. The default value of the parameter is <code>no</code>.</p> <pre>params ICG_IS_CTSC {enable_on_all_constant="no" "yes"}</pre>

The following example illustrates the occurrence of ICG_IS_CTSC:

```
module test(clk,rst,var_a,port_d);
input clk,rst,port_d;
output var_a;
reg var_a;
reg en;
reg latch_out;
assign en = 1'b0;
always @en
if(!clk)
latch_out = en;
assign clk_out = clk & latch_out;
always @(posedge clk_out or negedge rst)
begin
if (!rst)
var_a <= 1'b0;
else
var_a <= port_d;
end
endmodule
```

JasperGold Superlint Checks Reference

DFT Checks

In the above code, `clk_out` is clock to the flip-flop `var_a`, which is coming from a clock gating cell, and the enable of clock gating cell `en`, which is the data port of latch `latch_out`, is set to zero. Thus, the flip-flop `var_a` is disabled and cannot be a part of the scan chain.

To avoid this violation, modify the design so that the `en` is driven by a scannable flop or a top-level input in scan capture mode.

[Back to Top](#)

LAT_NO_TRTM

Short Message: *In shift mode, latch '%h' is not transparent.*

Severity	Warning
Description	For improved test coverage, latches should be transparent in scan capture mode. Latches that are not transparent lead to X propagation and reduce test coverage.

The following code illustrates the occurrence of LAT_NO_TRTM.

```
module mod_a(en, scan_enable, port_a,port_c);
input scan_enable;
input en;
input port_a;
output port_c;
reg port_c;
wire en_test;
assign en_test = scan_enable & en;
always @ (en_test or port_a)
    if ( en_test )
        port_c = port_a; // Latch port_c is not transparent
endmodule

Required DFT constraints in input Tcl file:
config_rtltds -signal -constant {{scan_enable 1'b0}} -mode shift
```

In the above code, a violation is reported for latch `port_c` because its enable is not always active high in the scan capture mode. To eliminate this issue, consider modifying the code so that the latch is transparent in scan capture mode. The following modified code eliminates this issue:

```
module mod_a(en, scan_enable, port_a,port_c);
```

JasperGold Superlint Checks Reference

DFT Checks

```
input scan_enable;
input en;
input port_a;
output port_c;
reg port_c;
wire en_test;
assign en_test = scan_enable | en;
always @ (en_test or port_a)
    if ( en_test )
        port_c = port_a;
endmodule
```

[Back to Top](#)

MEM_NC_CKCT

Short Message: *In shift mode, clock Signal '%h' driving flip-flop '%h' is not directly controllable from top.*

Severity	Warning
Description	In scan shift mode, the clock of a flip-flop/latch/memory should be directly controlled from top. If the clock is not directly controlled from top, it can lead to issues during scan insertion.

The following code illustrates the occurrence of MEM_NC_CKCT.

```
module test(clk,test_clk,rst,var_a,port_d);
input clk,rst,port_d,test_clk;
output var_a;
reg var_a;
assign clk = test_clk | clk;
always@(posedge clk or negedge rst)
begin
if (!rst)
var_a <= 1'b0;
else
var_a <= port_d;
end
endmodule
```

JasperGold Superlint Checks Reference

DFT Checks

In the above example `clk` is a clock to the flip-flop `var_a`, and in `scan_shift` mode, it is not directly controllable from top, which might lead to issues in scan insertion. To avoid this violation, `clk` can be avoided in scan shift mode using a clock bypass signal, as shown below:

```
assign clk_a = (clock_bypass) ? test_clk : clk;
```

Now `clk_a` can be a clock to any element, which in `scan_shift` is driven by `test_clk`.

[Back to Top](#)

MEM_NC_INPC

Short Message: *Input '%h' of the memory cell '%i' is driven by output '%h' of the memory cell '%i'.*

Severity	Warning
Description	Input of a memory is being driven by another memory cell in scan capture mode. This means that a path connecting two memory cells is present in the design and should be avoided.

The following code illustrates the occurrence of MEM_NC_INPC.

```
module top(clk_a, enable_a, port_a, port_c);
input clk_a, enable_a, port_a;
output port_c;
wire temp_a;
mem_a mem_inst (.mem_clk_a(clk_a), .en_a(enable_a), .mem_port_a(port_a),
.mem_port_c(temp_a));
mem_a mem_inst2 (.mem_clk_a(clk_a), .en_a(enable_a), .mem_port_a(temp_a),
.mem_port_c(port_c));
endmodule

module mem_a(mem_clk_a,en_a,mem_port_a,mem_port_c);
input mem_clk_a,en_a,mem_port_a;
output mem_port_c;
endmodule
```

In the above code, `mem_a` is a memory cell with `mem_port_a` as input and `mem_port_c` as output, and `temp_a` is driven by `port_a` and is also driving `port_c` via memory cell. This creates a memory to memory path.

JasperGold Superlint Checks Reference

DFT Checks

[Back to Top](#)

MEM_NC_OTPC

Short Message: *In shift mode, write_enable, chip_select, and write_clock of the memory cell '%s' are not bypassed.*

Severity	Warning
Description	<p>The state of memory must be controllable in scan capture mode, that is, the output of memory should not affect the design. To ensure this, do the following:</p> <ul style="list-style-type: none">■ Bypass the memory output in scan capture mode.■ Disable the write enable of memory.■ Disable the chip select of memory.■ Disable the write clock of memory.

The following code illustrates the occurrence of MEM_NC_OTPC.

```
module test ( clk, rst, port_d,scan_enable);
input clk,rst,scan_enable;
output port_d;
reg port_e;
reg port_d;
reg port_f;
wire sig_a;
wire sig_b;

mod a inst_a (.port_e(poer_e), .port_f(port_f), .scan_enable(scan_enable),
.clk(clk));
assign sig_a = scan_enable ? port_e : sig_b;

always@(posedge clk or negedge rst)
begin: blk_a
if(rst == 1'b0)
port_d <= 1'b0;
else
port_d <= sig_a;
```


JasperGold Superlint Checks Reference

DFT Checks

```
end
endmodule

module mod_a (port_e, port_f, clk, scan_enable);

input port_f;
output reg port_e;
input clk;
input scan_enable;

endmodule
```

Required DFT constraints in input Tcl file

```
config_rtltds -memory_cell mod_a -input port_f -output port_e
-write_enable scan_enable -write_enable_value 1'b1 -write_clock clk
```

In the above code, during scan capture mode `sig_a` is assigned the value `port_e`, which is connected to the output of memory cell `inst_a`. Here, the output of the memory cell `mod_a` is affecting the design in scan capture mode and needs to be corrected.

[Back to Top](#)

MEM_NO_FFMC

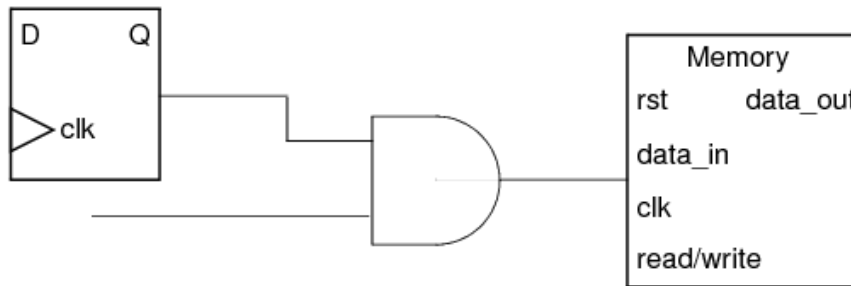
Short Message: *In capture mode, input '%s', of memory cell '%s', is not registered.*

Severity	Warning
Description	The output of a memory cell is x, which reduces the test coverage. To prevent the propagation of x inside the design during the scan capture mode, the output of a memory cell must be bypassed. You can do this by directly connecting the inputs and outputs of a memory cell to a flip-flop. If the input and output of a memory cell are directly connected to flip-flops, then in scan capture mode, the memory cell is bypassed and a scan chain is formed.

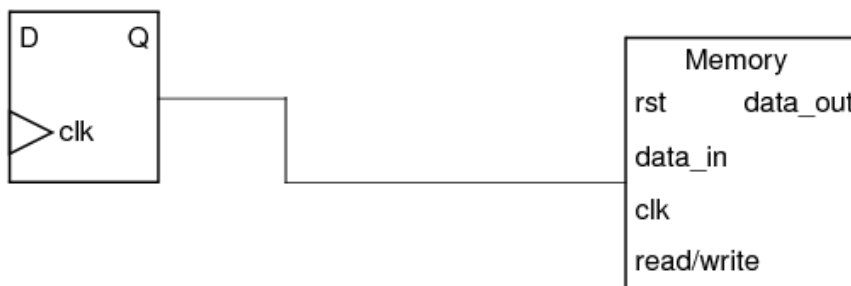
The following diagram illustrates this problem.

JasperGold Superlint Checks Reference

DFT Checks



In the above diagram, input to the memory cell is not directly connected to a flip-flop. As a result, the input of the memory cell cannot be bypassed. Consider modifying the schematic such that the input of memory cell comes from a flip-flop, as shown below.



[Back to Top](#)

JasperGold Superlint Checks Reference

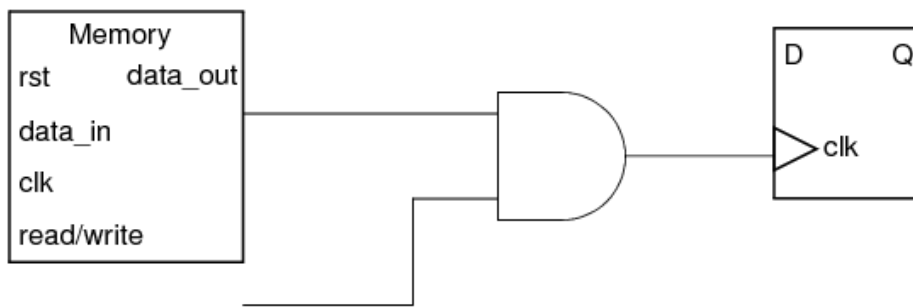
DFT Checks

MEM_NO_MCFF

Short Message: *In capture mode, output '%s', of memory cell '%s', is not registered.*

Severity	Warning
Description	The output of a memory cell is x, which reduces the test coverage. To prevent the propagation of x inside the design during the scan capture mode, the output of a memory cell must be bypassed. You can do this by directly connecting the inputs and outputs of a memory cell to a flip-flop. If the input and output of a memory cell are directly connected to flip-flops, then in scan capture mode, the memory cell is bypassed and a scan chain is formed.

The following diagram illustrates this problem.



In the above diagram, output of the memory cell is not directly connected to a flip-flop. As a result, the output of the memory cell cannot be bypassed. Consider modifying the schematic such that the output of memory cell comes from a flip-flop, as shown below.

JasperGold Superlint Checks Reference

DFT Checks



[Back to Top](#)

MEM_NR_MCLK

Short Message: *In capture mode clock '%h' of flip-flop '%h' is driven by multiple test clocks.*

Severity	Warning
Description	The clock of a flip-flop/latch/memory in scan capture should be driven by only one test clock. If multiple test clocks are driving the clock port, it can lead to issues in scan capture mode.

The following code illustrates the occurrence of MEM_NR_MCLK.

```
module test(clk,test_clk_a,test_clk_b,rst,var_a,port_d);
input clk,rst,test_clk_a,test_clk_b,port_d;
output var_a;
reg var_a;
assign clk = test_clk_a | test_clk_b;
always@(posedge clk or negedge rst)
begin
    if (!rst)
        var_a <= 1'b0;
    else
        var_a <= port_d;
    end
endmodule
```

JasperGold Superlint Checks Reference

DFT Checks

Required DFT constraints in input Tcl file:

- `config_rtltds -test_clock test_clk_a`
- `config_rtltds -test_clock test_clk_b`

In the above code, `clk` is the clock to the flip-flop `var_a` and is driven by test clocks `test_clk_a` and `test_clk_b`. To avoid this violation, assign `clk` to either `test_clk_a` or `test_clk_b`, and not to both.

[Back to Top](#)

MEM_SM_MCMB

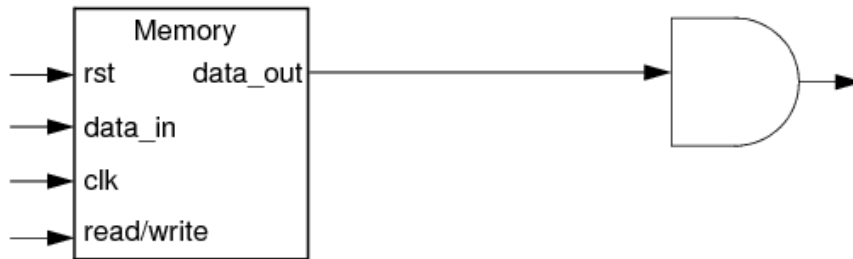
Short Message: *Output '%h' of memory cell '%m' is not by-passed in shift mode.*

Severity	Error
Description	Output of memory cells are unpredictable by ATPG. The output of memory cell is x. This reduces the test coverage. The output of memory cells must be bypassed to prevent the propagation of x inside the design during the scan capture mode.
Associated parameter	<pre>params MEM_SM_MCMB {output_bypass_mode="strict" "relax"}</pre> <p>Controls the behavior of MEM_SM_MCMB check.</p> <p>Setting this parameter to <code>strict</code> causes Superlint to flag the MEM_SM_MCMB check in situations where the output of memory cells is not bypassed using a multiplexer before being used.</p> <p>By default, the value of this parameter is considered strict.</p> <p>Setting this parameter to <code>relax</code> causes Superlint not to flag the MEM_SM_MCMB check if the output of the memory cell is used through a unary operator before being bypassed.</p>

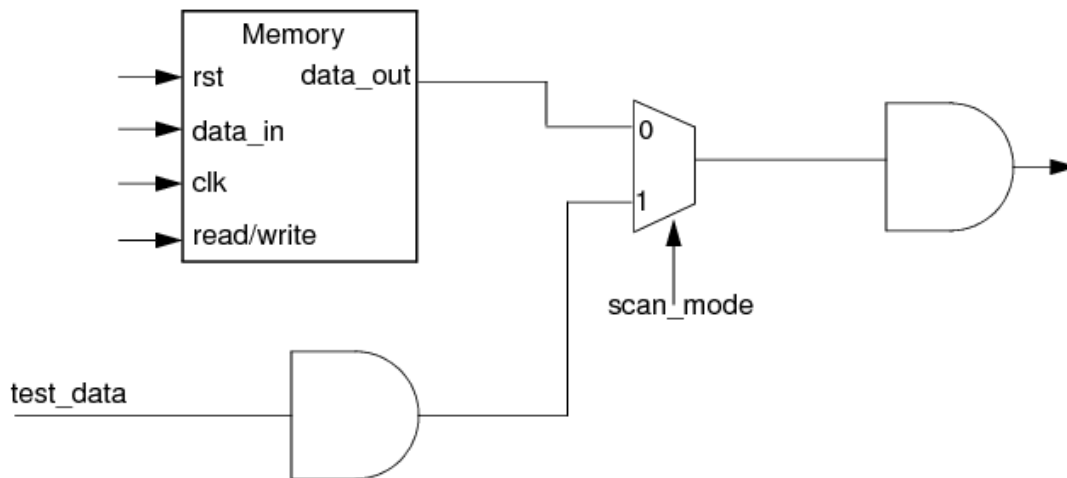
The following diagram illustrates this problem.

JasperGold Superlint Checks Reference

DFT Checks



In the above diagram, the output of the memory cell is feeding the logic. The output of a memory cell should be bypassed in the scan capture mode. Consider the following schematic, which rectifies the problem.



[Back to Top](#)

JasperGold Superlint Checks Reference

DFT Checks

RST_NC_CTCL

Short Message: *In shift mode, signal '%h' driving reset of flip-flop '%h' is neither disabled nor controllable from top.*

Severity	Error
Description	Asynchronous set/reset of flip-flops should either be inactive during scan shift or should be driven from primary port, which can be forced inactive during scan shift. If the set/reset is active, the register cannot be part of the scan chain, which can lead to issues during scan insertion.

The following code illustrates the occurrence of RST_NC_CTCL.

```
module test(clk,rst,var_a,port_d,scan_shift);
input  clk,rst,port_d,scan_shift;
output var_a;
reg var_a;
assign rst = (scan_shift) ? 1'b0 : clk;
always@(posedge clk or negedge rst)
begin
    if (!rst)
        var_a <= 1'b0;
    else
        var_a <= port_d;
end
endmodule
```

In the above code, `rst` is an asynchronous reset but it has the value `1'b0` during the scan shift mode. The reset of the above flip-flop is active when `rst` is low, which may lead to issues during scan insertion. To avoid this violation, control reset using `dftrstdisable` signal in combination with the original reset signal to ensure that reset can be disabled during scan shift mode as shown below:

```
assign rst_n = rst | dftrstdisable;
```

With this change, `rst_n` is reset to the flop instead of `rst`.

[Back to Top](#)

JasperGold Superlint Checks Reference

DFT Checks

RST_NR_CKSC

Short Message: *In capture mode, clock pin '%h' of flip-flop '%h' is driven by a reset signal.*

Severity	Warning
Description	If a reset is present in the path of the clock, it can lead to testability issues. This condition should be avoided.

The following code illustrates the occurrence of RST_NR_CKSC.

```
module mod_a (clk_a, rst, in_a, out_a);
  input clk_a, rst, in_a;
  output reg out_a;

  assign clk_a = rst;

  always @(posedge clk_a or negedge rst)
    if (rst == 1'b0)
      out_a = 1'b0;
    else
      out_a = in_a;

endmodule
```

Reset `rst` is present in the path of clock `clk_a`. This should be modified.

[Back to Top](#)

JasperGold Superlint Checks Reference

DFT Checks

RST_NR_NCTL

Short Message: *Signal '%h' driving reset of flip-flop '%h' has same second level disable signal.*

Severity	Warning
Description	<p>The tool issues this check when a reset disable signal for a flip-flop is disabling the flip-flop reset as well as the reset generator for this flip flop.</p> <p>For example, consider a flip-flop F1 and its reset signal R1 where R1 is driven by another flip-flop F2 with reset R2. If reset disable signals for R1 and R2 are both the same, the tool issues this check.</p>

The following code illustrates the occurrence of RST_NR_NCTL.

```
module test(rst_f,rst_b,dft_rst_disable,clk,in,rst_e,reg_b);
input clk,rst_f,rst_b,dft_rst_disable,in;
output reg_b,rst_e;
reg reg_b,rst_e;
assign rst_f = rst_b | dft_rst_disable;
always @(posedge clk or negedge rst_f)
    if(!rst_f)
        rst_e = 1'b0;
    else
        rst_e = in;
assign rst_b = rst_e | dft_rst_disable;
always @(posedge clk or negedge rst_b)
begin
    if(!rst_b)
        reg_b = 1'b0;
    else
        reg_b = in;
end
endmodule
```

In the above example, `rst_b`, which is the reset of `reg_b`, is disabled by `dft_rst_disable` and is driven by `rst_e`, and `rst_e` is driven by a flop with reset `rst_f`, which is again disabled by `dft_rst_disable`. The reset disable signal should be driven directly from a primary input.

JasperGold Superlint Checks Reference

DFT Checks

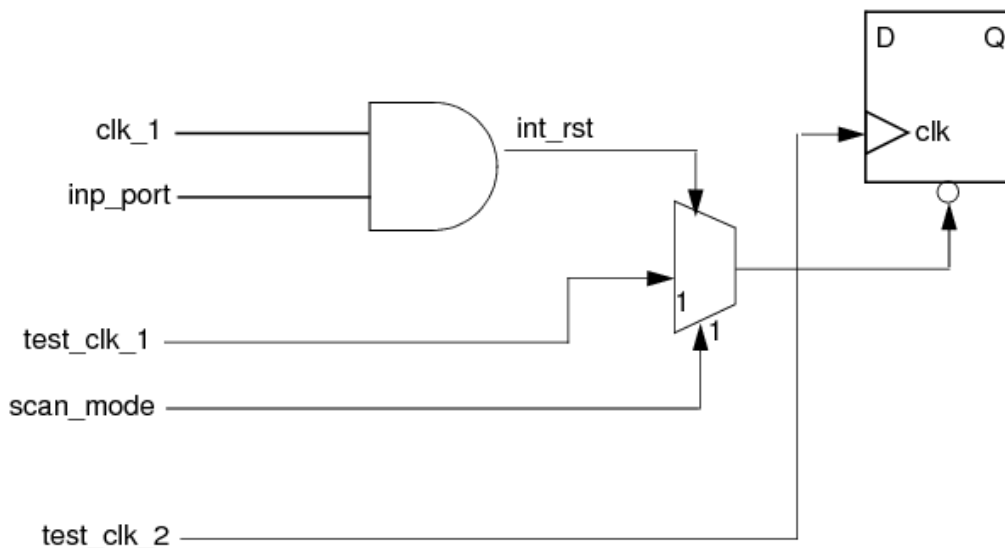
[Back to Top](#)

RST_NR_TCLK

Short Message: *Test clock '%h' drives asynchronous set/reset '%h' of flip-flop/latch '%h'.*

Severity	Error
Description	This error occurs if an asynchronous set/reset of a flip-flop/latch is driven by test clock in the scan capture mode. This reduces the test coverage.

The following diagram illustrates this problem.



In scan capture mode, testclock `test_clk_1` is connected to the reset pin of the flip flop through a mux.

[Back to Top](#)

JasperGold Superlint Checks Reference

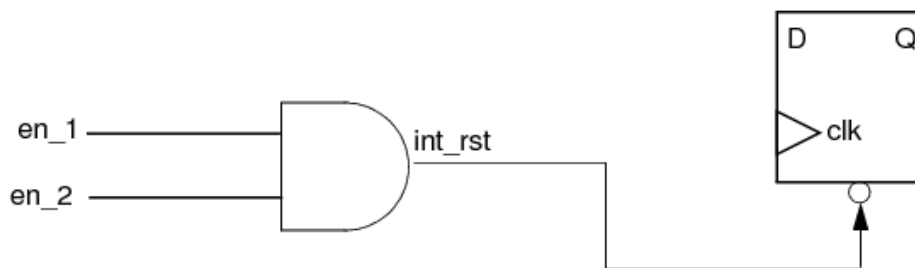
DFT Checks

RST_TM_NCSC

Short Message: *In capture mode, asynchronous set/reset '%s' of flip-flop '%s' is not controllable.*

Severity	Error
Description	In scan capture mode, the asynchronous set/reset pin should be directly controllable by a primary set/reset port. This primary set/reset port can be activated during scan capture. If an asynchronous set/reset pin of a register cannot be activated during scan capture, the stuck-at fault of the inactive value cannot be tested. As a result, the test coverage will be reduced.
Parameter associated	<code>controllable_thru_comb_logic=no</code> <code>yes</code> The default value for this parameter is <code>no</code> . When the value of this parameter is set to <code>no</code> , Superlint does not consider asynchronous set/reset as controllable if, in scan capture mode, asynchronous set/reset is controllable by primary inputs through some combinational logic. Setting a value of <code>yes</code> causes Superlint to consider the asynchronous set/reset as controllable.

The following diagram illustrates this problem.

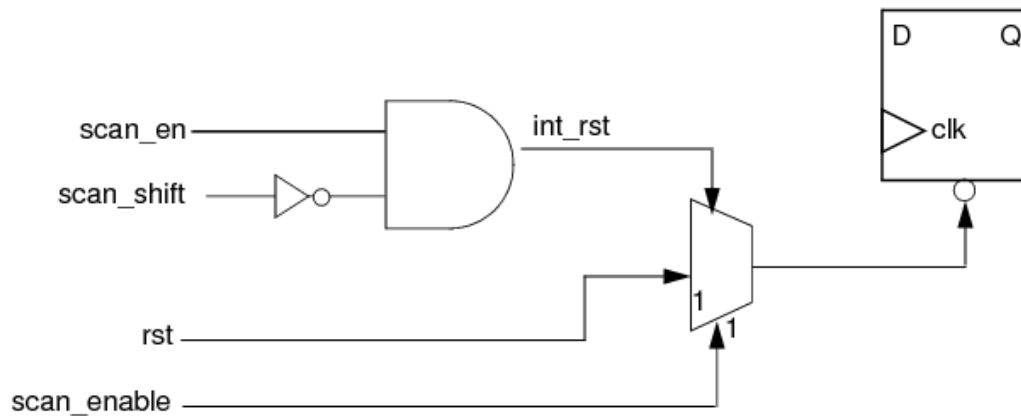


In the above diagram, asynchronous reset is either active or not controllable during scan capture mode. As a result, there can be issues during scan insertion.

The following schematic illustrates a fix for this problem.

JasperGold Superlint Checks Reference

DFT Checks



In the above diagram, the reset pin `int_rst` is bypassed in scan capture mode. The asynchronous set/reset pin of the flip flop is controlled by the primary input reset pin `rst`.

[Back to Top](#)

SCN_IS_VRFD

Short Message: *In shift mode, constraint '%s' for block '%s' has the same value that was used during block verification.*

Severity	Info
Description	A constraint specified for a black-boxed module has been verified by using design constraints.

The following code illustrates the occurrence of SCN_IS_VRFD:

```
module top(port_a);
  input port_a;
  wire sign_a
  assign sig_a = port_a;
  bot b1(.port_a(sig_a));
endmodule
```

Required DFT constraints in input Tcl file:

JasperGold Superlint Checks Reference

DFT Checks

```
config_rtltds -signal -constant {{port_a 1}} -mode shift
```

In the above example, `b1.port_a` is assigned the value 1 using design constraints. If the `bot` block was verified using the same value in shift mode and then the database was saved using the `check_superlint -save` command, the tool issues `SCN_IS_VRFD`.

Note: In the above example, this check would be issued only when module `bot` is black boxed and its database is loaded using the `check_superlint -import` command.

[Back to Top](#)

SCN_NO_VRFD

Short Message: *In shift mode, constraint '%s' for block '%s' either has a different value than used during block verification or is not constant'.*

Severity	Error
Description	A constraint specified for a black-boxed module has not been verified using design constraints.

The following code illustrates the occurrence of `SCN_NO_VRFD`:

```
module top(port_a);
    input port_a;
    wire sig_a
    assign sig_a = port_a;
    bot b1(.port_a(sig_a));
endmodule

Required DFT constraints in input Tcl file:
config_rtltds -signal -constant {{port_a 1}} -mode shift
```

In the above example, `b1.port_a` is assigned the value 1 using design constraints. If the `bot` block was verified using the value 0 and then the database was saved using the `check_superlint -save` command, the tool issues `SCN_NO_VRFD` in shift mode.

Note: In the above example, this check would be issued only when module `bot` is black boxed and its database is loaded using the `check_superlint -import` command.

[Back to Top](#)

JasperGold Superlint Checks Reference

DFT Checks

SIG_CM_DNCF

Short Message: *In capture mode, constraint at signal '%s' is driven by flip-flop with non-constant data.*

Severity	Warning
Description	A constraint applied at a signal might not be stable if it is driven by a flip-flop whose data pin is not constant. At the next clock cycle, the constraint might change depending on the value of the data.

The following example illustrates the occurrence of SIG_CM_DNCF:

```
module top (clk, rst, in, out, out2, dft);
  input in, clk, rst, dft;
  output reg[1:0] out, out2;
  wire set;
  reg wire1, wire2;
  wire wire3, wire4;

  assign set = rst & in;
  assign wire3 = dft & in;
  assign wire4 = dft | in;

  flop f1(clk, rst, set, wire3, wire1);
  flop f2(clk, rst, set, wire4, wire2);

  assign out = {wire1, wire2};
  assign out2 = {wire2, wire1};

endmodule

module flop (clk, rst, set, in, out);
  input in, clk, rst, set;
  output out;
  reg out;

  always @(posedge clk or negedge rst or negedge set)
  begin
    if (!set)
```

JasperGold Superlint Checks Reference

DFT Checks

```
        out = 1'b0;
    else if(!rst)
        out = 1'b1;
    else
        out = in;
    end

endmodule

Required DFT constraints in input Tcl file
config_rtltds -signal -constant {{dft 0} {out2 2'b11} {out 2'b11}} -mode capture
```

In the above example, constraint applied at `out_a` will not be stable until `in_a` is not constant. To avoid this violation, remodel the design.

[Back to Top](#)

SIG_CT_PROP

Short Message: *In capture mode, instance '%s' is stopping propagation of constraint '%s'.*

Severity	Information
Description	Combinational logic present in an instance has stopped propagation of the constraint. This could be intended or it could be a bug in the design.

The following example illustrates the occurrence of SIG_CT_PROP.

```
module test (reset, test_clk_a, enable, port_a, port_b);
    input reset, enable, port_b, test_clk_a;
    output port_a;
    reg port_a;
    wire reset_a;

    assign reset_a = reset | enable;

    always@(posedge test_clk_a or negedge reset_a)
    begin
        if(!reset_a)
            port_a <= 1'b0;
    end
endmodule
```

JasperGold Superlint Checks Reference

DFT Checks

```
else
port_a <= port_b;
end
endmodule

Required DFT constraints in Tcl file:
config_rtltds -signal -constant {{enable 0}} -mode capture
```

In the above example, `reset_a` is stopping propagation of constraint at `enable`. Verify that this is intended or remodel the design.

[Back to Top](#)

SIG_NR_ICGR

Short Message: *In capture mode, reset '%s' is present in the path of enable of clock gate '%s'.*

Severity	Warning
Description	If a reset is present in the path of the clock gate enable, it can lead to testability issues. This condition should be avoided.

The following example illustrates the occurrence of SIG_NR_ICGR.

```
module test(clk,rst,var_a,port_d);
input clk,rst,port_d;
output var_a;
reg var_a;
reg en,latch_out,clk_out;

assign en = rst;
always @en
if(!clk)
latch_out = en;
assign clk_out = clk & latch_out;
always @(posedge clk_out or negedge rst)
begin
if (!rst)
var_a <= 1'b0;
else
var_a <= port_d;
```


JasperGold Superlint Checks Reference

DFT Checks

```
end
endmodule
```

Reset `rst` is present in the path of clock gate enable `en`. This should be modified.

[Back to Top](#)

SIG_SM_DNCF

Short Message: *In shift mode, constraint at signal '%s' is driven by flip-flop with non-constant data.*

Severity	Warning
Description	A constraint applied at a signal might not be stable if it is driven by a flip-flop whose data pin is not constant. At the next clock cycle, the constraint might change depending on the value of the data.

The following example illustrates the occurrence of SIG_SM_DNCF:

```
module top (clk, rst, in, out, out2, dft);
    input in, clk, rst, dft;
    output reg[1:0] out, out2;
    wire set;
    reg wire1, wire2;
    wire wire3, wire4;

    assign set = rst & in;
    assign wire3 = dft & in;
    assign wire4 = dft | in;

    flop f1(clk, rst, set, wire3, wire1);
    flop f2(clk, rst, set, wire4, wire2);

    assign out = {wire1, wire2};
    assign out2 = {wire2, wire1};

endmodule

module flop (clk, rst, set, in, out);
    input in, clk, rst, set;
```

JasperGold Superlint Checks Reference

DFT Checks

```
output out;
reg out;

always @(posedge clk or negedge rst or negedge set)
begin
    if (!set)
        out = 1'b0;
    else if(!rst)
        out = 1'b1;
    else
        out = in;
end

endmodule

Required DFT constraints in input Tcl file
config_rtltds -signal -constant {{dft 0} {out2 2'b11} {out 2'b11}} -mode shift
```

In the above example, constraint applied at `out_a` will not be stable until `in_a` is not constant. To avoid this violation, remodel the design.

[Back to Top](#)

SIG_ST_PROP

Short Message: *In shift mode, instance '%s' is stopping propagation of constraint '%s'.*

Severity	Information
Description	Combinational logic present in an instance has stopped propagation of the constraint. This could be intended or it could be a bug in the design.

The following example illustrates the occurrence of SIG_ST_PROP.

```
module test (reset, test_clk_a, enable, port_a, port_b);
input reset, enable, port_b, test_clk_a;
output port_a;
reg port_a;
wire reset_a;
```

JasperGold Superlint Checks Reference

DFT Checks

```
assign reset_a = reset | enable;

always@(posedge test_clk_a or negedge reset_a)
begin
  if(!reset_a)
    port_a <= 1'b0;
  else
    port_a <= port_b;
  end
endmodule

Required DFT constraints in Tcl file:
config_rtltds -signal -constant {{enable 0}} -mode shift
```

In the above example, `reset_a` is stopping propagation of constraint at `enable`. Verify that this is intended or remodel the design.

[Back to Top](#)

TRI_NC_ENSS

Short Message: *In shift mode, enable of tristate buffer '%s' is not controllable.*

Severity	Error
Description	It is necessary to control the condition of a tristate enable during the scan shift operation to avoid conflict on nets or buses.

The following code illustrates this problem.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity ent_a is
  port ( clk :in std_logic;
         rst :in std_logic;
         port_a :in std_logic;
         port_b :in std_logic;
         port_c :out std_logic;
         port_d :out std_logic);
end ent_a;
```

JasperGold Superlint Checks Reference

DFT Checks

```
architecture arch_a of ent_a is
signal sig_a, sig_b : std_logic;
begin -- Architecture
proc_a :process(sig_a, port_a)
begin
    if(sig_a = '1') then
        port_c <= port_a;
    else
        port_c <= 'Z';
    end if;
end process proc_a;

proc_b :process(sig_b, port_b)
begin
    if(sig_b = '1') then
        port_d <= port_b;
    else
        port_d <= 'Z';
    end if;
end process proc_b;

end arch_a;
Required DFT constraints in input Tcl file:
config_rtlids -signal -constant {{sig_a 1'b0}} mode -shift
```

In the above code, enables of tristate buffers, `sig_a` and `sig_b` are not controllable from primary inputs. This might lead to conflict on `port_c` and `port_d`, respectively.

[Back to Top](#)

TRI_NO_DZSS

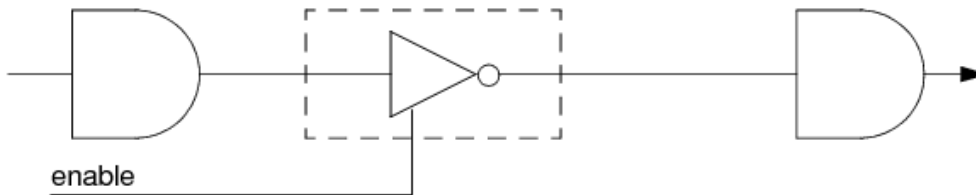
Short Message: *In shift mode, enable of tristate buffer '%h' does not drive 'Z' but is controllable.*

Severity	Error
Description	It is necessary to control the condition of a tristate enable during the scan shift operation to avoid conflict on nets or buses.

JasperGold Superlint Checks Reference

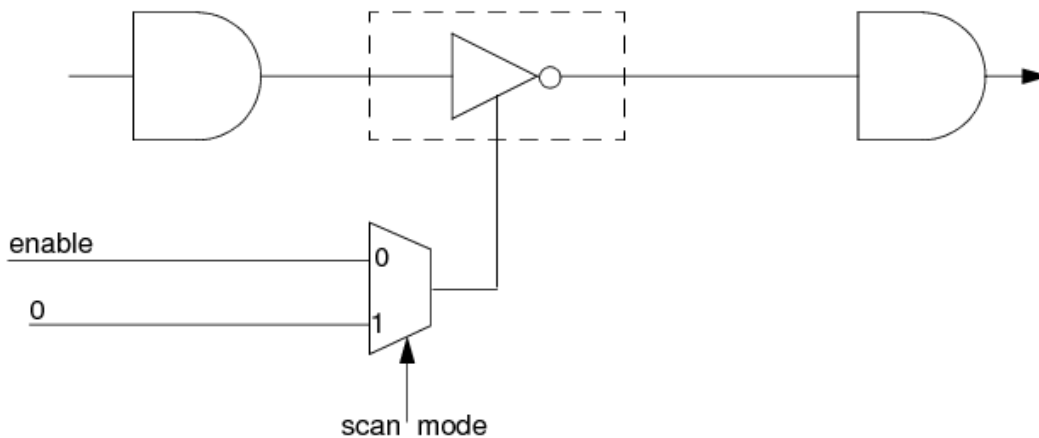
DFT Checks

The following diagram illustrates this problem.



In the above diagram, the enable of tristate buffer is not controlled in the scan capture mode. As a result, tristate buffer will not pass the z value to the output.

Consider the following schematic that rectifies the problem by controlling the enable of tristate buffer. The value z is passed to the output in scan capture mode.



[Back to Top](#)

JasperGold Superlint Checks Reference

DFT Checks

Automatic Formal Checks

This chapter provides descriptions, examples, and syntax to help you understand and implement automatic formal checks with the JasperGold Superlint App. All automatic formal checks fall under the `AUTO_FORMAL` domain in the tool. This domain is divided into functional categories, which are subdivided as needed into tags. For example, the `AUTO_FORMAL_CASE` category under the `AUTO_FORMAL` domain includes three tags as follows: `CAS_NO_PRIO` for priority case, `CAS_NO_UNIQ` for unique case, and `CAS_IS_DFRC` for default case.

This `AUTO_FORMAL` domain includes the following categories:

- `AUTO_FORMAL_CASE` on page 336
- `AUTO_FORMAL_SIGNALS` on page 339
- `AUTO_FORMAL_X_ASSIGNMENT` on page 351
- `AUTO_FORMAL_ARITHMETIC_OVERFLOW` on page 352
- `AUTO_FORMAL_BUS` on page 354
- `AUTO_FORMAL_DEAD_CODE` on page 356
- `AUTO_FORMAL_FSM` on page 357
- `AUTO_FORMAL_OUT_OF_BOUND_INDEXING` on page 367
- `AUTO_FORMAL_COMBO_LOOP` on page 368

AUTO_FORMAL_CASE

Case statements check the RTL for the proper use of unique and priority case statements as well as default case statements. Violations might indicate one of the following:

- RTL can behave differently than gates
- Synthesis can make false assumptions for optimization

The AUTO_FORMAL_CASE category includes the following checks:

- CAS_IS_DFRC on page 336
- CAS_NO_PRIQ on page 337
- CAS_NO_UNIQ on page 338

CAS_IS_DFRC

Short Message: *A reachable default case was found.*

Severity	Error
Description	Default case checks confirm that default case statements cannot be reached. Violations could be the result of an incompletely specified case

The following code illustrates the occurrence of CAS_IS_DFRC.

```
module FSM (clk, rst);
  input clk, rst;
  reg [2:0] state, next;
  wire en;
  always @(posedge clk)
    if (rst)
      state <= 3'b000;
    else
      state <= next;
  always @(state)
  begin
    case(state)
      3'b000:
        next = 3'b001;
      3'b001:
        if (en)
```


JasperGold Superlint Checks Reference

Automatic Formal Checks

```
        next = 3'b010;
    else
        next = 3'b011;
3'b010:
    next = 3'b100;
3'b011:
    next = 3'b101;
default:
    next = 3'bxxx;
endcase
end
endmodule
```

This example represents a default case that is reachable.

An example of the generated property follows:

```
~( ~(state == 3'b11) & ~(state == 3'b10) & ~(state == 3'b1) & ~(state == 3'b0) )
```

[Back to Top](#)

CAS_NO_PRIO

Short Message: *Priority case failure found.*

Severity	Error
Description	All possible case items should be covered in priority case statements.

The following code illustrates the occurrence of CAS_NO_PRIO.

```
module mod_a (port_a, port_b, port_c);
    input [3:0] port_a;
    input [1:0] port_c;
    output port_b;
    reg port_b;
    always @(port_c)
    begin
        priority case(port_c)
            2'b01 : port_b = port_a[1];
            2'b10 : port_b = port_a[2];
            2'b11 : port_b = port_a[3];
        endcase
    end
end
```

JasperGold Superlint Checks Reference

Automatic Formal Checks

```
endmodule
```

In the example above, the case statement specified with the `priority` keyword fails to cover all cases.

An example of the generated property follows:

```
((port_c == 2'b11) | (port_c == 2'b10) | (port_c == 2'b1))
```

Note: This check is also applicable for case statements with `full_case` pragma.

[Back to Top](#)

CAS_NO_UNIQ

Short Message: *Unique case failure found.*

Severity	Error
Description	Unique case statements must have exactly one case item that matches the case expression.

The following code illustrates the occurrence of CAS_NO_UNIQ.

```
module test (clk, out_a);

    input clk;
    output reg [31:0] out_a;

    reg [3:0] reg_a;
    always @(posedge clk)
    begin
        unique case ( reg_a )
            6, 12, 14: out_a = 32'd10012;
            2, 6, 7:   out_a = 32'd10015;
            3, 12, 10: out_a = 32'd50009;
            default:   out_a = 32'd0;
        endcase
    end

endmodule
```

In the example above, the case statement specified with the `unique` keyword has more than one case item that matches the case expression. `reg_a` is used as the case expression in a

unique case statement where two of the case item expressions are 12. Case item expressions should be mutually exclusive.

An example of the generated property follows:

```
1'b0
```

Note: This check is also applicable for case statements with `parallel_case` pragma.

[Back to Top](#)

AUTO_FORMAL_SIGNALS

The AUTO_FORMAL_SIGNALS category includes the following checks:

- [SIG_IS_DLCK](#) on page 341
- [SIG_IS_STCK](#) on page 343
- [SIG_NO_TGFL](#) on page 345
- [SIG_NO_TGRS](#) on page 347
- [SIG_NO_TGST](#) on page 349

By default, Superlint generates signal checks for flops and outputs only and filters signal checks for primary clocks and reset, user-defined clocks and resets, internal clocks and resets, and gated clocks. Also by default, the tool generates toggle checks for scalars only, and wide signals are extracted only for buses that have critical values or are in a sensitivity list of always. You can change the default behavior with the following configuration variables:

- `set_superlint_signals_extraction_heuristic (all | auto)`
 - ☐ Use `all` to extract all wide signals in the design.
 - ☐ Use `auto` to extract buses that have critical values or are in a sensitivity list of always. This is the default.
- `set_superlint_signals_filter_wide_signal (on | off | auto)`
 - ☐ Use `on` to filter all signal checks for wide signals and extract scalars only.
 - ☐ Use `off` to consider wide signals during extraction.
 - ☐ Use `auto` to extract wide signals for `SIG_AT_STCK` checks only. This is the default.

JasperGold Superlint Checks Reference

Automatic Formal Checks

- `set_superlint_signals_type { all | wire | flop | latch | input
| output }`

Enables signal extraction for the specified signal types.

SIG_IS_DLCK

Short Message: *The signal '%s' is a deadlock signal.*

Severity	Error
Description	<p>Signals deadlock checks confirm that all signals can be reached. For each possible value of the signal ($2^{(\text{width})}$), the tool generates a property associated with the deadlock signal's scope.</p> <p>You can tune signals deadlock checks as follows:</p> <ul style="list-style-type: none">■ <code>set_superlint_signals_ctl_deadlock</code> sets the mode of the signals deadlock (<code>true=CTL</code> or <code>false=LTL</code>). The default is <code>false</code>.■ <code>set_superlint_signals_deadlock_bitblasted true</code> enables the bit blasting of deadlock signals properties and is the default. Set this variable to <code>false</code> if you want to disable bit blasting. <p>Note: The number of bits checked depends on the setting of the variable <code>superlint_signals_max_bits</code>.</p> ■ <code>set_superlint_signals_deadlock_max_width <N></code> specifies the maximum width for which the tool generates signal deadlock checks when <code>set_superlint_signals_deadlock_bitblasted</code> is <code>false</code>. The default is 10. <p>Note: This variable enables the extraction of deadlock checks on all signals in the design smaller than the maximum width. The tool generates one property for each possible value.</p> ■ <code>set_superlint_signals_max_bits <N></code> limits the number of bits on which the tool generates signals deadlock properties when <code>superlint_signals_deadlock_bitblasted</code> is set to <code>true</code>. The default is 1. Use a non-positive value to unlimit max bits, but exercise caution as unlimiting bits can generate a large number of properties and affect performance. <p>Note: The tool attempts to divide a range equally. For example, for an 8bit vector, if <code>superlint_signals_max_bits</code> is set to 3, the tool might generate checks for bits 0, 3, and 7. If however, <code>superlint_signals_max_bits</code> is set to 1, then the tool chooses the first bit.</p>

JasperGold Superlint Checks Reference

Automatic Formal Checks

The following code illustrates the occurrence of SIG_IS_DLCK.

```
module FSM (din, a_rst, clk, z_o);
    input  din, a_rst, clk;
    output z_o;
    reg z_o;
    parameter [1:0] s0=0, s1=1, s2=2;
    reg [1:0] ps, ns;
    // sequential block
    always @ (posedge clk or posedge a_rst)
    begin: seq_block
        if (a_rst)
            ps = s0;
        else
            ps = ns;
    end

    // combinational block
    always @ (ps or din)
    begin: comb_block
        ns = s0;
        z_o = 1'b1;
        case (ps)
            s0: begin
                z_o = 1'b0;
            end
            s1: begin
                if (din == 1'b0)
                    ns = s0;
                else
                    ns = s2;
            end
            s2: begin
                if (din == 1'b1)
                    ns = s1;
                else
                    ns = s0;
            end
        endcase
    end
endmodule
```

In the above example, after reaching `s0`, the design remains in that state; thus, a deadlock has been reached, and the tool reports a violation.

An example of the generated property follows:

```
(ps == s0 | => ##[0:$] ps != s0)
```

[Back to Top](#)

SIG_IS_STCK

Short Message: *The signal '%s' is stuck-at %s.*

Severity	Error
Description	<p>Stuck-at signal checks confirm that RTL logic elements are never stuck at a constant value. By default, the Superlint App filters signals connected through hierarchy or direct assignment from the list of violations and reports a violation for the primary signal only. If you prefer not to filter out these connected signals, set the <code>superlint_signals_filter_connected</code> variable to <code>false</code>. With this variable set to <code>false</code>, connected signals are reported as violations and grouped with their primary signal.</p> <p>You can tune stuck-at checks as follows:</p> <ul style="list-style-type: none">■ <code>set_superlint_stuck_at_bitblasted false</code> prevents the bit blasting of stuck-at properties. This is the default.■ <code>set_superlint_signals_max_bits <N></code> limits the number of bits on which the tool generates stuck-at or toggle properties when the <code>superlint_stuck_at_bitblasted</code> variable is set to <code>true</code>. The default is 1. Use a non-positive value to unlimit max bits, but exercise caution as unlimiting bits can generate a large number of properties and affect performance. <p>Note: The tool attempts to divide a range equally. For example, for an 8bit vector, if <code>superlint_signals_max_bits</code> is set to 3, the tool might generate checks for bits 0, 3, and 7. If however, <code>superlint_signals_max_bits</code> is set to 1, then the tool chooses the first bit.</p> <ul style="list-style-type: none">■ <code>set_superlint_signals_type {all wire flop latch input output}+</code> enables signal extraction for the specified signal types only. By default, the tool generates properties for flops and outputs only.

The following code illustrates the occurrence of SIG_IS_STCK.

```
module mod_a(clk, rst, port_a, port_b);
    input clk, rst, port_a;
    output port_b;
    reg port_b;
```

JasperGold Superlint Checks Reference

Automatic Formal Checks

```
always @(posedge clk or negedge rst)
begin
if (!rst)
port_b = 1'b1;
else
port_b = port_a | 1'b1;
end
endmodule
```

The above example illustrates a `stuckat_signals` violation in which `port_b` is always stuck at `1'b1`

An example of the generated property follows:

```
##1 (!$stable(port_b))
```

Note: For some signals, the value cannot be identified. In those cases, use the following command to see the value:

```
visualize -new_window; visualize -violation -property $propName -bg
```

[Back to Top](#)

SIG_NO_TGFL

Short Message: *The signal '%s' has not toggled from 1 to 0.*

Severity	Error
Description	<p>Toggle checks confirm that RTL logic elements have toggled at least once. Toggle fall checks confirm that signals have toggled from 1 to 0.</p> <p>You can tune toggle checks as follows:</p> <ul style="list-style-type: none">■ <code>set_superlint_signals_max_bits <N></code> limits the number of bits on which the tool generates stuck-at or toggle properties when the <code>superlint_stuck_at_bitblasted</code> variable is set to true. The default is 1. Use a non-positive value to unlimit max bits, but exercise caution as unlimiting bits can generate a large number of properties and affect performance. <p>Note: The tool attempts to divide a range equally. For example, for an 8bit vector, if <code>superlint_signals_max_bits</code> is set to 3, the tool might generate checks for bits 0, 3, and 7. If however, <code>superlint_signals_max_bits</code> is set to 1, then the tool chooses the first bit.</p> <ul style="list-style-type: none">■ <code>set_superlint_signals_type {all wire flop latch input output}+</code> enables signal extraction for the specified signal types only. By default, the tool generates properties for flops and outputs only.

The following code illustrates the occurrence of SIG_NO_TGFL.

```
module top (clk1,reset,out);
  input clk1,reset;
  reg [7:0] count1;
  output [7:0] out;
  always @(posedge clk1)begin
    if(!reset)begin
      count1 <= 8'b00000000;
    end
    else begin
      if (count1 == 8'b11111110)
        begin
          count1 <= 8'b00000000;
        end
      else begin
```

JasperGold Superlint Checks Reference

Automatic Formal Checks

```
        count1 = count1+ 8'b000000010;
    end
end
end
assign out = count1;
endmodule
```

In the example above, the tool reports a violation for `count1` because the signal does not toggle from 1 to 0 at any point.

Examples of generated properties follow:

```
##1 $fell(out[0])
##1 $fell(out[3])
```

[Back to Top](#)

SIG_NO_TGRS

Short Message: *The signal '%s' has not toggled from 0 to 1.*

Severity	Error
Description	<p>Toggle checks confirm that RTL logic elements have toggled at least once. Toggle rise checks confirm that signals have toggled from 0 to 1.</p> <p>You can tune toggle checks as follows:</p> <ul style="list-style-type: none">■ <code>set_superlint_signals_max_bits <N></code> limits the number of bits on which the tool generates stuck-at or toggle properties when the <code>superlint_stuck_at_bitblasted</code> variable is set to true. The default is 1. Use a non-positive value to unlimit max bits, but exercise caution as unlimiting bits can generate a large number of properties and affect performance. <p>Note: The tool attempts to divide a range equally. For example, for an 8bit vector, if <code>superlint_signals_max_bits</code> is set to 3, the tool might generate checks for bits 0, 3, and 7. If however, <code>superlint_signals_max_bits</code> is set to 1, then the tool chooses the first bit.</p> <ul style="list-style-type: none">■ <code>set_superlint_signals_type {all wire flop latch input output}+</code> enables signal extraction for the specified signal types only. By default, the tool generates properties for flops and outputs only.

The following code illustrates the occurrence of SIG_NO_TGRS.

```
module top (clk1,reset,out);
  input clk1,reset;
  reg [7:0] count1;
  output [7:0] out;
  always @(posedge clk1)begin
    if(!reset)begin
      count1 <= 8'b00000000;
    end
    else begin
      if (count1 == 8'b11111110)
        begin
          count1 <= 8'b00000000;
        end
      else begin
```

JasperGold Superlint Checks Reference

Automatic Formal Checks

```
        count1 = count1+ 8'b000000010;
    end
end
end
assign out = count1;
endmodule
```

In the example above, the tool reports a violation for `count1` because the signal does not toggle from 0 to 1 at any point.

Examples of generated properties follow:

```
##1 $rose(out[0])
##1 $rose(out[3])
##1 $rose(out[7])
##1 $rose(count1[0])
##1 $rose(count1[3])
##1 $rose(count1[7])
```

[Back to Top](#)

SIG_NO_TGST

Short Message: *The signal '%s' has not toggled stable.*

Severity	Error
Description	<p>Toggle checks confirm that RTL logic elements have toggled at least once. Toggle stable checks confirm that signals have toggled back to a stable state.</p> <p>You can tune toggle checks as follows:</p> <ul style="list-style-type: none">■ <code>set_superlint_signals_max_bits <N></code> limits the number of bits on which the tool generates stuck-at or toggle properties when the <code>superlint_stuck_at_bitblasted</code> variable is set to <code>true</code>. The default is 1. Use a non-positive value to unlimit max bits, but exercise caution as unlimiting bits can generate a large number of properties and affect performance. <p>Note: The tool attempts to divide a range equally. For example, for an 8bit vector, if <code>superlint_signals_max_bits</code> is set to 3, the tool might generate checks for bits 0, 3, and 7. If however, <code>superlint_signals_max_bits</code> is set to 1, then the tool chooses the first bit.</p> <ul style="list-style-type: none">■ <code>set_superlint_signals_type {all wire flop latch input output}+</code> enables signal extraction for the specified signal types only. By default, the tool generates properties for flops and outputs only.

The following code illustrates the occurrence of SIG_NO_TGST.

```
module top (clk1,reset,out)
  input clk1,reset;
  reg [7:0] count1;
  output [7:0] out;
  always @(posedge clk1)begin
    if(!reset)begin
      count1 <= 8'b00000000;
    end
    else begin
      if (count1 == 8'b11111110)
        begin
          count1 <= 8'b00000000;
        end
    end
  end
end
```

JasperGold Superlint Checks Reference

Automatic Formal Checks

```
        else begin
            count1 = count1+ 8'b000000010;
        end
    end
end
end
assign out = count1;
endmodule
```

In the above example, the tool reports a violation for `count1` because it never becomes stable after toggling from its initial state.

Examples of generated properties follow:

```
##1 $stable(out[0])
##1 $stable(out[3])
##1 $stable(out[7])
##1 $stable(count1[0])
##1 $stable(count1[3])
##1 $stable(count1[7])
```

[Back to Top](#)

AUTO_FORMAL_X_ASSIGNMENT

The AUTO_FORMAL_X_ASSIGNMENT category includes the ASG_IS_XRCH check only.

ASG_IS_XRCH

Short Message: *A reachable x-assignment was found.*

Severity	Error
Description	<p>X-assignment checks confirm that <code>x</code> variables cannot be reached. If an x-assignment is reachable, it becomes an active source of <code>x</code>, and can lead to unexpected functionality.</p> <p>Note: <code>x</code> equals 1 or 0 in synthesis and is unknown in simulation.</p> <p>To reduce noise that can be associated with X-assignment checks, use the following command:</p> <pre>set_superlint_enable_observability_tags {ASG_IS_XRCH}</pre>

The following code illustrates the occurrence of ASG_IS_XRCH.

```
module test(port_d,port_a,port_b,port_c,port_e);
input port_a,port_b,port_c;
input [1:0] port_d;
output port_e;
reg port_e;
always @(port_a or port_b or port_c or port_d)
begin
casez(port_d)
2'b00: port_e = port_a;
2'b01: port_e = port_b;
2'b10: port_e = port_c;
2'b11: port_e = 1'bx;
endcase
end
endmodule
```

An example of the generated property follows:

JasperGold Superlint Checks Reference

Automatic Formal Checks

```
~(~({1'b0,port_d} == 2'b0) & ~({1'b0, port_d} == 2'b1) & ~({1'b0, port_d} == 2'b10) & 1'b{x})
```

AUTO_FORMAL_ARITHMETIC_OVERFLOW

The AUTO_FORMAL_ARITHMETIC_OVERFLOW category includes the EXP_IS_OVFL check only.

EXP_IS_OVFL

Short Message: *Arithmetic overflow failure found.*

Severity	Error
Description	<p>Arithmetic overflow checks confirm that the RTL contains no errors that could cause the array to overflow. For example, in the following code snippet, the value of <code>out</code> must never fold to <code>4'b0000</code> when you add <code>1'b1</code>:</p> <pre>logic [3:0] d_array; out = d_array + 1'b1;</pre> <p>A violation indicates an RTL error, which can cause the array to overflow.</p> <p>By default, Superlint checks for both statement and expression arithmetic overflow as follows:</p> <ul style="list-style-type: none">■ <code>statement</code> generates one arithmetic overflow for the assignment that contains the arithmetic operation. This checks overflow condition only on the top level of the assignment of the value expression assigned to the L.H.S variable.■ <code>expression</code> generates a check for every arithmetic operation within the expression in the assignment statement. Supported operators include <code>+</code> and <code>-</code>.

JasperGold Superlint Checks Reference

Automatic Formal Checks

Description (Continued)

You can tune the arithmetic overflow checks as follows:

- To enable arithmetic overflow checks to consider both the signed and unsigned versions of the arithmetic operation, use `set_superlint_arithmetic_overflow_both_signs true`.

Note: This command must be used before elaboration.

- To enable filtering arithmetic overflow of non-critical counters checks, use `set_superlint_arithmetic_overflow_counter_filter true`.

- To enable the check of multiplication operations for expression overflow, use `set_superlint_arithmetic_overflow_enable_multiplication true`.

Note: This is applicable only to the JasperGold Apps front end.

- If you prefer to check only statement or only expression overflow, use `set_superlint_arithmetic_overflow_style (statement | expression)`. By default, the tool generates both statement and expression overflow checks.

Note: `statement` is not supported on the Xcelium™ front end.

- To reduce noise that can be associated with arithmetic overflow checks, use `set_superlint_enable_observability_tags {EXP_IS_OVFL}`.

The following code illustrates the occurrence of `EXP_IS_OVFL`.

```
module mod_a();
    reg [1:0] reg_a;
    reg [1:0] reg_b;
    reg [1:0] reg_c;
    wire [1:0] wir_a;
    wire [1:0] wir_b;
    wire [1:0] wir_c;
    assign wir_a = reg_a + 2'd2;
    assign wir_b = reg_b + 2'b11;
    assign wir_c = reg_c - 2'b11;
endmodule
```

The above example illustrates an arithmetic overflow violation. In this example, 2-bit wires, `wir_a`, `wir_b`, and `wir_c` are being assigned a value that can be greater than the maximum value. Remodel the design to avoid this violation.

AUTO_FORMAL_BUS

The AUTO_FORMAL_BUS category includes the following checks: bus checks:

- [BUS_IS_CONT](#) on page 354
- [BUS_IS_FLOT](#) on page 355

BUS_IS_CONT

Short Message: *Contention bus failure found in bus '%s'.*

Severity	Error
Description	<p>Contention bus checks return a CEX if there is more than one active driver for the bus. The tool checks both values and conditions.</p> <p>To reduce noise that can be associated with contention bus checks, use the following command:</p> <pre>set_superlint_enable_observability_tags {BUS_IS_CONT}</pre>

The following code illustrates the occurrence of BUS_IS_CONT.

```
module test(var_a,var_b);  
  input var_a,var_b;  
  wire sig_a;  
  assign sig_a = var_a;  
  assign sig_a = var_b;  
endmodule
```

In the above example `sig_a` is multiply driven.

An example of the generated property follows:

```
!(var_a != var_b)
```

JasperGold Superlint Checks Reference

Automatic Formal Checks

[Back to Top](#)

BUS_IS_FLOT

Short Message: *Floating bus failure found in bus '%s'.*

Severity	Error
Description	<p>Floating bus checks confirm that there is always at least one active driver for the bus. The tool checks both values and conditions.</p> <p>To reduce noise that can be associated with floating bus checks, use the following command:</p> <pre>set_superlint_enable_observability_tags {BUS_IS_FLOT}</pre>

The following code illustrates the occurrence of BUS_IS_FLOT.

```
module test(clk,rst,in,in2,out);
    input clk;
    input rst;
    input in;
    input in2;
    output out;
    reg out;
    wire w;
    wire w1;
    wire [3:0]w2;
    logic a;
    logic b;
    always @(posedge clk or posedge rst)
    begin
        if(rst)
            out <= w;
        else
            out <= w1&w2;
        end
    assign w = in ? 1'b0 : in2;
    assign w1 = a ? 1'b1 : 1'bz;
    assign w2 = b ? 1'b1 : 3'bz;
endmodule
```

The above example illustrates a scenario where the tool detects a bus without a driver (that is, the tool reports a floating bus for w2 and w1).

Examples of the generated properties follow:

```
!(!a)
!(!b)
```

[Back to Top](#)

AUTO_FORMAL_DEAD_CODE

The AUTO_FORMAL_DEAD_CODE category includes the BLK_NO_RCHB check only.

BLK_NO_RCHB

Short Message: *The block is not reachable.*

Severity	Error
Description	Dead code checks confirm that the RTL is reachable.

The following code illustrates the occurrence of BLK_NO_RCHB.

```
module blkif (out, a, b);
  input a, b;
  output out;
  reg out;
  wire sel;
  assign sel = 1'b1;

  always @(sel or a or b)
  begin
    if (sel)
      out <= a;
    else
      out <= b;
  end
endmodule
```

In the example above, the tool reports a violation since `sel` has a constant value and, as a result, one of the branches of the `if` block is not reachable.

Dead code checks are generated for the following constructs:

- if, if-else statement
- functions
- loop (block property generated for a for/while looping)
- statement (not supported on JasperGold FE)
- case_default (block property generated for the default branch of a case)
- case (block property generated for one of the non-default)
- branches of the case statement
- cond_assign (block property generated for ternary operator (?) in Verilog or when-else in VHDL)

AUTO_FORMAL_FSM

The AUTO_FORMAL_FSM category includes the following checks:

- FSM_IS_DLCK on page 358
- FSM_IS_LLCK on page 360
- FSM_NO_MTRN on page 362
- FSM_NO_RCHB on page 364
- FSM_NO_TRRN on page 365

FSM_IS_DLCK

Short Message: *A deadlock situation was found (%d) for a state of the FSM '%s'.*

Severity	Error
Description	<p>FSM deadlock checks confirm that there is an outgoing edge from the current state whose condition can be met.</p> <p>Note: To reduce noise that can be associated with LTL FSM properties, use the following configuration commands.</p> <ul style="list-style-type: none"> ■ <code>set_superlint_add_automatic_task_assumption true</code> – Automatically generates assumptions for tasks that contain livelock/deadlock FSM or deadlock signal LTL properties The default is <code>true</code>. ■ <code>set_superlint_add_automatic_task_assumption_bitblasted true</code> – Enables bit blasting of automatic task assumptions The default is <code>false</code>.

The following code illustrates the occurrence of `FSM_IS_DLCK`.

```
module FSM (din, a_rst, clk, z_o);
    input  din, a_rst, clk;
    output z_o;
    reg z_o;
    parameter [1:0] s0=0, s1=1, s2=2;
    reg [1:0] ps, ns;

    // sequential block
    always @ (posedge clk or posedge a_rst)
    begin: seq_block
        if (a_rst)
            ps = s0;
        else
            ps = ns;
    end

    // combinational block
    always @ (ps or din)
    begin: comb_block
        ns = s0;
        z_o = 1'b1;
        case (ps)
```

JasperGold Superlint Checks Reference

Automatic Formal Checks

```
s0: begin
  z_o = 1'b0;
end
s1: begin
  if (din == 1'b0)
    ns = s0;
  else
    ns = s2;
  end
s2: begin
  if (din == 1'b1)
    ns = s1;
  else
    ns = s0;
  end
endcase
end

endmodule
```

In the above example, after reaching `s0`, the design remains in that state. This represents a deadlock and the tool reports a violation for state `s0`.

Important

The property created for the boldfaced section of the example below, `fsm_state_deadlock_s0_poi_1_prop_4`, will have a different proof status depending on whether the tool is using CTL or LTL modeling. With CTL modeling, this property is proven, but with LTL modeling, running the proof on this property returns a counterexample.

Note:

- ❑ By default, deadlock properties are liveness LTL properties, which require you to add fairness assumptions. LTL modeling reports a deadlock failure when at some reachable state there is one possible way it can get stuck. If the tool is not able to find any such state, it reports the status of the deadlock check as `proven`. The tool reports a failure when a reachable state gets stuck for *one possible* input scenario.
- ❑ If you prefer to generate deadlock properties using CTL modeling, set the `superlint_fsm_ctl_deadlock` variable to `true` before exporting the property. CTL modeling reports a deadlock failure when at some reachable state there is absolutely no way to get out of that state, that is, the tool reports a failure when a reachable state gets stuck for *all possible* input scenarios. If the tool is not able to find any such state, it reports the status of the deadlock check as `proven`.

An example of the generated property follows:

```
(ps == 0 | => ##[0:$] ps != 0)
```

[Back to Top](#)

FSM_IS_LLCK

Short Message: *Livelock condition found (%d) in the FSM '%s'.*

Severity	Error
Description	Livelock condition detected for the FSM. The states of the processes involved in this condition constantly change with regard to one another but do not progress to some other state. To avoid this violation, modify the RTL and rerun the design.

The following code illustrates the occurrence of FSM_IS_LLCK.

```
module FSM (err, ack, req, clk, rst);
    output err;
    output [3:0] ack;
    input [3:0] req;
    input clk, rst;
    reg err;
    reg [3:0] ack;
    reg [2:0] state, next;
    integer i;
    always @(posedge clk)
        if (rst)
            state <= 3'b000;
        else
            state <= next;
    always @(state or req)
    begin
        next = 3'bxxx;
        err = 0;
        ack = 0;
        case(state)
            3'b000: begin
                case(req)
                    4'b0001 : next = 3'b001;
                    default : next = 3'b000;
                endcase
            end
            3'b001: begin
                case(req)
                    4'b0010 : next = 3'b010;
                    default : next = 3'b001;
                endcase
            end
        endcase
    end
end
```


JasperGold Superlint Checks Reference

Automatic Formal Checks

```
        ack[0] = 1;
        ack[1] = 1;
    end
    3'b010: begin
        case(req)
            4'b0100 : next = 3'b011;
        endcase
        ack[1] = 1;
        ack[2] = 1;
    end
    3'b011: begin
        case(req)
            4'b1000 : next = 3'b100; // missing default statement
        endcase
        ack[3] = 1;
    end
    3'b100: begin
        case(req)
            4'b0000 : next = 3'b000;
            default : next = 3'b100;
        endcase
        err = 1;
    end
endcase
end
endmodule
```

In the above example, the tool detects an FSM livelock at state 3'b011 and reports a violation.

An example of the generated property follows:

```
(#[0:$] (state == 0)) //if 0 is the init state
(state == 0) | => #[0:$] (state == INIT_STATE) //if 0 is not in the init state
```

The init state is detected as the state with the lowest encoding number of the main FSM states group (eliminating the initial sequence states and states in a deadlock).

This check detects livelock within one FSM path; therefore, if an FSM transitions from init state to another state, it will not have a livelock and will revert back to init state. However, if an FSM transition from the init state to another state turns unreachable, this goes undetected. This behavior is intended to reach a balance between adding constraints complexity and covering all potential livelock detection.

Note:

- By default, deadlock and livelock properties are liveness LTL properties, which require you to add fairness assumptions. With LTL modeling, a livelock check fails when the FSM

JasperGold Superlint Checks Reference

Automatic Formal Checks

reaches a non-reset state and one infinite path exists for which the FSM never reaches reset.

- If you prefer to generate deadlock properties using CTL modeling, set the `superlint_fsm_ctl_livelock` variable to `true` before exporting the property. With CTL modeling, a livelock check fails when the FSM reaches a non-reset state from which there exists no path to reset.

[Back to Top](#)

FSM_NO_MTRN

Short Message: *Unreachable consequence transitions (%d1->%d2->%d3) found in the FSM '%s'.*

Severity	Error
Description	FSM transitions checks confirm that a transition from one state to the next state is possible.

The following code illustrates the occurrence of FSM_NO_MTRN.

```
module multi_transitions(clk, resetn, start, finish, curr_state);
    input clk;
    input resetn;
    input start;
    input finish;
    output [1:0] curr_state;
    reg [1:0] curr_state;
    reg [1:0] next_state;
    parameter [1:0] s0=0;
    parameter [1:0] s1=1;
    parameter [1:0] s2=2;
    parameter [1:0] s3=3;

    always @(posedge clk or negedge resetn)
    begin
        if(!resetn)
            curr_state <= s0;
        else
            curr_state <= next_state;
    end
end
```

JasperGold Superlint Checks Reference

Automatic Formal Checks

```
always @(*)
begin
next_state = s0;
case(curr_state)
s0:begin
if(start)
next_state = s1;
else
next_state = s0;
end
s1:begin
next_state = s2;
end
s2:begin
next_state = s0;
end
s3:begin
if(finish)
next_state = s0;
else
next_state = s3;
end
endcase
end
```

endmodule

In the example above, unreachable consequence transitions (0 -> 1 -> 2), (1 -> 2 -> 0), (2 -> 0 -> 1) and (3 -> 0 -> 1) are found in curr_state;

Examples of the generated properties follow:

```
(curr_state == 0) ##1 (curr_state == 1) [+] ##1 (curr_state == 2)
(curr_state == 1) ##1 (curr_state == 2) [+] ##1 (curr_state == 0)
(curr_state == 2) ##1 (curr_state == 0) [+] ##1 (curr_state == 1)
(curr_state == 3) ##1 (curr_state == 0) [+] ##1 (curr_state == 1)
```

[Back to Top](#)

FSM_NO_RCHB

Short Message: *Unreachable states found (%d) in the FSM '%s'.*

Severity	Error
Description	FSM reachable state checks confirm that a FSM state is reachable from the initial FSM state.

The following code illustrates the occurrence of FSM_NO_RCHB.

```
library ieee;
use ieee.std_logic_1164.all;

entity FSM is
port ( clk : in std_logic;
      rst : in std_logic;
      d : in std_logic;
      z : out std_logic);
end FSM;

architecture fsm_rtl of FSM is
type state_type is (state_0, state_1, state_2);
signal next_fsm, fsm : state_type;
begin
  pro_seq: process (clk,rst)
  begin
    if(rst = '0') then
      fsm <= state_0;
    elsif(clk'event and clk = '1') then
      fsm <= next_fsm;
    end if;
  end process;

  pro_comb: process (fsm,d)
  begin
    case fsm is
      when state_0 => z <= '0';
      next_fsm <= state_1;
      when state_1 => z <= '1';
```

JasperGold Superlint Checks Reference

Automatic Formal Checks

```
next_fsm <= state_0;
when state_2 => z <= d;
next_fsm <= state_0;
end case;
end process pro_comb;

end fsm_rtl;
```

In the above code, `State_2` is reported as an unreachable state because there is no path to this state from any other valid state in the FSM.

An example of the generated property follows:

```
(fsm == state_2)
```

[Back to Top](#)

FSM_NO_TRRN

Short Message: *Unreachable transitions found (%d1->%d2) in the FSM '%s'.*

Severity	Error
Description	FSM transitions checks confirm that a transition from one state to the next state is possible.

The following code illustrates the occurrence of `FSM_NO_TRRN`.

```
module fsm_transitions(clk, resetn, start, finish, curr_state);
    input clk;
    input resetn;
    input start;
    input finish;
    output [1:0] curr_state;
    reg [1:0] curr_state;
    reg [1:0] next_state;
    parameter [1:0] s0=0;
    parameter [1:0] s1=1;
    parameter [1:0] s2=2;
    parameter [1:0] s3=3;

    always @(posedge clk or negedge resetn)
    begin
        if(!resetn)
```

JasperGold Superlint Checks Reference

Automatic Formal Checks

```
    curr_state <= s0;
  else
    curr_state <= next_state;
  end

  always @(*)
  begin
    next_state = s0;
    case(curr_state)
      s0:begin
        if(start)
          next_state = s1;
        else
          next_state = s0;
        end
      s1:begin
        next_state = s2;
        end
      s2:begin
        next_state = s0;
        end
      s3:begin
        if(finish)
          next_state = s0;
        else
          next_state = s3;
        end
    endcase
  end

endmodule
```

In this above example, unreachable transitions $0 \rightarrow 1$, $1 \rightarrow 2$, $2 \rightarrow 0$, $3 \rightarrow 0$, and $3 \rightarrow 3$ are found in curr_state;

Examples of the generated properties follow:

```
(curr_state == 0) ##1 (curr_state == 1)
(curr_state == 1) ##1 (curr_state == 2)
(curr_state == 2) ##1 (curr_state == 0)
(curr_state == 3) ##1 (curr_state == 0)
(curr_state == 3) ##1 (curr_state == 3)
```

[Back to Top](#)

AUTO_FORMAL_OUT_OF_BOUND_INDEXING

The AUTO_FORMAL_OUT_OF_BOUND_INDEXING category includes the ARY_IS_OOBI check only.

ARY_IS_OOBI

Short Message: *Out-of-bound indexing failure found.*

Severity	Error
Description	<p>Out-of-bound indexing checks confirm that expressions and arrays are indexed within the defined range. For example, in the following code snippet, index <i>i</i> must be in the range from 0 to 3:</p> <pre>logic [3:0] d_array; out = d_array[i];</pre> <p>A violation might indicate an RTL error, which can cause indexing of an undefined element in the array.</p> <p>To reduce noise that can be associated with out-of-bound indexing checks, use the following command:</p> <pre>set_superlint_enable_observability_tags {ARY_IS_OOBI}</pre>

The following code illustrates the occurrence of ARY_IS_OOBI.

```
module test1 (a, b, out1);  
    input [3:0] a;  
    input [3:0] b;  
    output out1;  
    wire [3:0] a;  
    reg [2:0] out1;  
    wire [2:-1] q;  
    wire [2:-4] w;  
    always @(a or b)  
        out1 = q[4] & b[3];  
        assign a[3] = out1[3];  
        assign q = w[1+:3];  
endmodule
```

JasperGold Superlint Checks Reference

Automatic Formal Checks

In the above example, `q[4]` and `out1[3]` are outside the defined range.

An example of the generated property follows:

```
1'b0
```

AUTO_FORMAL_COMBO_LOOP

The AUTO_FORMAL_COMBO_LOOP category includes the MOD_IS_FCMB check only.

MOD_IS_FCMB

Short Message: *Combinational loop detected passing through '%s'.*

Severity	Error
Description	<p>A combinational loop has been detected involving the listed signals/wires/expressions. As Per RMM section 5.5.3, you should avoid combinational feedback that is looping of combinational processes.</p> <p>To specify the maximum number of loops in the design for which Superlint extracts properties, use the following command:</p> <pre>set_superlint_loop_count_limit <limit></pre> <p>Note: When the value is 0, Superlint issues properties for all loops in the design.</p>

The following code illustrates the occurrence of MOD_IS_FCMB.

```
library ieee;
use ieee.std_logic_1164.all;
entity TEST is
  port (
    a,b,c,d,e,f: buffer std_logic;
    o ,o2 : out std_logic);
end;

Architecture structural of test is
  component NAND_2
  port (I0, I1: in std_logic;
        o: buffer std_logic);
  end component;
```


JasperGold Superlint Checks Reference

Automatic Formal Checks

```
component OR_2
  port (I0, I1: in std_logic;
        o:      buffer std_logic);
end component;

begin
  x1: NAND_2 port map (a, b,c);
  x2: OR_2 port map (c,d,b);
end structural;

library ieee;
use ieee.std_logic_1164.all;

entity nand_2 is
  port (I0, I1: in std_logic;
        o: buffer std_logic);
end;
architecture dataflow of NAND_2 is
begin
  O <= I0 nand I1;
end dataflow

library ieee;
use ieee.std_logic_1164.all;

entity OR_2 is
  port (I0, I1: in std_logic;
        o : buffer std_logic);
end entity;

architecture dataflow of OR_2 is
begin
  O <= I0 or I1;
end dataflow;
```

In the above example, c is NAND of a and b and b is OR of c and d resulting in a combinational feedback loop between c and b. This kind of looping of combinational processes should be avoided.