The Role of Scripting in Turning EDA Flows
By Dr. Danny Rittman, July 2004

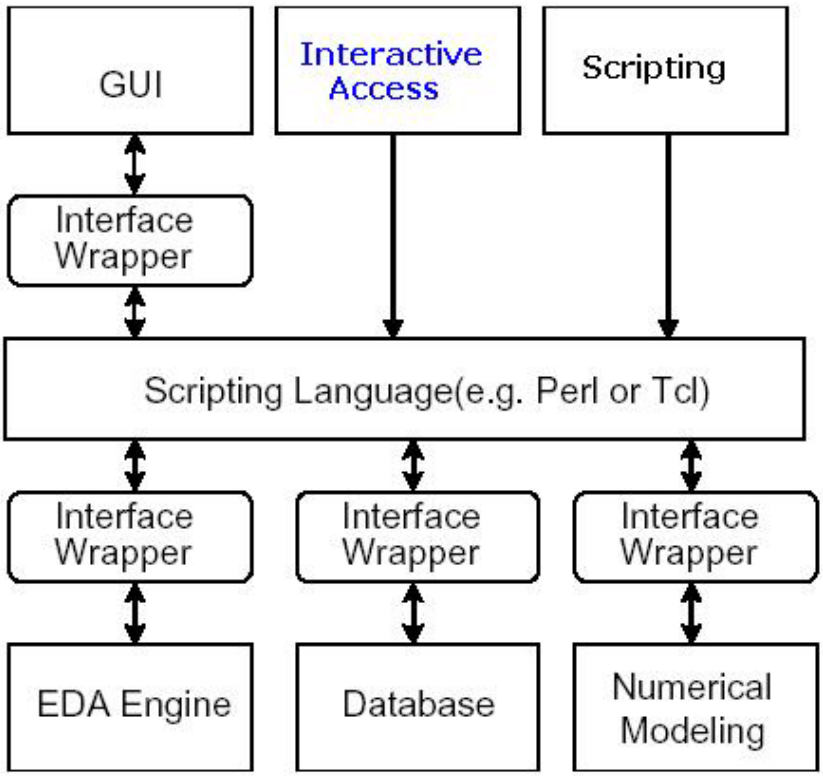
Introduction

Scripting languages have become a key factor in the EDA world due to the fact that they are well suited for high level programming and system integration. The code required for a same task is usually much less compared with a system programming language such as C/C++ or similar. (Estimated by a factor of 5X to 10X) However, it is not efficient or optimal for performance oriented tasks, for which the traditional system programming languages can work better. One approach naturally combines a scripting language at the top layer and uses dedicated and optimized algorithm engines from system programming languages for the underlying structures. This approach is very effective, flexible, and easy for scripting to embed with an application system. EDA tools are often characterized as an efficient core engine optimized for performance based on a system programming language. Hence this core often lacks the ability to integrate with the other existing systems, or need more flexible scripting and customization capability. Integrating EDA tools to enable interoperability and ease of use has been a very time-consuming and complicate process. Traditionally, each tool comes with a unique and simple set of commands for interactive use. In addition EDA platforms are offering their own proprietary macro languages. The most common are SKILL language for Cadence[1], Scheme for Synopsys[2] and Ample for Mentor Graphics[3]. These macro languages provide very powerful capabilities to create customizable features for a wide variety of purposes. The main disadvantage of these macro languages is that they are interpreter based. This reduces the responding time and therefore affecting the entire program performance. Indeed, complicate programs can be developed but when real-time response is needed a major delay may accrue especially if advanced mathematical algorithms are used. The code is hard to reuse and rapid prototyping of a new algorithm is fairly hard task. As we are stepping into VDSM arena the need for globalization is become a necessity. Interfacing to verification tools, importing/exporting data and similar tasks are demanding a different type of scripting approach.

A language that is easy and mainly intuitive! The EDA world was introduced to scripting language like Perl, Tcl and Python. These scripting languages enabled easy integration of application's interface (API)'s, interacting with industry's standard EDA tools and many other essential tasks. These scripting tools enable full programming capability connecting to EDA tools, and most important of all, any tool can be interoperated over a uniform platform on an API level. Rapid prototyping of new algorithms and entire systems becomes much easier and faster. Software reuse has become easier. Many existing extension packages for the scripting languages can be therefore integrated such as Tcl for graphic user interface (GUI), and CPAN[4] collection for various Perl applications. From a standpoint of high software quality, this approach also provides a very good vehicle for comprehensive testing of each API within EDA tools. Along the years corporations have developed entire programs and utilities using scripting languages that became an integral part of their design flow.

Tools Integration

One of the major usages of scripting language is tools integration providing efficient design customization capability to the end-users. After loading all tools within a platform, user can choose specific components based on the task need. This enables tool vendors to develop their own application system independently and hook it up later, or even create revisions without affecting the system integrity. The same methodology can be used for design flows integration. Using scripting language an entire design flow may be assembled together running few applications serially or in parallel. This opens a whole world of possibilities using EDA tools from different vendors to provide advanced design flows. Furthermore, using scripting users can create automate an entire flow and iterates tools in a sequence. Another useful feature is to access specific features within tools. Major EDA vendors are gradually providing more access to internal functions using scripting language like Tcl.



Scripting EDA tools Integration

Software Reuse, Testing and API's Access

One of the key demands from scripting language is the capability to create rapid prototypes. Many high level algorithms are not even possible without the underlying database and supporting routines. It is desired that an EDA tool developer can implement a new algorithm efficiently by leveraging existing software components. Scripting languages can bridge the gap by providing a full programming environment and linking with existing tools from a higher level of language description without any compilation. EDA tools have been lacking of interoperability for a long time. The industry is trying to provide a true interoperability, not just data exchange. The result was the development of OpenAccess which is a community effort to provide true interoperability among IC design tools through an open standard data API and reference database supporting that API for IC design. OpenAccess API's provide a high performance, high capacity electronic design database with architecture designed for easy integration and fast application development. Access to the reference database source

code is provided to allow companies to offer contributions to future database enhancements and add proprietary extensions. It will also allow for this database to be used in production environments where software maintenance is critical. Many EDA vendors are joining to support OpenAccess as part of the efforts to provide better service. One of the most popular tasks for scripting languages is to perform as a GUI or shall interface to access API functions. End-users can access APIs to do customization to fit their need using most popular scripting languages such as Perl or Tcl. Advanced GUI's can be developed by Tcl to lunch internal functions or to activate routines. The ease of use of high level language enables users to quickly develop graphical or shall based wrappers and application to run design flows. Another highlight of scripting languages is testing! Comprehensive testing of a software routine is generally very difficult and time-consuming. The common testing approach is based on an outer input and output pair. It can not handle finer grain testing for any specific API. However, with the integrated APIs in the scripting language, a tool developer can design a set of very dedicated scripts to test each API and does not have to compile another testing program to intervene with the production code. A series of regression tests for the API can be easily created to guarantee high software quality.

Scripting Languages

Scripting languages are considered to be RAD (Rapid Application Development) tools that allow programmers/users to create applications in very little time. These languages have become very popular programming tools, particularly in the EDA world, and have more programmers and lines of code than any of its nearest competitors. We can categorize scripting languages to two main types which are commercial tools macro/scripting and public domain. The industry standard EDA vendors like Cadence, Synopsys and Mentor have developed their own platforms that are equipped with strong programming proprietary languages which enable highly complex development. The other types are shareware tools that are available everywhere. The two leading scripting languages in the EDA arena are Tcl and Perl. There are also Python, Ruby and the standard shells that are equipped with UNIX, Linux platforms. (cshell, tcshell, kshell, etc') Scripting languages are considered to be one of the main achievements of the open source movement. The scripting languages are fairly simple, flexible and protect us from the "over complexity" of

the system tools. Open source is the most valuable when you are still able to change the source and this is where scripting languages come into play, as they help to write the same applications in a fraction of time. Up to few years ago Perl was the ruler in the EDA world as the main scripting tool. In the recent few years Tcl (Tool Command Language) has become the rising star due to an almost zero learning curve. Compared to Perl that requires memorizing many syntax and idioms Tcl is based on maybe a dozen basic commands to get you up and running. Major EDA vendors offer access to their tool's internal functions via Tcl scripting and it looks like it became a standard.

Conclusion

Scripting languages like Tcl and Perl were created for rapid control and systems integration. In the recent years Tcl has become an integral part of design houses design flows as Perl remain a handy tool for other useful tasks. EDA vendors are standardizing on platform-independent Tcl scripts to automate repetitive design tasks and extend the basic features of their tools. Since most of design flows use applications from multiple vendors, a designer now only needs to learn Tcl in order to control multiple tools. Furthermore, Tcl's power, versatility and ease of use is fast being adopted for a wider range of methodology tasks such as data processing; file management and tool interface solutions. No Doubt, the usage of scripting languages created a whole world of possibilities for the EDA world and therefore became an integral part of today's design flows.

References

[1] *http://www.cadence.com*.
[2] *http://www.avanticorp.com*, *http://www.synopsys.com*
[3] *http://www.mentor.com*.
[4] "Comprehensive Perl Archive Network". *http://www.cpan.org*.
[5] J. K. Ousterhout. "Scripting: Higher Level Programming for the 21st Century". *IEEE Computer magazine*, Mar. 1998.
[6] E. M. Sentovich and et al. "SIS: A System for Sequential Circuit Synthesis". *Electronics Research Laboratory Memo. No. ERL/UCB M92/41*, May 1992.
[7] L. Wall, T. Christiansen, and R. Schwartz. *"Programming Perl"*. O'Reilly and Assoicates, 1996.
[8] "Magic - A VLSI Layout System". *http://www.research.digital.com/wrl/projects/magic/magic.html*.

[9] “ScriptEDA”.
http://www-cad.eecs.berkeley.edu/~pinhong/scriptEDA.
[10] “Simplified Wrapper and Interface Generator”.
http://www.swig.org.
[11] A. Aziz and et al. “VIS User’s Manual”. *http://wwwcad.eecs.berkeley.edu/Respep/Research/vis/index.html*.
[12] D.M. Beazley, D. Fletcher, and D. Dumont. “Perl Extension Building with SWIG”. *O’Reilly Perl Conference 2.0*, pages 17–20, Aug. 1998.
[13] J. K. Ousterhout. *“Tcl and the Tk Toolkit”*. Addison-Wesley, 1994.
[14] Doulos, A Brief History of VHDL, http://www.doulos.com/fi/desguidevhdl/vb2_history.htm.
[15] Doulos, A Brief History of Verilog, http://www.doulos.com/fi/desguidevlg/vb2_history.htm
[16] Johnson, Jeff, GUI bloopers: Don’ts and Do’s for Software Developers and Web Designers, Morgan Kaufman Publishers, San Francisco, (2000), pp 42.
[17] MSDN web site: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/winui/windowsuserinterface/windowing/multipledocumentinterface.asp
[18] Petasis, George, TkDND, http://www.iit.demokritos.gr/~petasis/Tcl/tkDND/tkDND.html.
[19] Schwartz, Michael I., GDI and Print extensions for Tcl, http://www.du.edu/~mschwart/Gdi.txt, http://www.du.edu/~mschwart/Printer.txt.
[20] Svensson, Jesper, Mysund MDI http://www.geocities.com/SiliconValley/Lab/6236/tcltk.html