An efficient text mining application for log file analysis in an emulation environment using Tcl/Tk with C

Mishra, Shyam
Mentor Emulation Division,
Mentor Graphics Corp.

Abstract

Text mining refers to the process of deriving high quality information from text files. Hardware emulation is the preferred way for verification of multi-million gates SOC designs. Text mining can be applied for log file analysis of huge log files that get generated in an emulation based design verification flow. Typically an emulation based verification flow consists of two discrete steps, namely compile and runtime. During the compile stage, a HDL design is prepared for emulation. The compile tools generate log files and other reports. The emulation based verification flow is used typically for largest of design databases, and the mapping to hardware involves multiple complex compilation steps. This makes it imperative to have intelligent debug systems with advanced data mining capabilities. Text mining is applied to extract useful information from these log files and reports in order to help the user detect errors and warnings in compile that might affect the emulation. Logs and reports generated during emulation runtime are also similarly analyzed.

Using Tcl/Tk , a GUI is developed to use text mining methods on very large emulation databases for log file analysis. Main considerations for design for such text mining application has been that interactive user response remains fast, the parent Emulation control and Debug GUI is able to interact and work with the text mining widget with fast response time, in unblocking manner, and with minimal overhead to the parent Emulation control and Debug GUI. Besides design ensures search operations are fast, the application memory image is low, and the application provides host of ease of debug utilities like GUI based linkages to user RTL source, informative help from the messages in log files. To achieve this intelligent partitioning of functionalities between C and TCL code is done. The application makes use of a C/C++ based shared object for efficient retrieval of information from the huge log files generated by the emulation tools. The application GUI makes use of the latest Tcl/Tk features to provide an easy to use interface to give the users a rich debugging experience.

1. INTRODUCTION

Hardware emulation is the preferred way for verification of the next generation multi-million gates SOC designs. In a typical emulation flow, the user design is compiled and prepared for configuration on the emulator hardware.
In the process, the user code which consists of RTL and transactor level models is compiled by a set of compilers to generate the model which can be configured on the emulator.
The compile flow is quite complex .The error, warning and other messages generated by the compilers provide important information to the user, which can help understand the changes or modifications required in the user code in order to perform the emulation. During design emulation at runtime also advanced debug and log file analysis capabilities are required to understand any functional mismatches. Often the clues to a bad design behavior at runtime or a compile failure are hidden in the log files and the reports generated by the tools.
The user can manually check the log files and the reports to locate the cause of such failures.
However, manually browsing through a huge database, locating all the log files generated by the different compile and runtime tools and checking the information present therein can be time consuming. Besides, the user might be unable to

locate the relevant information.
Therefore text mining methods are applied to allow the user access the relevant information from the log files and the reports without losing precious time.

Text mining refers to the process of deriving high quality information from text. Text mining usually involves the process of structuring the input text (usually parsing, along with the addition of some derived linguistic features and the removal of others, and subsequent insertion into a database), deriving patterns within the structured data, and finally evaluation and interpretation of the output. 'High quality' in text mining usually refers to some combination of relevance, novelty, and interestingness. Typical text mining tasks include text categorization, text clustering, concept/ entity extraction, production of granular taxonomies, sentiment analysis, document summarization, and entity relation modeling (*i.e.*, learning relations between named entities).
(Reference : http://en.wikipedia.org/wiki/Concept_mining)

In the following sections we will discuss how an efficient text mining tool was developed using Tcl/ Tk 8.5 with C.
The text mining tasks which are computationally intensive are implemented in C. The Tcl/Tk makes calls to the C functions as and when required. Display is managed entirely by Tcl/Tk side.

2. C based library for text mining

A C based database manager is developed to store the information related to the tools and the corresponding log file paths.
C functions are implemented to access interesting information from the log files.
Those functions efficiently extract the requested information from the log files and provide it to the caller code.
The C functions are embedded inside a shared library which registers Tcl commands on a Tcl interpreter. The Tcl commands can be called from any Tcl/Tk based GUI that loads this shared library. Internally, those Tcl commands are mapped to the C functions.
Searching through the large database of log files can be time consuming, so C is preferred over Tcl.
Besides, C can be used to implement an efficient parser that parses the log files on demand to retrieve the requested information for the user.

The command interface between the Tcl/Tk GUI and the C shared library is designed to be backward compatible. Thus, the GUI can be modified without requiring a recompile of the shared library and conversely, the shared library can change the implementation of its parser and search functions without necessary build of the GUI, as long as the interface is maintained intact.
Assuming that the C library is named "libloganalyze.so", the Tk gui makes the following call:

load <path to libloganalyze.so>
The load call passes the Tcl interpreter handle to the C library. Commands are created on this interpreter for use by the subsequent GUI queries.

3. GUI display of tools and log files: text categorization

The GUI is designed to have a tree view for the tools and log files hierarchy.
For example, a hierarchy looks like this : tool → log files → messages. Under a tool such as "HDL compiler" , there could be logs such as "hdl_compile.log", "hdl_compile.report". Further, the hdl_compile.log node can be expanded to display the "Errors", "Warnings", "Note", "Status" and other categories of messages.
The ttk treeview widget is used for this purpose. It provides the user a convenient way to view the various messages occurring for the different tools in a single window.
Whenever the user expands a node of the tree, a query is generated for the C shared library. The query is executed in C code and the relevant information is fetched by the GUI.
GUI side : Treeview->Expand (node)
Calls C function : loganalyze -get_child_nodes –queryString <queryString>
GUI gets the results of the C call and changes tree display / log file view as applicable.

For example, if the user expands a tool node, then the result of the C function call will return the log file names associated with the tool.
Similarly, for an individual log file node, the C function will return the message types as the child nodes and also the text to display as the contents of

the given log file in the text view widget.
The text mining operation is carried out in the C function and the results are displayed in the Tcl/Tk GUI.

4. Search results display using text clustering

The text display clusters messages of a particular type based upon the type. For example, the warnings are displayed clustered together, as are the other message types.
If the search is based upon some pattern, the pattern is highlighted in the search results.
For example, if the user searches for "simulation mismatch", the clause "simulation mismatch" will be highlighted in the search results displayed in the text view.
For example :
Warning [100] : Net top.a has been removed from the design.
Warning [100] : Net top.b has been removed from the design.
……
SimWarning [200] : Net top.c has multiple drivers, this may cause a simulation mismatch.
SimWarning [200] : Net top.inst.q has multiple drivers, this may cause a simulation mismatch.
……

5. Concept / entity extraction

To display file names, line numbers and net names in the text view, the file names are extracted and displayed with hyperlink tags. The hyperlink is programmed to open the corresponding file and line number in an editor such as vi or emacs, as specified by the user , upon right mouse button click.
For example a message could look like this :
Warning [101] : File *design.v*, line 11, syntax error near "=".
In the above message, the file path "design.v" will be hyperlinked.

Code snippet :

```
set textWidget $mainWidget.logFileDisplay
   $textWidget tag  configure hyperlink -foreground royalblue  -underline true
   $textWidget tag bind hyperlink <Double-Button-1> { clickALink %x %y %W}
      $textWidget tag bind hyperlink <Return> {clickALink %x %y %W}
```

Search for all the file names and tag those as hyperlink.

```
proc  clickALink {x y w} {
set i [$w index @$x,$y]
 set range [ $w tag prevrange hyperlink $i]
 set url [eval $w get $range]
sourceViewFile  $url
}
```

The procedure sourceViewFile opens the specified url in the editor selected from the user environment.

A separate display canvas is provided for the design statistics , such as the design size, compile status of the tools and various performance / capacity related metrics.
This information is obtained via a call to the C library at start up.
GUI call : loganalyze –get_design_stats
C function : loganalyzer->GetDesignStats().
Returns the design stats after mining the log and reports database.
During startup, a list of predefined phrases is also searched in the database and those are displayed in a different view as the "Analysis Report".
The analysis report allows the user to browse to the relevant phrase in the log files spread across the emulation database using hyperlinks.

GUI side : loganalyze –queryString <get statistics for important messages>
C side : loganalyzer->GenerateReport()
Returns the statistics for the important messages in all the log files and reports.
This call returns the statistics of all the important messages in which the user might be interested, right at the start up.

6. GUI architecture for multiple views

The three log file related views : namely the text view, the design statistics canvas and the analysis report view are implemented as tabs in a ttk notebook widget.
The text display changes for each and every text file, so the text view tab has sub-tabs for each and every log file that is opened for search.

Using some customization using ttk::style, the sub tabs are provided with a X icon at the right top corner to allow the user to close the view for a

particular log file.

Code snippet :[Ref : wiki.tcl.tk]

```
image create photo  closeImage -file $::closex.gif
ttk::style element create ButtonNotebook.close image closeImage
ttk::style layout ButtonNotebook {
    ButtonNotebook.client -sticky nswe
}
 ttk::style layout ButtonNotebook.Tab {
    ButtonNotebook.tab -sticky nswe -children {
        ButtonNotebook.padding -side top -sticky nswe -children {
       ButtonNotebook.focus -side top -sticky nswe -children {
     ButtonNotebook.close -side right  -sticky n
     ButtonNotebook.label -side left -sticky {}
      }
     }
   }
}
```

It can be reopened later on if required, using the appropriate node in the tree view.

7. Query generation interface

The log file analyzer GUI provides a versatile query editor. The user can select the type(s) of message(s) to display and can specify the scope of the search in terms of the tools or the log files.

The user is also allowed to input text patterns for search including regular expressions. Search is possible with and without case sensitivity. The query editor is implemented using check buttons and text entry fields.

Code snippet (query creation)

```
proc  CreateQueryString {} {
  Get all  check button status
  Get search entry
  Get  regular expression or not
  Get case sensitive or not
   Create a query string for loganalyze command call.
}
```

The user can also use the tree widget to specify the scope of the search.

GUI side : loganalyze –queryString <queryString>
C function : loganalyzer->Search(queryString)
Returns the search results for the specific query.

The search results are displayed in a categorized form in the log file text view tab which is embedded in the ttk notebook widget.

8. Sentiment analysis : comparative analysis of log files

Often the user likes to compare the number of warnings generated in the current compile with the numbers generated in a prior compile of the same design.

For this purpose, the tool allows the user to save a given set of log files in a compact form. After re-compiling the design, the user can load the older set of log files and do a comparative analysis based upon the types and contents of the messages generated in both the older and the newer compile sessions.

This allows the user to check whether the number of warnings has increased or reduced, whether the area requirements have changed and whether or not a better performance can be expected from the new compile. It also allows the user to know if new bugs have crept into the design in compile flow, possibly leading to erroneous behavior later, during emulation run.

GUI side : loganalyze –compare <project 1> <project 2> -queryString <query string>
Returns the results for the comparison to GUI.
Display categorization is managed by the GUI.

9. Online help system

For the log file analyzer to be useful, it must not only display the relevant messages or search results , but should also provide some tips to the user for the various errors or warning messages.

The log file analyzer extracts the message mnemonic or id and searches the available documents and web resources for relevant help. The user can make use of this online help functionality to understand the cause of an error or a warning or just the significance of a status message.

GUI side : loganalyze –help <search phrase>
C function: loganalyzer->HelpDatabaseQuery(searchPhrase)
Returns the help string for display in the GUI.

10. Design debug using the parent emulation debug gui

The log file analyzer GUI maintains a socket based connection with the parent emulation debug GUI.
Through this connection, the extracted name of a signal or a module can be passed to the GUI, where it can be browsed in the design path viewer.
Thus the user can understand the reason for a typical warning message such as "Net is dead logic" or "Net has multiple drivers" by browsing the design in the emulation debug gui.
The sequence of actions done by the user would look like this :

a) Search for "multiple drivers" in compiler logs.
b) Results are displayed categorized in the text view.
c) Visit any particular interesting message and click on the hyperlink for the net name.
d) The net name is displayed in the emulation gui path viewer.

Similarly, the log file analyzer allows the user to view the waveforms for an interesting net where those are available with the emulation debug gui.
GUI side : "Send Parent GUI command : Add net to path viewer".
Parent GUI : Receives and parses the command and calls appropriate command : "add pathviewer <net name>".

11. Summarization of area and performance reports

Area reports are generated at compile time. Performance reports are generated at runtime.
The log file analyzer can display the modules that consume the most of the design area. The user can focus on the relevant modules and remodel the HDL code to optimize the area requirements.
The number of simulation cycles consumed , the design frequency, the number of transaction calls made and the time taken are available in performance reports.
A summarized display of those allows the user to optimize the test bench and the design quickly without having to browse through the reports manually and undertaking the effort to interpret them.

12. Cost , limitations and future work

The text mining techniques applied here make use of the standard messaging format used by the emulation tools.
In case there are third party tools which generate huge log files in an unknown format, the log file analyzer is not able to apply text mining techniques for those.
The current implementation can be made more intelligent to accept a user defined messaging format to analyze any log file database generated by any product.
If the log files are very large in number, the volume of information extracted can be quite huge. In such cases, the user has to do selective searches and not go for generic pattern searches which could become time consuming.

13. Conclusion

A text mining tool using Tcl/Tk and C for emulation databases has been described here.
It makes use of text mining techniques such as text categorization, entity/concept extraction, text clustering, document summarization and sentiment analysis for analysis of log files and reports.
It makes use of the efficiency of C to quickly analyze and retrieve useful information from the emulation database. A Tcl/TK 8.5 based GUI interfaces with the C shared library to provide a rich and interesting debugging experience to the emulation users.
The concept can be extended in the future to any system where the log files are generated in a predefined messaging format and the debug functionality can be made configurable for the relevant system.

REFERENCES:

[1] http://en.wikipedia.org/wiki/Text_mining
[2] wiki.tcl.tk