Building an Electronic Design Automation (EDA) tool using Tcl/Tk and object oriented programming.

John Hughes
Software Engineer
Mentor Graphics Corp.
Wilsonville, OR

14th Annual Tcl/Tk Conference
New Orleans, Louisiana
September, 2007

Abstract

This paper discusses how a powerful debugging, and design analysis, tool was created for the Design-For-Test (DFT) division at Mentor Graphics Corp. This tool is called DFTVisualizer and is a quantum leap forward from the old tool. The old tool, known as DFTInsight, will be covered briefly for comparison purposes.

This paper will describe the reasons why Tcl/Tk was chosen and how the Tcl code interacts with the C/C++ "kernel" code. It will also cover the object oriented techniques used to build the tool and the various packages used in the implementation.

Introduction & Outline

Several years ago we were presented with the problem of replacing our current debugging tool with something new and exciting. The old tool was basically a schematic viewer that would allow the user to view portions of their chip design that were flagged as having design rule errors.

After nearly a year of meetings and gathering requirements we had an “idea” of the tool we needed to build. It needed to encompass the capability of the old tool, but offer several additional debugging and analysis features. Many of those features already existed in the system, but in the form of textual reports. While those reports contained the desired information, they were often labor intensive to read and “digest”. Showing the data in a graphical form was a requirement.

The remainder of this paper will be organized as follows:

1. A brief history of the legacy DFTInsight tool and textual reports.
2. Requirements and the functional specification.
3. Language and widget toolkit considerations.
4. Design of the DFTVisualizer tool.
5. Conclusion with a brief description of the implementation status and its success or failure with our customers.

History

The Design-for-Test tools aid in creating test patterns to run on the Automatic Test Equipment (ATE) in the “fab” when chips are being produced. Some DFT tools also modify the chip design to contain extra circuitry, so the chip can test itself. This is called Built-in-Self-Test or BIST for short. Creating test patterns for the latest microprocessor is a task that is very time and memory consuming. This can take days, even using grid computing engines, so the user wants the debugging process to be intuitive, effective, and reduce the overall time spent debugging the design. The Mentor Graphics DFT products have utilized a tool called DFTInsight to examine the design via a schematic that represents a portion of the chip design containing a problem.

See figure 1.

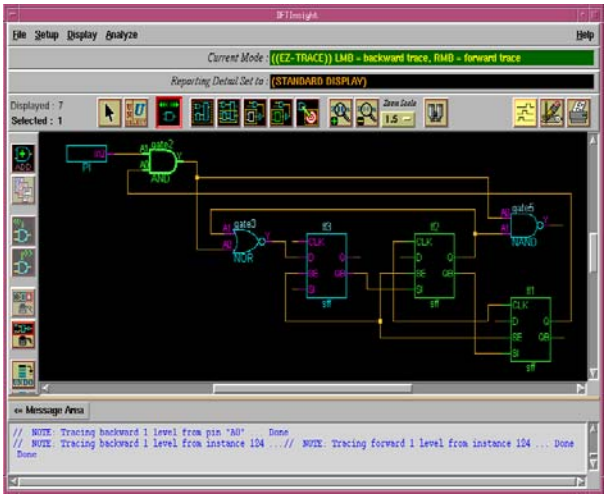

Figure 1. DFTInsight showing a DRC violation

Trouble shooting a DRC violation is only one of the problems that DFT customers encounter. There are other issues they need to analyze, like test coverage. Test coverage reports will tell the user what areas of the chip are testable and have test patterns generated for it. The user wants to see the coverage statistics for the entire chip and for the different design blocks used to build the chip. That way they can concentrate their efforts on the areas with the most problems. Traditionally this has been a textual report from the DFT products that the customer has to read. Often the customer would create scripts to produce more readable reports or convert them into CSV for importing into a spreadsheet program.

The next generation tool would evolve from strictly a debugging tool for design rule violations, to one that would allow graphical analysis of many textual reports. An example text report is shown below.

<ATPG> report statistics -hierarchy
... analyzing hierarchical statistics (Coll: 8248 faults, Full: 13282)

	# faults	UO	AU	% test
-top- (cm)	13282	0	34	16.07%
dir (txStatus)	176	0	0	23.86%
cfsm (cmFSM)	1138	0	0	11.51%
dec (decode)]	1142	0	22	17.81%
skipgen (skip_generate)	394	0	0	19.54%
lbmux (mux2to1b20)	86	0	0	0.00%
hpdech (hp_decode)	214	0	0	27.10%
hpdecl (hp_decode)	212	0	0	26.42%
dcx (detect_cx)	178	0	0	0.00%
dd (detect_data)	18	0	0	0.00%

Requirements and Functional Spec.

For approximately a year, a team met weekly to gather and discuss requirements for the new debug and analysis tool. This team was comprised of people from technical marketing, customer support, technical documentation, QA, and engineering.

Once the requirements were finalized (theoretically), work began on a functional specification document. This was largely authored by the engineering team and approved by customer support and technical marketing personnel. This document broke down each requirement into greater detail and how it would work in the new tool. One of the requirements was that there would be multiple windows, or views, of the data and those windows would be contained within a framework or parent window. These windows could be un-docked from the parent window and re-docked at a later time.

Another requirement is that this new environment needed to work with our legacy GUI and with our tool in no-GUI mode. The DFT tools can be run without a GUI in interactive mode or batch mode. In interactive mode, the user still needs to be able to bring up the new debugging and analysis environment by typing a command. This means that our command loop needs to work in conjunction with the TK event loop and not block each other. The DFT tools are single threaded applications, but do use grid distribution to run multiple processes. So, the master process is single threaded (legacy issue), but it spawns multiple slave processes on the grid.

Lastly, all of this needs to work in a Unix/Linux world because that is where the DFT tools are utilized.

Language & widget toolkit considerations

Several languages and widget toolkits were considered before coding started on the new tool. We knew that some of the code would be coded in C/C++ because that is what the “kernel” of our DFT products is written in. The DFTInsight tool, and most of our GUI, was written in Tcl/Tk, but we did consider the following options:

- Tcl/Tk
- Java
- Qt
- incrTcl/incrTk
- SWidgets (Mentor toolkit)
- mtiWidgets (Mentor toolkit)

incrTcl/incrTk with a C interface to the kernel code was ultimately selected and we also used the mtiWidgets toolkit. The reason that the mtiWidgets were selected is that they were already used in a popular Mentor product whose customers liked the GUI, and they were created with incrTcl/incrTk. Also, since we would be switching back-and-forth between C++ and Tcl code, incrTcl just seemed like a natural choice to do object oriented tasks in Tcl.

Design of the DFTVisualizer

The first thing we did once we started the design phase was to write a design document that detailed the different parts of the tool being built. The document contained the details of the data structures, classes, and API used by the framework and the "child" windows within the framework.

- What is the framework and what does it do?
    - The framework is a parent window that will contain all of the other windows in the tool. It also implements the API that is used to communicate between windows. This API is a singleton class written in incrTcl.
    - The framework also includes the tool bar, menu bar, and an MTI pane manager window [1].
    - The MTI pane manager window will contain the various debug & analysis windows, a transcript window, and a status bar.
    - The framework will handle the selection set across all windows.
    - It will also manage the drag-n-drop operations between windows.

- Framework API
    - addTextToTranscript
    - clearTranscript
    - showStatusBar
    - hideStatusBar
    - updateStatusBar
    - updatePreferences
    - start/endDragNDrop
    - methods for cross selection between windows.
    - ……. Many more

- Child Window API
    - showChildWindow
    - hideChildWindow
    - acceptDndObjects
    - dock
    - undock
    - addData
    - addObject
    - deleteObject
    - selectObject
    - …… many more

All of these API's are implemented via incrTcl methods. For example, here is a portion of the class signature for ChildWindow. A ChildWindow is one of the windows contained within the Framework window/class. For example, a window showing a hierarchical browser, a transcript window, a schematic viewer window, a waveform viewing window, etc.

```
itcl::class ChildWindow {

  constructor {frameHandle windowName windowFrame \
          windowDisplayName} {

    set _windowName $windowName
    set _frameHandle ""
    set _toolBar ""
    set _menuBar ""
    set _frameVars(child_frame) ""
    set _saveFocus ""

    ::ChildWindow::Create $frameHandle $windowFrame \
          $windowDisplayName

  }

  destructor {}

  # routine to have the child create itself.
  # i.e. Build the main window for the child
  private method Create {frameHandle childFrame childName}

  # get the name of the child window
  method getChildWindowName {}

  # get the displayed name of the child window
  method getChildWindowDisplayName {}

  # get the frame of the child window
  method getChildWindowFrame {}

  # tell the child to select/deselect something
```

```
    method notifySelection {selObject selectType {numOfSwitches "0"}  \
                {switches ""} }

    # tell the signal child window to select/deselect
    method notifySelectionToSignal {objectType objectName \
                dataType selectType}

    # add an object to the child window
    method addObject {object {numOfSwitches "0"} {switches }}

    # delete an object from the child
    method deleteObject {object {numOfSwitches "0"} {switches ""} }

    # add data to Child Window
    method addData {data}

    # delete data from Child Window
    method deleteData {data}


    # Class data members
    # handle/ptr back to framework
    protected variable _frameHandle;

    # generic "name" of a child Window
    protected variable _windowName;

    # displayed "name" of a child Window
    protected variable _displayName;

    protected variable _toolBar
    protected variable _menuBar

    protected variable _exportFormat
    protected variable _dlgToCmdFormat
    protected variable _saveFilePath
    protected variable _saveFileFormat

    protected variable _createDofilePath
    protected variable _replaceDofiletag

    protected variable _saveFocus

} ;# end ChildWindow class
```

See figure 2 for an example of the DFTVisualizer.

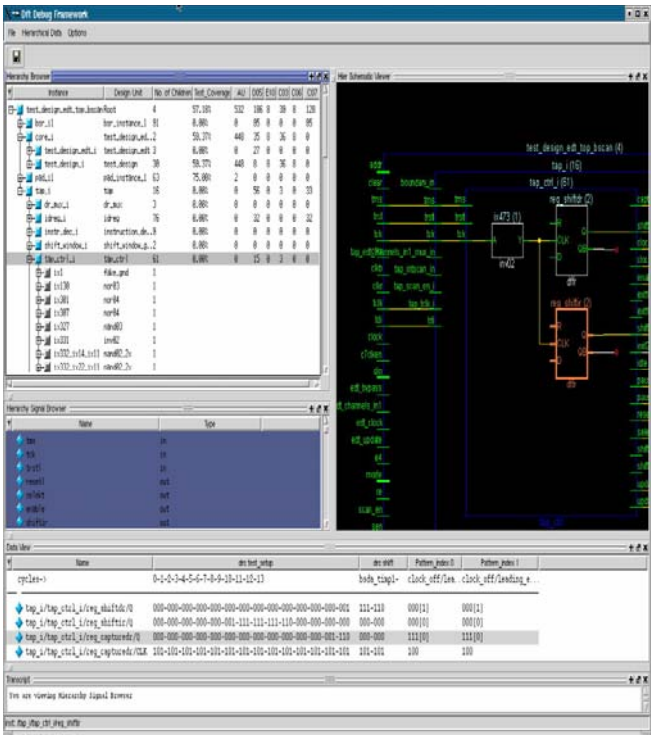

Figure 2. DFTVisualizer with multiple windows shown

See figure 3 for an example of the graphical representation of the textual report. This version will allow the user to browse through the design hierarchically and they can turn on/off the columns of data shown. This way the user can quickly navigate to the data they are most interested in seeing.

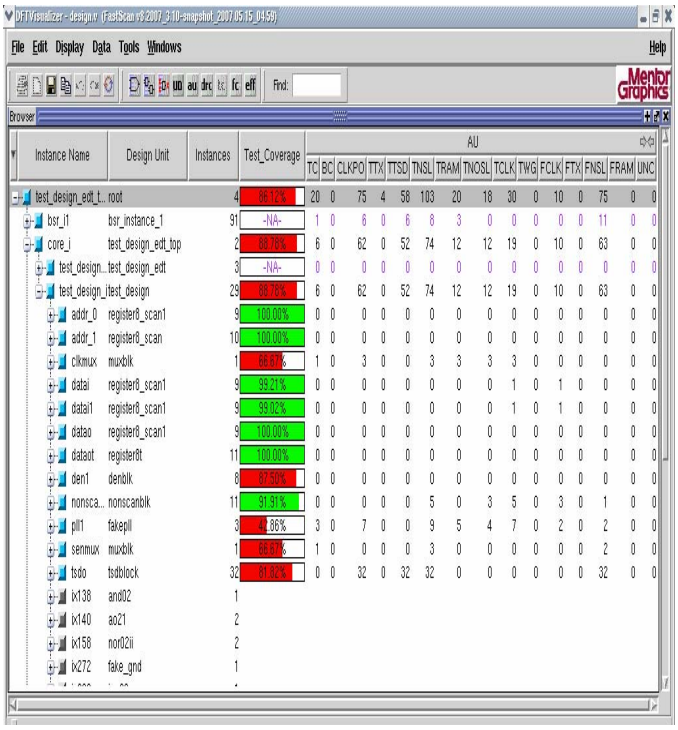

Figure 3. Graphical reporting of statistical data

Conclusion

The DFTVisualizer has many capabilities and is an order of magnitude better than its predecessor. The customers have given great feedback on the tool and have adopted it very quickly. We have many stories of customers saving days or weeks of debugging effort because of the DFTVisualizer.

Of course the customers have also been quick to ask for more features; some that we never even considered in our long term planning.

In general, the tool is solid and the customers are happy with it. The overall goal was to have a simple, easy to use, and clear process to debug problems in the customer design, and to present data in an easy to analyze format. Based on the customer response, we have achieved that goal.

Now we just need to add the other million features that the customers desire! ☺

Future Work

Currently there are 11 windows in the DFTVisualizer to aid with debugging and/or data analysis. One of the newer windows is a task manager that helps guide the user through common operations in the tool and will automatically open the appropriate windows for that task. Continuing to evolve the tool to provide more debugging and analysis options are a high priority, but looking at a higher level, this environment may eventually become a replacement for our current GUI. Streamlining task based operations will continue to be important and I expect that some project management features will be added too.

References

[1] Griffin, Brian S. The MTI Panemanager Widget 2-D Paned Window for user configurable U/I. Tcl/Tk 2005 (www.tcl.tk/community/tcl2005/abstracts/GUI/Panemanager.pdf)