**AUTOMATED REGRESSION ENVIRONMENT FOR DESIGN VERIFICATION PRODUCTIVITY**

By

Keshav Rath

(Roll No. 2013104)

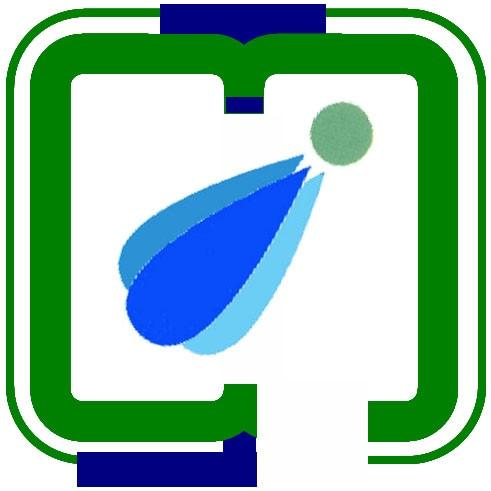
**SUPERVISOR(S):**

**EXTERNAL:** **INTERNAL:**

Mrs. RAMA KOWSALYA NAMBURI Prof. P. N. KONDEKAR

Senior Project Manager Professor

Canon ISDC, Bangalore PDPM IIIT DM Jabalpur



**Electronics and Communication Engineering (B.Tech 2013)**

**INDIAN INSTITUTE OF INFORMATION TECHN­­OLOGY, DESIGN AND MANUFACTURING JABALPUR**

Report

(16th May 2016 ­- 22nd August 2016)

**1. Introduction**

The main purpose of this project is to enhance the design verification productivity using PERL for automation. It will enhance the speed for the analysis of the reports which further help in the design and verification productivity.

Prior to this, there used to be different scripts written in PERL to execute different kinds of SystemVerilog codes. The main aim of this project is to write a common script which could work across all kinds of verification projects making it easier to operate and get the result without changing the script at all. This script would have features which are common across all projects. If anything is specific to any particular project that has to be specified in the script.

PERL is the most well-known language for the practical extraction of the files and reporting language. It has borrowed features from other programming languages such as C, AWK, sed (stream editor), sh (shell scripting), and BASIC (Beginners’ All-purpose Symbolic Instruction Code). It provide a powerful text processing facilities, facilitating easy manipulation of text files.

For this project prior knowledge of PERL as well as VHDL has to be acquired. Some knowledge about verification methodology (such as OVM, UVM) is also required. The project then extends to writing a PERL script used to run Verilog codes either sequentially (one after another) or parallel (many at a time).

**2. PERL HDL AND UVM**

**2.1 PERL**

PERL stands for Practical Extraction and Report Language. Perl is a stable, cross platform programming language. It is also a scripting language that has been developed by Larry Wall in 1987. This language was initially developed as a general purpose Unix Scripting Language but later it became a different independent Language. It is a dynamic, dynamically-typed, high-level, scripting (interpreted) language most comparable with PHP and Python. This language is used for application such as graphics programming, system administration, network programming, finance, bioinformatics, and other applications.

For this language all the basics were learnt, which include the basic syntax and semantics to using external CPAN modules for better productivity.

**2.1.1 CPAN**

Perl has mechanisms to use external libraries of code, making one file contain common routines used by several programs. Perl calls these modules. The Comprehensive Perl Archive Network (CPAN) is a [repository](https://en.wikipedia.org/wiki/Software_repository) of over 250,000 [software modules](https://en.wikipedia.org/wiki/Modular_programming) and accompanying documentation for 39,000 distributions, written in the [Perl](https://en.wikipedia.org/wiki/Perl) [programming language](https://en.wikipedia.org/wiki/Programming_language) by over 12,000 contributors. The CPAN's main purpose is to help programmers locate modules and programs not included in the Perl standard distribution. The authors maintain and improve their own module. It’s structure is decentralized. Files on the CPAN are referred to as distributions. A distribution may consist of one or more modules, documentation files, or programs packaged in a common archiving format, such as a gzipped tar archive or a Zip file.

For the understanding of this and to practice these modules available, following modules I have downloaded and practiced upon them.

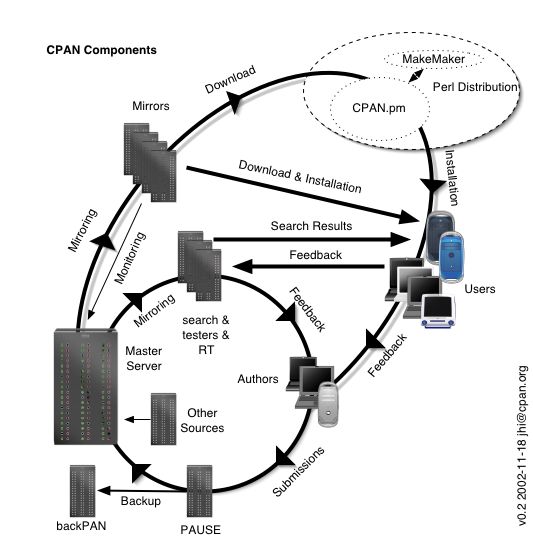


Fig. 1 – The entire understanding of CPAN network

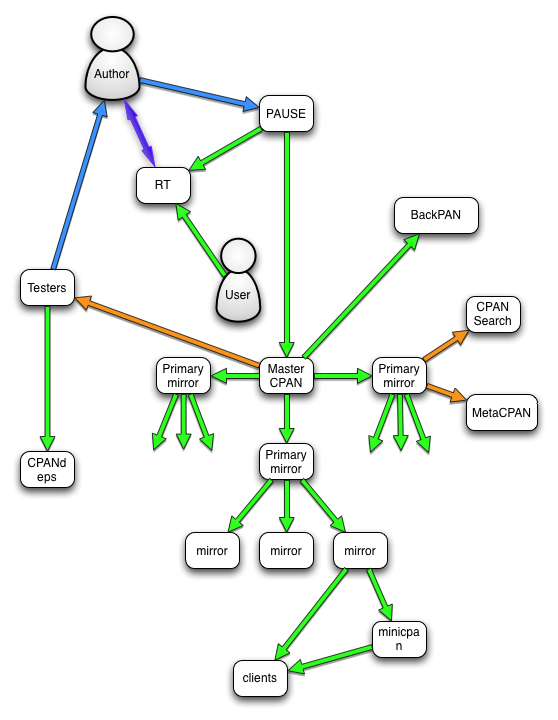


Fig. 2 – A map of CPAN ecosystem

**CPAN Class: Spreadsheet**

Spreadsheet is a broad class in CPAN and it has the many modules in it. These modules can be used manipulate excel files programmatically. It provides an object interface that makes it easier to create and parse Excel files.

* Spreadsheet::WriteExcel – To write a string, number to an excel file (Excel 95-2003).
* Spreadsheet::XLSX – To read the content of the excel file (Excel 2007).
* Spreadsheet::ParseExcel – Also to read the content of the excel file (Excel 95-2003).
* Spreadsheet::WriteExcel::Chart – To draw a pie chart using the data in the excel sheet in the sheet itself.

Some of them I have described above. All the above modules are not independent, they need some of the other modules for their implementation. Some of those dependant modules are:

* Crypt::RC4 – To use the RC4 encryption algorithms
* Digest::MD5 – Used for RSA data encryption.
* OLE::Storage\_Lite – For Document Interfacing
* ExtUtils::MakeMaker – For creating Makefile module

**2.2 HDL**

HDL stands for Hardware Description Language. As the name suggests, it is a hardware description language used in electronic design automation to describe digital and mixed-signal systems such as field-programmable gate arrays and integrated circuits. HDL can also be used as a general purpose parallel programming language.

For making RTL designs we mainly use Hardware Description Language (HDL).The mainly used HDL these days is Verilog, SystemVerilog and VHDL language. VHDL and Verilog implement Register-Transfer-Logic (RTL). SystemVerilog is a hybrid of VHDL and Verilog and it takes Object Oriented Programming approach. It can support the Logic of a SoC (System on a Chip) design. The HDL that is used for this project is SystemVerilog.

Hierarchical Modeling Concepts in SystemVerilog tells about writing modules in SystemVerilog language. Different Methodologies of design hierarchy like top-down design methodology, bottom-up design methodology are followed in SystemVerilog language. These methodologies are used to write code in SystemVerilog for the digital circuits that are available. In bottom-up methodology, we first identify the building blocks that are available to us. We build bigger cells using these building blocks until we build the top level block. Bottom up design methodology follows the vice-versa. Different levels of coding that are present in SystemVerilog language are Behavioral or Algorithm level, Data Flow Level, Gate Level, Switch Level. In Algorithm level designing is done very similar to C language. At this Data Flow level the module is designed by specifying the data flow. At the Gate Level the module is implemented in terms of logic gates. At Switch level the module is implemented in terms of switches, storage nodes, and the interconnections between them.

Different blocks that are available in SystemVerilog language are design block and simulation block. In design block we will write the code for the particular design and the simulation block is used for test bench purpose.

Different lexical conventions such as comments, operators, number specification, strings, different data types like value set, Nets, Registers, Vectors, integer, real and time register data types are used in SystemVerilog. This SystemVerilog language is quite similar to C language in using different functions for example if-else, case statement, for loop etc.

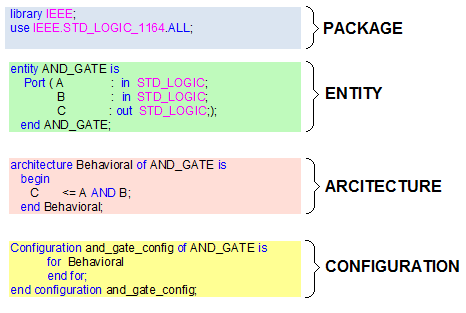


Fig. 3 – A sample of a HDL program

**2.3 UVM**

The Universal Verification Methodology (UVM) is a standardized methodology for verifying integrated circuit designs. UVM is derived mainly from the OVM (Open Verification Methodology). The UVM class library brings much automation to the SystemVerilog language such as sequences and data automation features (packing, copy, compare) etc. Unlike OVM which was independent for different softwares, UVM is a standard verification methodology supported across various vendors.

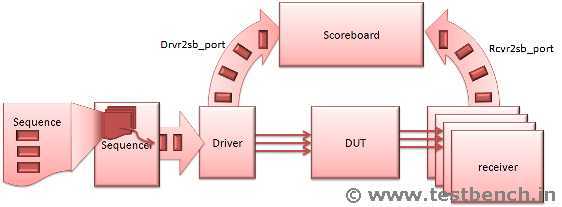
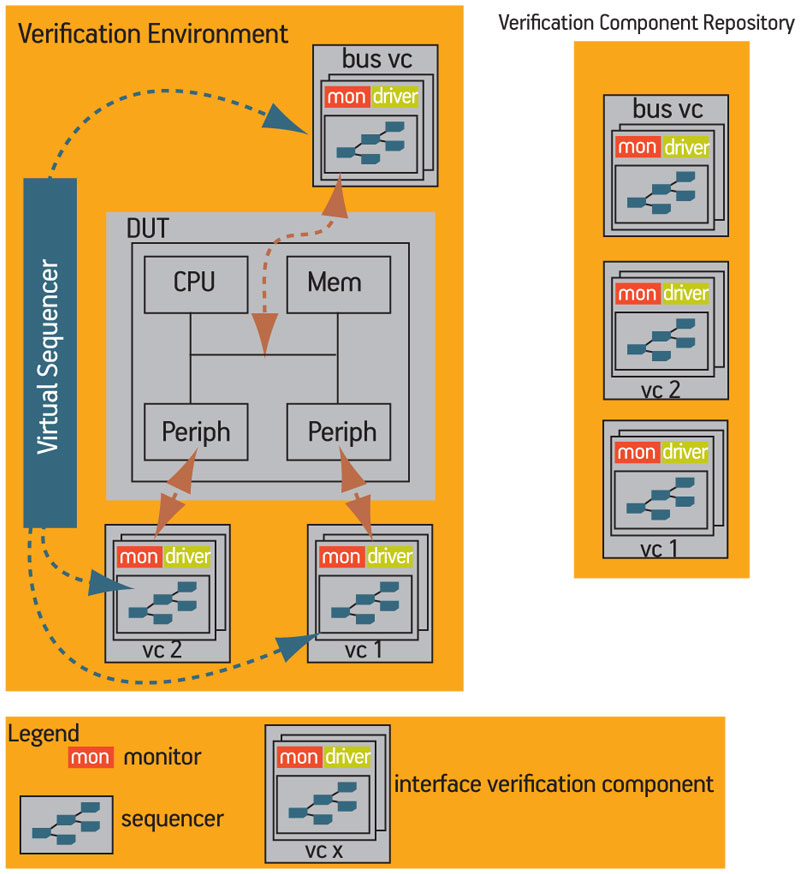


Fig3 – Basic Verification Methodology



**3. TASK**

The task at hand was to make a script which would be used to run single and regression (multiple) tests (SystemVerilog and Verilog codes) with some options placed. Various options can be given for both single and regression tests according to the users through the command line in linux. For both kinds of tests there are options which can be provided according to the users’ requirement for the given situation. These options can be mandatory or optional.

The mandatory options are:

* Type of test to be run (single/regression)
* Name of testcase (single), name of file containing name of testcases(regression)
* Timescale to run each testcase
* The simulation mode for these tests
* Tools used for simulations
* Directories to be included to run these test cases properly

Some of the extra options that can be provided by the user are:

* The UVM/OVM (Open Verification Methodology) options for running the program
* Number of times the same test needs to be run again
* GUI mode enabling/disabling, if single test is run
* PBS(Portable Batch System) mode enabling/disabling and no of licenses, if regression list is provided
* Extra options to be provided for running the test
* VIP(Verification IP) library creation (won’t be created if created once)
* Directory were the test has to be compiled
* Name of the log file (output)
* A README Document to give a brief understanding of the idea of using the scripts
* A HELP Document for telling the entire use of the script along with the description of each option provided.

For all the given options After providing all these information the code is run systematically so that

**4. RESULTS AND DISCUSSIONS**

* Initially PERL and its basics were studied.
* The study of CPAN provides an efficient way for processing the text document and display the output in the ordered manner.
* SystemVerilog and basics of UVM were studied so as to get an idea of electronic simulation.
* A script was prepared to initially check the Verilog/SystemVerilog programs.
* This script, once run, was able to develop the necessary output files which was generated after running the SystemVerilog programs.
* In some cases the simulations were seen to check whether the

**5. CONCLUSION**

I got a hands on experience on PERL language and its external libraries storage like CPAN which is a collection of external libraries and Template toolkit which has inbuilt many libraries which I have mentioned in the beginning. I also have the basic practical knowledge of Verilog and SystemVerilog. I was able to write a script according to the project requirement of the task.