Check Usage

**Software Quality Team**

ABSTRACT

xxxxxxxx

# CONTENTS

[CONTENTS 2](#_Toc112840785)

[Support methods 6](#_Toc112840786)

[1 check\_lines 6](#_Toc112840787)

[1.1 Format 6](#_Toc112840788)

[1.2 Description 6](#_Toc112840789)

[1.3 Demo 6](#_Toc112840790)

[2 check\_flow 7](#_Toc112840791)

[2.1 Format 7](#_Toc112840792)

[2.2 Description 7](#_Toc112840793)

[2.3 Demo 7](#_Toc112840794)

[3 check\_synpro 8](#_Toc112840795)

[3.1 Format 8](#_Toc112840796)

[3.2 Description 8](#_Toc112840797)

[3.3 Demo 8](#_Toc112840798)

[4 check\_lse 9](#_Toc112840799)

[4.1 Format 9](#_Toc112840800)

[4.2 Description 9](#_Toc112840801)

[4.3 Demo 9](#_Toc112840802)

[5 check\_map 10](#_Toc112840803)

[5.1 Format 10](#_Toc112840804)

[5.2 Description 10](#_Toc112840805)

[5.3 Demo 10](#_Toc112840806)

[6 check\_partrce 11](#_Toc112840807)

[6.1 Format 11](#_Toc112840808)

[6.2 Description 11](#_Toc112840809)

[6.3 Demo 11](#_Toc112840810)

[7 check\_block 12](#_Toc112840811)

[7.1 Format 12](#_Toc112840812)

[7.2 Description 12](#_Toc112840813)

[7.3 Demo 12](#_Toc112840814)

[8 check\_multiline 13](#_Toc112840815)

[8.1 Format 13](#_Toc112840816)

[8.2 Description 13](#_Toc112840817)

[8.3 Demo 13](#_Toc112840818)

[9 check\_no 14](#_Toc112840819)

[9.1 Format 14](#_Toc112840820)

[9.2 Description 14](#_Toc112840821)

[9.3 Demo 14](#_Toc112840822)

[10 check\_grep 15](#_Toc112840823)

[10.1 Format 15](#_Toc112840824)

[10.2 Description 15](#_Toc112840825)

[10.3 Demo 15](#_Toc112840826)

[11 sim\_check\_block 16](#_Toc112840827)

[11.1 Format 16](#_Toc112840828)

[11.2 Description 16](#_Toc112840829)

[11.3 Demo 16](#_Toc112840830)

[12 check\_sdf 17](#_Toc112840831)

[12.1 Format 17](#_Toc112840832)

[12.2 Description 17](#_Toc112840833)

[12.3 Demo 17](#_Toc112840834)

[13 check\_file 18](#_Toc112840835)

[13.1 Format 18](#_Toc112840836)

[13.2 Description 18](#_Toc112840837)

[13.3 Demo 18](#_Toc112840838)

[14 check\_lut\_reference 19](#_Toc112840839)

[14.1 Format 19](#_Toc112840840)

[14.2 Description 19](#_Toc112840841)

[14.3 Demo 19](#_Toc112840842)

[15 check\_clk\_reference 20](#_Toc112840843)

[15.1 Format 20](#_Toc112840844)

[15.2 Description 20](#_Toc112840845)

[15.3 Demo 20](#_Toc112840846)

[16 check\_binary 21](#_Toc112840847)

[16.1 Format 21](#_Toc112840848)

[16.2 Description 21](#_Toc112840849)

[16.3 Demo 21](#_Toc112840850)

[17 check\_diamond\_flow 22](#_Toc112840851)

[17.1 Format 22](#_Toc112840852)

[17.2 Description 22](#_Toc112840853)

[17.3 Demo 22](#_Toc112840854)

[18 check\_radiant\_flow 23](#_Toc112840855)

[18.1 Format 23](#_Toc112840856)

[18.2 Description 23](#_Toc112840857)

[18.3 Demo 23](#_Toc112840858)

[19 check\_simrel 24](#_Toc112840859)

[19.1 Format 24](#_Toc112840860)

[19.2 Description 24](#_Toc112840861)

[19.3 Demo 24](#_Toc112840862)

[20 check\_greater 25](#_Toc112840863)

[20.1 Format 25](#_Toc112840864)

[20.2 Description 25](#_Toc112840865)

[20.3 Demo 25](#_Toc112840866)

[21 check\_less 26](#_Toc112840867)

[21.1 Format 26](#_Toc112840868)

[21.2 Description 26](#_Toc112840869)

[21.3 Demo 26](#_Toc112840870)

[22 check\_almost 27](#_Toc112840871)

[22.1 Format 27](#_Toc112840872)

[22.2 Description 27](#_Toc112840873)

[22.3 Demo 27](#_Toc112840874)

[23 check\_sim\_rtl 29](#_Toc112840875)

[23.1 Format 29](#_Toc112840876)

[23.2 Description 29](#_Toc112840877)

[23.3 Demo 29](#_Toc112840878)

[24 check\_sim\_syn 30](#_Toc112840879)

[24.1 Format 30](#_Toc112840880)

[24.2 Description 30](#_Toc112840881)

[24.3 Demo 30](#_Toc112840882)

[25 check\_sim\_par 31](#_Toc112840883)

[25.1 Format 31](#_Toc112840884)

[25.2 Description 31](#_Toc112840885)

[25.3 Demo 31](#_Toc112840886)

[26 check\_sdf\_simflow 32](#_Toc112840887)

[26.1 Format 32](#_Toc112840888)

[26.2 Description 32](#_Toc112840889)

[26.3 Demo 32](#_Toc112840890)

[27 check\_simulation\_flow 33](#_Toc112840891)

[27.1 Format 33](#_Toc112840892)

[27.2 Description 33](#_Toc112840893)

[27.3 Demo 33](#_Toc112840894)

[28 check\_value 34](#_Toc112840895)

[28.1 Format 34](#_Toc112840896)

[28.2 Description 34](#_Toc112840897)

[28.3 Demo 34](#_Toc112840898)

[29 check\_rbt 35](#_Toc112840899)

[29.1 Format 35](#_Toc112840900)

[29.2 Description 35](#_Toc112840901)

[29.3 Demo 35](#_Toc112840902)

[Unified Parameters 36](#_Toc112840903)

[1 show 36](#_Toc112840904)

[2 cr\_note 36](#_Toc112840905)

# Support methods

## 1 check\_lines

Check lines method is used to check one or multiple specific string(s) in a given file.

### 1.1 **Format**

[check\_lines\_<section\_num>]

title = -- optional

file =

check\_1 = <string\_1>

times|times\_strict|times\_range = ..

check\_<num> = <string\_2> -- optional

use\_grep = <0 (default), 1> --optional, default: 0

### 1.2 Description

1. “[check\_lines\_<section\_num>]”: a configuration file may have multiple “check\_lines” methods but we need identify them with different section number.
2. “title”: title for this check method, optional.
3. “file”: file path to be used for ‘specific string’ searching.
4. “check\_1”: search “string\_1” in file and make this line as the start line (num = 1).
5. “times”: repeat times before toggle the “check\_1”.
6. “times\_strict”: strict repeat times for toggle the “check\_1”
7. “times\_range”: repeat times for toggle the “check\_1, for example “>=4”
8. “check\_<num>”: find “string\_2” in line “num”
9. “use\_grep”: if set to 1, will treat check string as regular expression

In summary:

This method will try to find the <string\_1> assigned by “check\_1” in <file> and treat it as a start point (line 1), if “times” option is used, script will repeat <repeat\_number> times and then treat the last one as start point) then try to check whether line <num> (a shift value) has <string\_2>.

If all conditions are matched, this check section will return pass.

### 1.3 Demo

#### 1.3.1 Single line check

Check specific string in twr report

[check\_lines\_1]

title = check\_test1

file = ./\_scratch/impl/RAM\_DP\_TRUE\_impl.twr

check\_1 = 100.00% coverage

#### 1.3.2 Continues lines check

Check the contents of the two consecutive lines in twr report

[check\_lines\_1]

title = check\_test1

file = ./\_scratch/impl/RAM\_DP\_TRUE\_impl.twr

check\_1 = Preference: FREQUENCY PORT "ClockA" 100.000000 MHz ;

check\_2 = 0 items scored

#### 1.3.3 Continues lines check and with check\_1 times settings

Check the contents of the two consecutive lines in twr report, string for check\_1 should be matched as few as $times in the file.

[check\_lines\_1]

title = check\_test1

file = ./\_scratch/impl/RAM\_DP\_TRUE\_impl.twr

check\_1 = Preference: FREQUENCY PORT "ClockA" 100.000000 MHz ;

times = 4

check\_2 = 0 items scored

#### 1.3.4 Continues lines check and with check\_1 times\_strict settings

Check the contents of the two consecutive lines in twr report, string for check\_1 should be matched as same as $times\_strict in the file.

[check\_lines\_1]

title = check\_test1

file = ./\_scratch/impl/RAM\_DP\_TRUE\_impl.twr

check\_1 = Preference: FREQUENCY PORT "ClockA" 100.000000 MHz ;

times\_ strict = 4

check\_2 = 0 items scored

## 2 check\_flow

This method is used to check string "All signals are completely routed" in the par report.

### 2.1 Format

[check\_flow\_<section\_num>]

title= --optional

file = <path>/<file>

### 2.2 Description

1. “[check\_flow\_<section\_num>]”: a configuration file may have multiple “check\_flow” methods but we need identify them with different section number.
2. “title”: title for this check method, optional.
3. “file”: file path to be checked, <file> need to be a par report.

In summary:

This method will try to find the string "All signals are completely routed" in the report.

If all conditions are matched, this check section will return pass.

### 2.3 Demo

[check\_flow\_1]

file = ./\_scratch/\*/\*.par

## 3 check\_synpro

This method is used to check string “@E:” and “Mapper successful” in the synpro report.

### 3.1 Format

[check\_synpro\_<section\_num>]

Titile= --optional

file = <path>/<file>

### 3.2 Description

1. “[check\_synpro\_<section\_num>]”: a configuration file may have multiple “check\_synpro” methods but we need identify them with different section number.
2. “title”: title for this check method, optional.
3. “file”: file path to be checked, <file> need to be a synpro report.

In summary:

This method will confirm synpro report with "Mapper successful" and without “@E:” then it will report ‘pass’ otherwise ‘fail’.

### 3.3 Demo

[check\_synpro\_1]

file = ./\_scratch/\*/\*.srr

## 4 check\_lse

This method is used to check string “ERROR” and “CPU time for LSE flow” in the LSE report.

### 4.1 Format

[check\_lse\_<section\_num>]

title= --optional

file = <path>/<file>

### 4.2 Description

1. “[check\_lse\_<section\_num>]”: a configuration file may have multiple “check\_lse” methods but we need identify them with different section number.
2. “title”: title for this check method, optional.
3. “file”: file path to be checked, <file> need to be a LSE report.

In summary:

This method will confirm LSE report with " CPU time for LSE flow " and without “ERROR” then it will report ‘pass’ otherwise ‘fail’.

### 4.3 Demo

[check\_lse\_1]

file = ./\_scratch/\*/ synthesis.log

## 5 check\_map

This method is used to check string “ERROR” and “Peak Memory Usage” in the map report.

### 5.1 Format

[check\_map\_<section\_num>]

file = <path>/<file>

### 5.2 Description

1. “[check\_map\_<section\_num>]”: a configuration file may have multiple “check\_map” methods but we need identify them with different section number.
2. “title”: title for this check method, optional.
3. “file”: file path to be checked, <file> need to be a map report.

In summary:

It will try to find line start with “ERROR” in report. If string is found, the check section will return fail.

It will try to find string "Peak Memory Usage" in report. If string is found, the check section will return pass.

### 5.3 Demo

[check\_map\_1]

file = ./\_scratch/\*/ \*.mrp

## 6 check\_partrce

This method is used to check string “Cumulative negative slack” in the place & route trace report.

### 6.1 Format

[check\_partrce\_<section\_num>]

file = <path>/<file>

### 6.2 Description

1. “[check\_partrce\_<section\_num>]”: a configuration file may have multiple “check\_partrce” methods but we need identify them with different section number.
2. “title”: title for this check method, optional.
3. “file”: file path to be checked, <file> need to be a place & route trace report.

In summary:

It will try to find string "Cumulative negative slack" in report. If string is found, the check section will return pass.

### 6.3 Demo

[check\_partrce\_1]

file = ./\_scratch/\*/ \*.twr

## 7 check\_block

This method is used to check golden file included in compare file.

### 7.1 Format

[check\_ block\_<section\_num>]

golden\_file = <path>/< golden\_file >

compare\_file = <path>/<compare\_file>

### 7.2 Description

1. “[check\_block\_<section\_num>]”: a configuration file may have multiple “check\_block” methods but we need identify them with different section number.
2. “title”: title for this check method, optional.
3. “golden\_file”: golden file path to be checked.
4. “compare\_file”: compare file path to be checked.

In summary:

It will try to compare the <golden\_file> and <compare\_file>. If <golden\_file> is included in <compare\_file>, the check section will return pass.

### 7.3 Demo

[check\_block——1]

golden\_file = ./\_scratch/sim\_rtl/test\_out

compare\_file = ./\_scratch/sim\_map\_vlg/test\_out

## 8 check\_multiline

This method is used to check a continuous string regardless the "space" and "line break" in a given file.

### 8.1 Format

[check\_multiline\_<section\_num>]

file = <path>/<file >

check\_line = <strings\_in\_multi\_lines>

### 8.2 Description

1. “[check\_multiline\_<section\_num>]”: a configuration file may have multiple “check\_multiline” methods but we need identify them with different section number.
2. “file”: file path to be checked.
3. “check\_line”: string to be checked, regardless the "space" and "line break".

In summary:

It will try to find a continuous string regardless the "space" and "line break" in a given file. If string is found, the check section will return pass.

### 8.3 Demo

[check\_multiline]

file = par\ecp3\impl\GSR\_NET\_impl.mrp

check\_line = “Target Vendor: LATTICE Target Device: LFE3-95EAFPBGA484”

## 9 check\_no

This method is used to check the specific string not in a given file.

### 9.1 Format

[check\_no\_<section\_num>]

file = <path>/< file >

check\_line =<string>

### 9.2 Description

1. “[check\_no\_<section\_num>]”: a configuration file may have multiple “check\_no” methods but we need identify them with different section number.
2. “title”: title for this check method, optional.
3. “file”: file path to be checked.
4. “check\_line”: string to be checked.

In summary:

It will try to find the <string> in a given file. If the <string> is not found, the check section will return pass.

### 9.3 Demo

[check\_no\_1]

File = a.txt

check\_line = df1

## 10 check\_grep

This method is used to check the specific regular expression in a given file.

### 10.1 Format

[check\_grep\_<section\_num>]

file = <path>/<file>

grep =<string>

modifier = <string>

action= <positive (default), negative> --optional, default ‘positive’

### 10.2 Description

1. “[check\_grep\_<section\_num>]”: a configuration file may have multiple “check\_grep” methods but we need identify them with different section number.
2. “file”: file path to be checked.
3. “grep”: pattern of regular expression.
4. “modifier”: match of regular expression.
5. “action”: string to find regular expression or not, default value is positive, optional

In summary:

The default value for action is positive, check section will return pass when regular expression is found.

If action = negative, the check section will return fail when the regular expression is found.

### 10.3 Demo

[check\_grep\_1]

file = ./\_scratch/test\_results.log

grep = ^fail

modifier = re.I

action = negative

## 11 sim\_check\_block

This method is used to check golden file included in compare file, ignore “x”, “X”, “z”, “Z”, “?” strings difference.

### 11.1 Format

[sim\_check\_ block\_<section\_num>]

golden\_file = <path>/< golden\_file >

compare\_file = <path>/<compare\_file>

### 11.2 Description

1. “[sim\_check\_block\_<section\_num>]”: a configuration file may have multiple “sim\_check\_block” methods but we need identify them with different section number.
2. “title”: title for this check method, optional.
3. “golden\_file”: golden file path to be checked.
4. “compare\_file”: compare file path to be checked.

In summary:

It will try to compare the <golden\_file> and <compare\_file>. If <golden\_file> is included in <compare\_file> and only “x”, “X”, “z”, “Z”, “?” strings difference, the check section will return pass.

### 11.3 Demo

[sim\_check\_block]

golden\_file = ./\_scratch/sim\_rtl/test\_out

compare\_file = ./\_scratch/sim\_map\_vlg/test\_out

## 12 check\_sdf

This method is used to check the sdf file.

### 12.1 Format

[check\_sdf\_<section\_num>]

sdf\_dir = <path>/< file >

### 12.2 Description

1. “[check\_sdf\_<section\_num>]”: a configuration file may have multiple “check\_sdf” methods but we need identify them with different section number.
2. “title”: title for this check method, optional.
3. “sdf\_dir”: file path to be checked, <file> need to be sdf file.

In summary:

If the file matches sdf format, the check section will return pass.

### 12.3 Demo

[check\_sdf\_1]

sdf\_dir = ./\_scratch/\*/\*.sdf

## 13 check\_file

This method is used to check whether the specific file exists or not.

### 13.1 Format

[check\_file\_<section\_num>]

file = <path>/< file >

reverse = [0 (default), 1]

### 13.2 Description

1. “[check\_file\_<section\_num>]”: a configuration file may have multiple “check\_file” methods but we need identify them with different section number.
2. “title”: title for this check method, optional.
3. “file”: file path to be checked.
4. “reverse”: boolean value to find file or not ,default value is 0, optional.

In summary:

The default value for reverse is 0, the check section will return pass when the <file> is found.

If reverse = 1, the check section will return fail when the <file> is found.

### 13.3 Demo

[check\_file]

file = ./\_scratch/\*/\*.sdc

reverse = 0

## 14 check\_lut\_reference

This method is used to check whether LUT number is correct.

### 14.1 Format

[check\_lut\_reference\_<section\_num>]

file = <path>/< file >

check\_pattern = <string>

init\_result = <num>

allowance = <num>

### 14.2 Description

1. “[check\_lut\_reference\_<section\_num>]”: a configuration file may have multiple “check\_lut\_reference” methods but we need identify them with different section number.
2. “file”: file path to be checked.
3. “check\_pattern”: regular expression.
4. “init\_result”: number for base value.
5. “allowance”: percentage for tolerance range.

In summary:

It will try to find the value by regular expression in given file. If the value is in the tolerance range, the check section will return pass.

### 14.3 Demo

[check\_lut\_reference]

check\_pattern = Number of LUT4s:\s+(\d+)

file = ./\_scratch/\*/\*.mrp

init\_result = 100

allowance = 5

If the lut number is in [100-100\*5%, 100+100\*5%], the check section will return pass.

## 15 check\_clk\_reference

This method is used to check whether clock value is correct.

### 15.1 Format

[check\_clk\_reference\_<section\_num>]

file = <path>/< file >

check\_pattern = <string>

init\_result = <num>

allowance = <num>

clk\_name = <string>

### 15.2 Description

1. “[check\_clk\_reference\_<section\_num>]”: a configuration file may have multiple “check\_clk\_reference” methods but we need identify them with different section number.
2. “file”: file path to be checked.
3. “check\_pattern”: regular expression.
4. “init\_result”: number for base value.
5. “allowance”: percentage for tolerance range.
6. “clk\_name”: string for clock name.

In summary:

It will try to find the value by regular expression in given file. If the value is in the tolerance range, the check section will return pass.

### 15.3 Demo

[check\_clk\_reference]

check\_pattern = PAP=.\*?\|\s+([\d\.]+)\s\*MHz\s\*\|\s\*([\d\.]+)\s\*MHz.\*?FREQUENCY NET "MachXO\_programmer/tck" 100.000000 MHz ;

file = ./\_scratch/\*/\*.twr

init\_result = 100

allowance = 10

clk\_name = MachXO\_programmer/tck

If the clock value is in [100-100\*10%, 100+100\*10%], the check section will return pass.

## 16 check\_binary

This method is used to check whether the specific binary files are same.

### 16.1 Format

[check\_binary\_<section\_num>]

golden\_file = <path>/< golden\_file >

compare\_file = <path>/<compare\_file>

partial = 0/1 --optional

### 16.2 Description

1. “[check\_binary\_<section\_num>]”: a configuration file may have multiple “check\_binary” methods but we need identify them with different section number.
2. “golden\_file”: golden file path to be checked, <golden\_file> need to be a binary file.
3. “compare\_file”: compare file path to be checked, <compare\_file> need to be a binary file.

In summary:

If <golden\_file> and <compare\_file> are same, the check section will return pass.

D) “partial”: flag whether allow golden\_file is part of compare\_file

### 16.3 Demo

[check\_binary\_1]

golden\_file = ./\_scratch/sim\_rtl/test\_out

compare\_file = ./\_scratch/sim\_map\_vlg/test\_out

## 17 check\_diamond\_flow

This method is used to check whether the specific flow passes.

### 17.1 Format

[check\_diamond\_flow\_<section\_num>]

check\_flow = <flow>

### 17.2 Description

1. “[check\_diamond\_flow\_<section\_num>]”: a configuration file may have multiple “check\_diamond\_flow” methods but we need identify them with different section number.
2. “check\_flow”: flow result to be checked, <flow> need to refer to following table.

|  |  |  |
| --- | --- | --- |
| **Flow** | **File** | **Check string** |
| synp | \_scratch/\*/\*.srr | Mapper successful! |
| lse | \_scratch/\*/synthesis.log | CPU Time for LSE flow |
| synthesis | Synp or lse pass | Synp or lse pass |
| map | \_scratch/\*/\*.mrp | Number of errors:    0 |
| par | \_scratch/\*/\*.par | All signals are completely routed. |
| ibis | \_scratch/\*/IBIS/\*.ibs | FILE\_EXISTS |
| bitstream | \_scratch/\*/\*.bit;\_scratch/\*/\*.rbt | FILE\_EXISTS |
| jedec | \_scratch/\*/\*.jed | FILE\_EXISTS |
| prom | \_scratch/\*/\*.mcs | FILE\_EXISTS |
| download | \_scratch/\*/\*.bit;\_scratch/\*/\*.rbt;\_scratch/\*/\*.jed;  \_scratch/\*/\*.bin;\_scratch/\*/\*.hex;\_scratch/\*/\*.nvcm | FILE\_EXISTS |

In summary:

It will try to find the check string in specific file. If string is found, the check section will return pass.

### 17.3 Demo

[check\_diamond\_flow]

check\_flow = map

## 18 check\_radiant\_flow

This method is used to check whether the specific flow passes.

### 18.1 Format

[check\_radiant\_flow\_<section\_num>]

check\_flow = <flow>

### 18.2 Description

1. “[check\_radiant\_flow\_<section\_num>]”: a configuration file may have multiple “check\_radiant\_flow” methods but we need identify them with different section number.
2. “check\_flow”: flow result to be checked, <flow> need to refer to below table.

|  |  |  |
| --- | --- | --- |
| **Flow** | **File** | **Check string** |
| synp | \_scratch/\*/\*.srr | Mapper successful! |
| lse | \_scratch/\*/synthesis.log | CPU Time |
| synthesis | Synp or lse pass | Synp or lse pass |
| map | \_scratch/\*/\*.mrp | Number of errors:    0 |
| par | \_scratch/\*/\*.par | All signals are completely routed. |
| ibis | \_scratch/\*/IBIS/\*.ibs | FILE\_EXISTS |
| bitstream | \_scratch/\*/\*.bit;\_scratch/\*/\*.rbt | FILE\_EXISTS |
| jedec | \_scratch/\*/\*.jed | FILE\_EXISTS |
| prom | \_scratch/\*/\*.mcs | FILE\_EXISTS |
| download | \_scratch/\*/\*.bit;\_scratch/\*/\*.rbt;\_scratch/\*/\*.jed;  \_scratch/\*/\*.bin;\_scratch/\*/\*.hex;\_scratch/\*/\*.nvcm | FILE\_EXISTS |

In summary:

It will try to find the check string in specific file. If string is found, the check section will return pass.

### 18.3 Demo

[check\_radiant\_flow]

check\_flow = map

## 19 check\_simrel

This method is used to compare avc and out file of simrel results.

### 19.1 Format

[check\_simrel]

avc = default: ./\_scratch/simrel/fc.avc

out = default: ./\_scratch/simrel/fc.out

acc = default: 1.0

ignore = default: 0

shift = default: 0

signals = default: all

### 19.2 Description

a) avc, out: the golden and simrel result files

b) acc: the desired accuracy the comparison needs to get

c) ignore: how many lines ignore at the beginning

d) shift: how many lines needs shift when do comparing

e) signals: partial shift for specific signals, higher priority than shift

In summary:

It will try to compare the avc and out file about only the output, namely LHX, and return the ratio of similarity. If the ratio is less than acc, the check fail.

### 19.3 Demo

[check\_simrel]

;acv = xxx.avc

;out = xxx.out

acc = 0.917489200857

ignore = 1047

shift = 1

signals = data\_out[\d+]:10, o\_oea:10

## 20 check\_greater

This method is used to compare the searched number and golden.

### 20.1 Format

[check\_greater\_<label>]

file =

pattern =

group\_name = default: compare

golden =

### 20.2 Description

a) file: the file being searched

b) pattern: the pattern to search the number, must set the group name using (?P<group\_name>).

c) group\_name: the group name in the pattern

d) golden: the golden number

In summary:

It will try to search the desired pattern in the file and get a number. If the number is greater than golden, check pass, otherwise check fail.

### 20.3 Demo

file = ./\_scratch/synthesis\*.log

pattern = Peak Memory Usage: (?P<memory>[\d.]+) MB

group\_name = memory

golden = 503

## 21 check\_less

This method is used to compare the searched number and golden.

### 21.1 Format

[check\_less\_<label>]

file =

pattern =

group\_name = --optional, default: 1

golden =

### 21.2 Description

a) file: the file being searched

b) pattern: the pattern to search the number, must set the group name using (?P<group\_name>).

c) group\_name: the group name in the pattern

d) golden: the golden number

In summary:

It will try to search the desired pattern in the file and get a number. If the number is less than golden, check pass, otherwise check fail.

### 21.3 Demo

file = ./\_scratch/synthesis\*.log

pattern = Peak Memory Usage: (?P<memory>[\d.]+) MB

group\_name = memory

golden = 503

## 22 check\_almost

This method is used to compare the searched number and golden.

### 22.1 Format

[check\_almost\_<label>]

file =

pattern =

group\_name =

golden =

tolerance =

Or:

[check\_almost\_<label>]

file =

pattern =

formula =

tolerance =

before\_pattern\_10 =

### 22.2 Description

a) file: the file being searched

b) pattern: the pattern to search the number, must set the group name using (?P<group\_name>).

c) group\_name: the group name in the pattern, default is 1 in the match object

d) golden and formula: golden value or formula, golden has higher priority

e) tolerance: the tolerance of difference ratio between searched number and golden. Searched number must between ( expected\_value \* (1 ± tolerance)

f) before\_pattern\_10: search before\_pattern\_10 firstly, then search the pattern in 10 lines. 10 can be other integer, such as 15.

In summary:

It will try to search the desired pattern in the file and get a number. If the difference ratio of number and golden is less than tolerance, check pass, otherwise check fail. And the scripts will try to stop when get pass string.

### 22.3 Demo

file = ./\_scratch/synthesis\*.log

pattern = Peak Memory Usage: (?P<memory>[\d.]+) MB

group\_name = memory

golden = 503

tolerance = 0.1

Or:

file = ./\_scratch/\*/top\_impl1.twr

formula = %(ECLK\_PERIOD)s + %(START\_DELAY)s - ( %(ECLK\_PERIOD)s \* 2.0)

ECLK\_PERIOD = 5

START\_DELAY = 0.101

pattern = - Setup Time ([\.\d-]+)

tolerance = 0.05

before\_pattern\_5 = search it firstly

## 23 check\_sim\_rtl

This method is used to search “PASS” and “FAIL” in RTL simulation results log file.

### 23.1 Format

[check\_sim\_rtl]

; file = ./\_scratch/sim\_rtl/outlog.log

; passkey = PASS

; failkey = FAIL

; language = verilog

show = 1

### 23.2 Description

a) file: the file will be searched.

b) passkey: keyword “PASS” in file

c) failkey: key word “FAIL” in file

d) language: verilog (reserved)

In summary:

Check RTL simulation results. It will try to search keyword “FAIL” and “PASS” in $file.

1. If not found $file, Failed;
2. If not found $passkey in $file, Failed;
3. If found $failkey in $file, Failed.
4. Others, Passed.

### 23.3 Demo

[check\_sim\_rtl]

show=1

## 24 check\_sim\_syn

This method is used to search “PASS” and “FAIL” in post synthesis simulation results log file.

### 24.1 Format

[check\_sim\_syn]

; file = ./\_scratch/sim\_syn\_vlg/outlog.log

; passkey = PASS

; failkey = FAIL

; language = verilog

show = 1

### 24.2 Description

a) file: the file will be searched.

b) passkey: keyword “PASS” in file

c) failkey: key word “FAIL” in file

d) language: verilog

In summary:

Check RTL simulation results. It will try to search keyword “FAIL” and “PASS” in $file.

If lauguage is vhdl, check outlog.log file in folder “./\_scratch/sim\_syn\_vhd”, else use default.

1. If not found $file, Failed;
2. If not found $passkey in $file, Failed;
3. If found $failkey in $file, Failed.
4. Others, Passed.

### 24.3 Demo

[check\_sim\_syn]

show=1

## 25 check\_sim\_par

This method is used to search “PASS” and “FAIL” in RTL simulation results log file.

### 25.1 Format

[check\_sim\_par]

; file = \_scratch/sim\_par\_vlg/outlog.log

; passkey = PASS

; failkey = FAIL

; language = verilog

show = 1

### 25.2 Description

a) file: the file will be searched.

b) passkey: keyword “PASS” in file

c) failkey: key word “FAIL” in file

d) language: verilog

In summary:

Check RTL simulation results. It will try to search keyword “FAIL” and “PASS” in $file.

If lauguage is vhdl, check outlog.log file in folder “./\_scratch/sim\_par\_vhd”, else use default.

1. If not found $file, Failed;
2. If not found $passkey in $file, Failed;
3. If found $failkey in $file, Failed.
4. Others, Passed.

### 25.3 Demo

[check\_sim\_par]

show=1

## 26 check\_sdf\_simflow

This method is used to check sdf simulation flow in sim\_par\_vlg log file.

### 26.1 Format

[check\_sdf\_simflow]

; file = \_scratch/sim\_par\_vlg/run\_sim\_par\_vlg.log

; passkey1 = SDF Backannotation Successfully (modelsim, after vsim command line) | VSIM: Simulation has finished (active-hdl)

; passkey2 = Errors: 0 (modelsim, after vsim command line) | NA (active-hdl)

; failkey1 = Failed to parse SDF (modelsim) | SDF: 0 SDF entries loaded (active-hdl)

; failkey2 = Error loading design (modelsim) | SDF: Error (active-hdl)

;use\_grep = <0 (default), 1>

### 26.2 Description

1. File: check log file
2. Both passkey1 and passkey2 must be found
3. Any failkey1 or failkey2 found result will be failed.
4. Use\_grep: switch for python regexp support.

### 26.3 Demo

[check\_sdf\_simflow]

show=1

## 27 check\_simulation\_flow

This method is used to check simulation flow for ActiveHDL and modelsim

### 27.1 Format

[check\_simulation\_flow]

; check\_flow = rtl, syn\_vlg, syn\_vhd, map\_vlg, map\_vhd, par\_vlg, par\_vhd

; rtl\_file = \_scratch/sim\_rtl/run\_sim\_rtl.log

; syn\_vlg\_file = \_scratch/sim\_syn\_vlg/run\_sim\_syn\_vlg.log

; syn\_vhd\_file = \_scratch/sim\_syn\_vhd/run\_sim\_syn\_vhd.log

; map\_vlg\_file = \_scratch/sim\_map\_vlg/run\_sim\_map\_vlg.log

; map\_vhd\_file = \_scratch/sim\_map\_vhd/run\_sim\_map\_vhd.log

; par\_vlg\_file = \_scratch/sim\_par\_vlg/run\_sim\_par\_vlg.log

; par\_vhd\_file = \_scratch/sim\_par\_vhd/run\_sim\_par\_vhd.log

; msim\_pass1 = # Errors: 0

; msim\_fail1 = Errors: [1-9]

; msim\_fail2 = # \*\* Error

; asim\_pass1 = VSIM: Simulation has finished

; use\_grep = <1 (default), 0>

### 27.2 Description

1. Key check\_flow is used to list which simulation stage will be checked, separate with comma. Valid choices are rtl, syn\_vlg, syn\_vhd, map\_vlg, map\_vhd, par\_vlg, par\_vhd
2. Key \*\*\_file can be extended from check\_flow name
3. Script will invoke related check based on current simulator (modelsim link:msim\_xxxx, aldec link:asim\_xxx)
4. All ‘xxxx\_passx’ check string must be found before final PASS, any ‘xxxx\_failx’ check string will result a final FAIL

### 27.3 Demo

Demo 1:

[check\_simulation\_flow]

check\_flow = rtl, syn\_vlg, par\_vlg

Demo 2:

[check\_simulation\_flow]

check\_flow = rtl, par\_vlg

rtl\_file = \_scratch/sim\_rtl/sim\_log.txt

msim\_pass1 = # Errors: 0

msim\_fail1 = Error loading design

## 28 check\_value

This method is used to check values in report file. Which must be used together with \_get\_value\_xxx function.

### 28.1 Format

[check\_value]

title = check lut number for different map arguments

judge = {lut\_one} > {lut\_updated}

[\_get\_value\_lut\_one]

file = ./\_scratch/impl1/\*.mrp

pattern = Number of SLICEs: (\d+)

[\_get\_value\_lut\_updated]

file = ./\_scratch/impl2/\*.mrp

pattern = Number of SLICEs: (\d+)

### 28.2 Description

1. get value by [\_get\_value\_xxx], xxx is the value name;
2. eval judge string and get True or False;
3. no blank in section name;
4. in order to use judge string in Python scripts, no blank between curly brace;
5. Section [\_get\_value\_xxx] valid choices are:
   1. file: specify report file name or search string
   2. pattern | (pattern\_1, pattern\_2, …): pattern has higher priority than others. When 2 or more patterns for search value, use pattern\_1, pattern\_2, …
   3. [before\_pattern\_$d]: search before\_pattern\_$d firstly, then search the pattern in $d lines. $d is an integer which is bigger than 0
   4. [default\_value]: return this value when no value found in report file
   5. [p\_index | p\_name]: regular expression settings. Used by m.group(p\_index)
   6. [p\_flags]: regular expression settings. For example, re.I | re.X

### 28.3 Demo

[\_get\_value\_lut\_one]

file = ./\_scratch/impl1/\*.mrp

pattern = Number of LUT4s: (\d+)

p\_flags = re.I

[\_get\_value\_slice\_one]

file = ./\_scratch/impl1/\*.mrp

pattern\_1 = Number of slice registers: (\d+)

pattern\_2 = Number of SLICEs: (\d+)

p\_flags = re.I

[\_get\_value\_setup\_slack]

file = ./\_scratch/impl1/\*.par

before\_pattern\_8 = All signals are completely routed.

; PAR\_SUMMARY::Worst slack<setup/<ns>> = 0.614

pattern = PAR\_SUMMARY::Worst\s+slack<setup.+=\s+([\d\.]+)

[check\_value\_1]

judge = {lut\_one} > 899

[check\_value\_2]

judge = {lut\_one} >= {setup\_slack} \* 1.8 and {slice\_one} != {lut\_one}

## 29 check\_rbt

This method is used to check the rbt files generated with different Radiant build.

### 29.1 Format

[check\_ rbt\_<section\_num>]

golden\_file = <path>/< golden\_file >

compare\_file = <path>/<compare\_file>

### 29.2 Description

1. “[check\_rbt\_<section\_num>]”: a configuration file may have multiple “check\_rbt” methods but we need identify them with different section number.
2. “title”: title for this check method, optional.
3. “golden\_file”: golden file path to be checked.
4. “compare\_file”: compare file path to be checked.

In summary:

It will try to compare the <golden\_file> and <compare\_file>. It will check the first 30 lines in rbt file and try to compare the items like: "Rows", "Cols", "Bits", "BitstreamCRC", "FileFormat". It all value are same, will return pass.

### 29.3 Demo

[check\_rbt\_1]

golden\_file= \_old/\*/\*.rbt

compare\_file= \_new/\*/\*.rbt

## 30 check\_clock\_skew

This method is used to check clock skew number between twr data and par data.

### 30.1 Format

[check\_clock\_skew\_<section\_num>]]

; KEYWORDS: "get\_twr.", "get\_par." and ".string", ".regexp"

file = ./\_scratch/impl/\*.twr

trunk = 5

mode = more

get\_twr.0.string = Path Begin : temp4\_Z/Q (SLICE\_R5C1F)

get\_twr.1.string = Path End : Qout2\_Z/DF (SLICE\_R54C50E)

get\_twr.7.regexp = Setup Constraint :

get\_twr.8.regexp = Common Path Skew : ([\d\.]+) ns

get\_par.0.regexp = from .+gclk\_cent2trunk to .+: cip cnt=0

get\_par.1.regexp = rc delay:.+?<(.+?)>

### 30.2 Description

1. File: specify file relative path or search pattern
2. Fixed keywords:"get\_twr.", "get\_par." and ".string", ".regexp"
3. MUST have the key get\_twr.0.string and get\_par.0.regexp
4. .string will be compared with the lines simply without space
5. .regexp will be matched by regular expression to get your data
6. Value trunk is the parameter to calculate par data, for example: raw data in par file is “65.600000, 90.200000, 65.600000, 90.200000”, par data is “(90.200000 - 65.600000) \* 5 / 1000 “
7. Twr data and par data will be used in a round a number to a given precision in 3 decimal digits.
8. If mode is more/less, par data should be larger/smaller than twr data, else should be equal.

### 30.3 Demo

[check\_clock\_skew\_<section\_num>]]

; KEYWORDS: "get\_twr.", "get\_par." and ".string", ".regexp"

file = ./\_scratch/impl/\*.twr

trunk = 5

mode = normal

get\_twr.0.string = Path Begin : temp4\_Z/Q (SLICE\_R5C1F)

get\_twr.1.string = Path End : Qout2\_Z/DF (SLICE\_R54C50E)

get\_twr.7.regexp = Setup Constraint :

get\_twr.8.regexp = Common Path Skew : ([\d\.]+) ns

get\_par.0.regexp = from .+gclk\_cent2trunk to .+: cip cnt=0

get\_par.1.regexp = rc delay:.+?<(.+?)>

# Unified Parameters

## 1 show

For every method, if set “show = 1”, it will show the result in the TMP web page, key checks column.

Example:

[check\_lines\_1]

…

show = 1 (default is 0)

## 2 cr\_note

cr\_note can apply to every individual section now. The cr will be checked unless cr\_status = 0 .

Example:

[check\_lines\_1]

…

cr\_note = DNG-xxx

cr\_status = 1 (default)