eda-log-file-warning-suppressor

Release 0.9.0

Sep 10, 2023

Contents:

CHAPTER 1

Overview

EDA Log File Warning Suppressor (ELFWS) provides warning suppression for EDA log files.

1.1 Why ELFWS?

ELFWS was created after trying to triage warnings in the Mentor Graphics Precision and Microsemi Designer tools. Precision had a built in method to suppress warnings, but only at the warning ID level. There was not enough granularity to allow suppression of individual warnings. Designer does not provide the ability to suppress warnings in their log file.

I like to run through the synthesis and place and route tools early in the design to discover any issues with IP. This worked well the first time I went through the design as I found an issue with two PLLs. The issue was fixed and design continued. When the design was almost completed, I triaged the warnings again and found something new. This required another change to one of the PLLs.

The new warning was buried with warnings I had previously reviewed. It was difficult to detect by scanning the file. It wasn't until I started grepping out those warnings I had seen before that I discovered the new one. If I was able to properly suppress the warnings I had reviewed before, the new issue would have easily been detected.

I also use Continuous Integration (CI) tools when designing to ensure design quality. Having to manually triage the warnings each time precludes the use of CI. If ELFWS had existed, the second issue with the PLL would have been detected much earlier in the design phase.

1.2 Key Benefits

- Provides a common method to suppress warnings
- Suppress warnings on supported EDA vendor tools
- Additional vendor tools can be added
- Reduce warning triage time

1.3 Key Features

- Command line tool
- Continuous Integration support
  - JUnit XML output of unsuppressed warnings
- Define suppression rules using YAML
- Reports for auditing suppressed warnings
  - Warnings not suppressed
  - Which warnings were suppressed by which suppression rule
  - Suppression rules which did not suppress any warnings
  - Warnings suppressed by multiple rules
- Operates on the log file
  - Do not need to re-run tool to validate warnings are suppressed

CHAPTER 2

Installation

There are two methods to install ELFWS.

2.1 PIP

The most recent released version is hosted on PyPI. It can be installed using **pip**.

```
pip install elfws
```

This is the preferred method for installing ELFWS.

2.2 Git Hub

The latest development version can be cloned from the git hub repo.

```
git clone https://github.com/jeremiah-c-leary/eda-log-file-warning-suppressor.git
```

Then install using the setup.py file.

```
python setup.py install
```

CHAPTER 3

Usage

ELFWS can be invoked using **elfws** at the command line prompt:

```
$ elfws
usage: elfws [-h] {create,report,show,suppress,version} ...

Suppresses Warnings in logfiles.

positional arguments:
  {create,report,show,suppress,version}
    create              Create suppression file
    report              Generate an audit report
    show                Show warnings in logfiles
    suppress            Suppresses warnings in logfiles
    version             Displays ELFWS version information

optional arguments:
  -h, --help            show this help message and exit
```

ELFWS has five subcommands: create, report, show, suppress and version.

3.1 create

Use the create subcommand to generate a suppression rule file from a given warning file.

This can be used as a starting point for a suppression file. Care should be taken as the output messages are not formatted to support regular expressions.

The arguments for the subcommand can be listed using the *-h* option:

```
$ elfws create -h
usage: elfws create [-h] [--suppression_file SUPPRESSION_FILE]
                    log_file output_suppression_file
```

(continues on next page)

(continued from previous page)

```
positional arguments:
  log_file              Log file with warnings to extract
  output_suppression_file
                        Suppression file to create

optional arguments:
  -h, --help            show this help message and exit
  --suppression_file SUPPRESSION_FILE
                        Existing suppression file to filter out existing
```

3.2 report

Use the report subcommand to generate detailed output of suppression warnings.

The report will show the following information:

- Unsuppressed warnings
- Which suppression rules suppressed which warnings
- Unused suppression rules
- Warnings that were suppressed by multiple suppression rules
- Summary of suppression rules and warnings

The report can be used during reviews to ensure the suppressions are valid.

This command has the option argument **–junit**, which will output a JUnit XML file. This file can be used with continuous integration tools to check for new warnings.

The arguments for the subcommand can be listed using the *-h* option:

```
$ elfws report -h
usage: elfws report [-h] [--junit JUNIT] log_file suppression_file report_file

positional arguments:
  log_file          Log file to check for warnings
  suppression_file  YAML formatted warning suppression file
  report_file       Output report file

optional arguments:
  -h, --help        show this help message and exit
  --junit JUNIT     Generate JUnit XML file JUNIT
```

3.3 show

Use the show subcommand to list all the warnings in a logfile.

This can be useful when first starting out suppressing warnings and a suppression rule file does not exist.

The arguments for the subcommand can be listed using the *-h* option:

```
$ elfws show -h

usage: elfws show [-h] log_file

positional arguments:
  log_file    Log file to show warnings

optional arguments:
  -h, --help  show this help message and exit
```

3.4 suppress

Use the suppress subcommand to suppress warnings in a logfile.

This can be useful when creating a suppression rule file. It reports the results to the screen and only shows warnings which have not been suppressed.

The arguments for the subcommand can be listed using the *-h* option:

```
$ elfws suppress -h

usage: elfws suppress [-h] log_file suppression_file

positional arguments:
  log_file          Log file to check for warnings
  suppression_file  YAML formatted warning suppression file

optional arguments:
  -h, --help        show this help message and exit
```

3.5 version

Use the version subcommand to report the installed version of ELFWS.

There are no arguments for this subcommand.

```
$ elfws version

EDA Log File Warning Suppressor (ELFWS) version 1.0.0
```

CHAPTER 4

Evaluating Warnings

There is a natural process for evaluating warnings. The process follows the diagram below:

```
warning -> evaluate -+-> suppress
                     |
                     +-> mitigate
                     |
                     +-> investigate -+-> suppress
                                      |
                                      +-> mitigate
```

4.1 Evaluate

During the evaluate phase, the warning is reviewed. One of three classifications will be applied: suppress, mitigate or investigate.

4.2 Suppress

If the warning can be safely ignored, then it can be suppressed.

A justification for suppressing should be given in the **comment** field of the suppression rule. Giving the justification documents the reasoning behind suppressing. This communicates the intent for other users and as a reminder to the author why the warning was suppressed.

The name, or initials, of the person doing the classification should be given in the **author** field of the suppression rule. This indicates to other users who to contact if there are questions about the suppression rule.

4.3 Mitigate

The warning could point to a real issue in the design. Addressing the issue will lead to the warning no longer being reported by the tool.

4.4 Investigate

If the warning requires a significant amount of time to evaluate, it falls into the investigate classification. This can happen for any of the following reasons:

- warnings in vendor IP
- warnings in a part of code base you are unfamiliar with
- warnings requiring someone else to evaluate

Each warning which requires investigation should be tracked. This can be done with a spreadsheet or a dedicated issue tracking system.

Depending on the result of the investigation, the warning will either be suppressed or must be mitigated.

In ELFWS these types of warnings will be suppressed, but can be tagged with an **investigate** field in the suppression rule.

Use the **comment** field to document questions about the warning to be investigated. The tracking ID can be added to the **comment** field.

If the warning under investigation is mitigated, then the suppression rule should be removed from the suppression file. If the warning under investigation is suppressed, then remove the **investigate** field and update the comment to indicate why the warning can be suppressed.

CHAPTER 5

Suppression Rules

The suppression rules are a YAML formatted file with the following basic form:

```
suppress:
  rules:
    <warning_id>:
      - msg: Port I_CLK_A not used
        comment: This port is not used in this design.
        author: jcleary
      - msg: Port I_DATA_A not used'
        comment: This port is not used in this design.
        author: jcleary
        investigate : True
    <warning_id>:
      - msg: Signal fifo_enable is tied high
        comment: The FIFO is always enabled in this design to support data throughput.
        author: jcleary
        options:
          - suppress_in_json_if_unmatched
```

It starts with the **suppress** key and then a **rules** key. The **rules** key contains one or more suppression rules. Each **rules** key is further divided into one or more warning ids.

5.1 Warning IDs

Regular expressons are support in warning ids. This allows for another method of grouping suppressions. In the example below, every ID that starts with **Synth-** and has **flip-flop** in the message will be suppressed.

```
suppress:
  rules:
    Synth-.*:
      - msg: flip-flop
```

5.2 Suppression Rule Fields

Each suppression rule will have the following fields available:

Option	Required	Description
msg	Yes	The message to suppress. This will be a regular a regular expression which will match after the defined message ID. LSW will prepend a .* to this field.
comment	No	While optional, it is strongly recommened to use this field to document why the warning was suppressed.
author	No	Document who created the suppression.
investigate	No	Boolean(True/False) indicating the warnings the rule suppresses need further analysis. This defaults to False.
options	No	A list of options applied to the suppression.

In addition to the standard warning ID, each tool may have warnings without IDs. When processing these warnings, ELFWS will use a warning ID of **no_id**.

```
suppress:
  no_id:
    msg: Unused port
    comment: This port is not used.
    author: mpw
  <warning_id>:
    msg: Port I_CLK not used
    comment: This port is not used.
    author: jcl
```

If these unique IDs exist, they are listed in the tool section of this documentation.

5.3 Grouping Rules

It can be helpful to group rules based on some criteria. For example, file names, sections in a log file, design elements or warning types.

```
suppress:
  <group id>:
    rules:
      <warning_id>:
        - msg: FIFO uses same clock for read and write
```

Arbitrary levels of groupings are also supported:

```
suppress:
  <group id>:
    rules:
      <warning_id>:
        - msg: FIFO uses same clock for read and write
  <group id>:
    <group id>:
      rules:
        <warning_id>:
```

(continues on next page)

(continued from previous page)

```
        - msg: RAM address bits [12:3] are unused
  <group id>:
    rules:
      <warning_id>:
        - msg: invalid false path
  rules:
    <warning_id>:
      - msg: UART is blackboxed
rules:
  <warning_id>:
    - msg:  signal fifo_wr is tied high
```

Dividing suppression rules into groups helps with maintaining the suppress rules. ELFWS flattens all the suppression rules into a single list.

5.4 Options

Options change the behavior of a suppression.

The following table includes all avialable options:

Option	Description
suppress_in_json_if_unmatched	The suppression will be suppressed in the json xml file only.

CHAPTER 6

Theory of Operation

ELFWS performs the following tasks:

1. Read suppression rule file
2. Read logfile
3. Determine vendor
4. Load warning extractor
5. Extract warnings from logfile
6. Compare suppression rules against extracted warnings a. Mark which rule suppressed a warning
7. Report results

6.1 Read suppression rule file

This reads the YAML file which will create a dictionary.

6.2 Read logfile

This reads the text logfile and stores it as a list of strings.

6.3 Determine vendor

The log file will be interrogated to determine the vendor log file.

6.4 Load warning extractor

There are different patterns of warnings depending on the vendor and tool. To support multiple styles, each vendor and tool will require it's own warning extractor.

6.5 Extract warnings from logfile

The logfile is parsed using the loaded warning extractor. Any warning discovered will be stored in a list for further analysis.

6.6 Compare suppression rules against extracted warnings

Each warning will be compared against every suppression rule. If a match is found then that warning will be stored with the rule. If not match is found then that warning will be stored in an unmatched list.

6.7 Report results

The format of the reporting results will depend on the subcommand chosen. Refer to the subcommand for further details.

CHAPTER 7

Vendors

The following sections contain information about each vendor tool supported by ELFWS.

7.1 Mentor Graphics

7.1.1 Precision

Precision has two warning formats: those with IDs and those without.

Warnings With IDs

Warnings with IDs are identified with the Warning keyword and the ID between colons inside square brackets.

<warning keyword>:[<ID>]: "<filename>", <linenumber>: <module>: <message>

where:

Item	Regular Expression Match
warning keyword	^# Warning
ID	[0-9]+
filename	\W+
linenumber	line\s[0-9]+
module	Module\s\W+
message	.*$

Warnings Without IDs

Warnings without IDs are identified with the Warning keyword without the ID in square brackets. The message is considered to be everything after the first colon.

<warning keyword> : <message1> : <message2>

where:

Item	Regular Expression Match
warning keyword	^Warning
message1	.*
message2	.*$

Extracting Warnings

The fields filename, linenumber, module and message will be combined into a single message.

1. Search for lines starting with **# Warning:**
2. Extract string from between colons
3. Classify warning
    a. As ID if there are no spaces within the extracted string
    b. As NO_ID if the there are spaces within the extracted string
4. Save the message
    a. Everything after the second colon if the message has an ID
    b. Everything after the first colon if the message does not have an ID

7.1.2 Quasta Lint

Questa Lint has a single format for errors, warnings and infos.

<ID>: <message>, Module '<module>', File '<filename>', Line '<linenumber>'

where:

Item	Regular Expression Match
ID	\W+
filename	\W+
linenumber	\s[0-9]+
module	\W+
message	.*$

Extracting Warnings

The fields filename, linenumber, module and message will be combined into a single message.

1. Search for line starting with 'Section 2' before processing warnings
2. Search for lines starting with 'Check:'
    a. extract ID
3. Search for lines starting with extracted ID
    a. create warning
4. Search for a line starting with '| Info'
    a. Stop searching for

Quasta CDC

Questa CDC provides several reports which can be parsed.

cdc_run.log

The cdc_run.log file reports violations, cautions, and evaluations in addition to run time errors and warnings.

CDC Results

This section reports the CDC crossings and whether they are Violations, Cautions, and Evaluations. The reporting is divided into reporting the number of types of crossings and then the details of the crossing. Only the number of types of crossings will be reported for Violations, Cautions, and Evaluations.

Message Summary

This section reports errors and warnings encountered while running the tool. There should not be any Errors or Warnings in this section.

The cdc_detail.rpt file provides the most items to be checked.

Section 1 : Clock Information

The number of inferred clocks should be 0. The following line will be checked:

2. Inferred :(0)

If the number in the parenthesis is not 0, then a warning will be reported.

Section 2 : Reset Information

The number of inferred resets should be 0. The following line will be checked:

2. Inferred :(0)

If the number in the parenthesis is not 0, then a warning will be reported.

Section 3 : CDC Results

This section reports the CDC crossings and whether they are Violations, Cautions, and Evaluations. The reporting is divided into reporting the number of types of crossings and then the details of the crossing. Only the number of types of crossings will be reported for Violations, Cautions, and Evaluations.

Section 9 : Design Information

This section reports information about the design. The number of empty modules and unresolved modules should be 0 to ensure a proper analysis.

The following lines will be checked:

Number of Empty Modules = 0 Number of Unresolved Modules = 0

If the number after the equal sign is not 0, then a warning will be reported.

Section 10 : Port Domain Information

This section reports each port and the clock assigned to it. It also reports whether the user defined the clock domain or if QuestaCDC assigned it.

Each line follows this format:

<Port> <Direction> <Constraints> {<Clock Domain>} <Type>

If <Type> is not "User" then an warning will be reported.

7.2 Microsemi

7.2.1 Designer

Designer has two different warning formats: those with ID's and those without. Warning messages can also span multiple lines.

Warnings With IDs

Warnings with IDs are identified with the Warning keyword and the ID between colons. The message is after the second colon.

<warning keyword> : <ID> : <message>

where:

Item	Regular Expression Match
warning keyword	^Warning
ID	\W+
message	.*$

Warnings Without IDs

Warnings without IDs are identified with the Warning keyword and colon. The message is after the colon.

<warning keyword> : <message>

where:

Item	Regular Expression Match
warning keyword	^Warning
message	.*$

Multiline Warnings

Multiline warnings can span any number of lines. They are identified with at least on space at the beginning of the line for each line after the initial warning line.

<warning keyword> : <ID> : <message1> <warning continuation><message2> <warning continuation><message3>

where:

Item	Regular Expression Match
warning keyword	^Warning
ID	\W+
message1	.*$
warning continuation	^s+
message2	.*$
message3	.*$

..or

<warning keyword> : <message1> <warning continuation><message2> <warning continuation><message3>

where:

Item	Regular Expression Match
warning keyword	^Warning
message1	.*$
warning continuation	^s+
message2	.*$
message3	.*$

Extracting Warnings

Extraction of warnings from the logfile will follow this process:

1. Search for lines starting with **Warning**
2. Classify warning

a. As ID if ID pattern matches

b. As no_id if ID pattern does not match

3. Check successive lines for line beginning with spaces

a. append line to existing message

7.3 Xilinx

7.3.1 Vivado

Vivado has a single warning format, but two different type of warnings: critical warnings and non critical warnings. Some warning messages can also span multiple lines.

Critical Warnings

Warnings with IDs are identified with the Warning keyword and the ID between colons. The message is after the second colon.

<warning keyword>: [<ID>] <message>

where:

Item	Regular Expression Match
warning keyword	^CRITICAL WARNING
ID	\W+sW+
message	.*$

Non Critical Warnings

Non critical warnings are identical to critical warnings except the keyword.

<warning keyword>: [<ID>] <message>

where:

Item	Regular Expression Match
warning keyword	^WARNING
ID	\W+sW+
message	.*$

Multiline Warnings

Multiline warnings can span any number of lines. They are identified with at least on space at the beginning of the line for each line after the initial warning line.

<warning keyword>: [<ID>] <message1> <warning continuation><message2> <warning continuation><message3>

where:

Item	Regular Expression Match
warning keyword	^[WARNING|CRITICAL WARNING]
ID	\W+sW+
message1	.*$
warning continuation	^s+
message2	.*$
message3	.*$

Extracting Warnings

Extraction of warnings from the logfile will follow this process:

1. Search for lines starting with **WARNING** or **CRITICAL WARNING**
2. Extract ID and message
3. Check successive lines for line beginning with spaces

a. append line to existing message

CHAPTER 8

Adding Vendor Tools

To add a new vendor tool, you must understand the ELFWS vendor directory structure.

8.1 Directory Structure

ELFWS uses the following directory structure for vendors and their tools:

```
elfws
 |
 +-- vendor
      |
      +-- <vendor name>
           |
           +-- <tool_name>.py
```

For example, for our example the tools Microsemi Designer and Mentor Graphics Precision are in this directory structure:

```
elfws
 |
 +-- vendor
      |
      +-- mentor_graphics
      |    |
      |    +-- precision.py
      |
      +-- microsemi
           |
           +-- designer.py
```

The vendor directories and tool files expand as they are added:

```
elfws
 |
 +-- vendor
      |
      +-- mentor_graphics
      |    |
      |    +-- precision.py
      |    +-- questa_sim.py
      |
      +-- microsemi
      |    |
      |    +-- designer.py
      |
      +-- synopsys
           |
           +-- synplify_pro.py
           +-- design_compiler.py
```

ELFWS will search the elfws->vendor directory for all directories. It will then search each of those directories for tool files. ELFWS will pass the log file to each tool file and ask if the log file is from that tool. If the tool file does not recognize the log file, then ELFWS moves to the next tool. If the tool file does recognize the log file, then ELFWS uses the extract_warnings function to parse out the warnings in the file.

CHAPTER 9

Vendor Tool File

The Vendor Tool file performs the initial parsing of warnings from the log file. It must contain the following functions:

- get_vendor()
- get_tool_name()
- is_logfile()
- extract_warnings()

9.1 imports

To support the *extract_warnings* function, the following imports must be included:

```
from elfws import warning
from elfws import warning_list
```

9.2 get_vendor

This function just returns a list of strings listing the vendor. A list was choosen to manage company acquisitions. The most recent company name should be first in the list.

```
def get_vendor():
    return ['Microsemi', 'Actel']
```

9.3 get_tool_name

This function returns a string with the name of the tool.

```
def get_tool_name():
    return 'designer'
```

9.4 is_logfile

This function is responsible for parsing the logfile and determining whether it is a logfile for the tool. There are typically some unique strings in the beginning of the logfile that identifies which tool generated it. The function must return a boolean.

```
def is_logfile(lFile):
     for iLineNumber, sLine in enumerate(lFile):
         if sLine.startswith('Microsemi Libero Software'):
             return True
         if iLineNumber == 10:
             return False
     return False
```

9.5 extract_warnings

This function parses the logfile for warnings. It returns a warning_list object with a collection of warning objects.

The following code looks for lines starting with *Warning* and then proceeds to handle warnings without IDs and multiline warnings.

```
def extract_warnings(lFile):
    oReturn = warning_list.create()

    fWarningFound = False
    for iLineNumber, sLine in enumerate(lFile):
        # Clear the warning found flag
        if not sLine.startswith(' ') and fWarningFound:
            fWarningFound = False
            oReturn.add_warning(oWarning)
        if sLine.startswith(' ') and fWarningFound:
            oWarning.message += ' ' + sLine.strip()
        if sLine.startswith('Warning:'):
            fWarningFound = True
            iColon1Index = sLine.find(':')
            iColon2Index = sLine.find(':', iColon1Index+1)
            if iColon2Index == -1:
                sID = 'NO_ID'
                sMessage = sLine[iColon1Index+1:].strip()
            else:
                sID = sLine[iColon1Index+1:iColon2Index].strip()
                sMessage = sLine[iColon2Index+1:].strip()
                if ' ' in sID:
                    sID = 'NO_ID'
                    sMessage = sLine[iColon1Index+1:].strip()
            oWarning = warning.create(sID, sMessage, None, iLineNumber + 1)
    return oReturn
```

Note: Use existing functions from other vendor tools as a basis to generate new ones.

CHAPTER 10

Contributing

I welcome any contributions to this project.

There are several ways to contribute:

1. Bug reports
2. Code base improvements
3. Feature requests
4. New vendor warning suppressions

10.1 Bug Reports

If you run into anything that is not handled correctly, please submit an issue. When creating the issue, use the **bug** label to highlight it. Fixing bugs is prioritized over feature enhancements.

10.2 Code Base Improvements

My Python journey is never ending and I learn new things with each project. I run the code through *Codacy* and *Code Climate*, and they are very helpful. However, I would appreciate any suggestions to improve the code base.

Create an issue and use the **refactor** label for any code which could be improved.

10.3 Feature Requests

Let me know if there is anything I could add to make ELFWS easier to use. Create an issue with the **enhancement** label.

10.4 New Vendor Warning Suppresssions

I plan to update ELFWS as I run into tools where I need to suppress warnings. If there is a tool that is currently not supported, then create an issue with the **Vendor Tool** label.

Provide a small sample showing the format of the warnings and I can add it or show you how to add it.

CHAPTER 11

Indices and tables

- genindex
- modindex
- search