Increasing the Productivity of Automation Tool by Developing Socket

Steven Fernandes and Josemin Bala
Steven Fernandes and Josemin Bala are with Department of Electronics and Communication Engineering, Karunya University, Coimbatore, India

***Abstract*—Socket is a wrapper around the automation tool which needs to be tested and it is intended to check the effectiveness of the automation tool. The main functionality of Socket includes Auto generation of test cases, parallel firing of auto generated test cases and Auto report generation. Auto test case development is a vital part to check the quality and effectiveness of the automation tool. Parallel firing of auto generated test cases reduces the testing time considerably and Auto report generated by consolidating all the log files reduces testing complexity. The proposed Socket reduces the testing time of the automation tool from 8-10 weeks to 4-5 hours. Manual labor is reduced from a team of 8 members to 2. The accuracy of the automation tool tested was increased from 45% to 92% and the overall productivity of the automation tool was increased to 99.8%.**

***Index Terms*—socket, automation tool, test cases.**

I. Introduction

In every business environment, for a project to be most successful, quality must be maximized while minimizing cost and keeping delivery time short [1]. Research has shown that at least 50% of total software cost is comprised of testing activities [2], [3]. To optimize the time and cost spent on testing, prioritization of test case is beneficial. Before preparation of test case a test plan is prepared. A test plan is a document, detailing a systematic approach to test a system which can be a machine or software. In principle, the test plans should cover all important scenarios**.** Test plan is prepared when the automation tool is in the development phase based on requirement specific, design documents.

Specification based test plan preparation is a general approach used so that test plan would contain a list of probable test cases. Automation tool is a black box that transforms inputs into desired outputs. The functionality of this black box is defined as per the specification required to carry out this transformation.

IEEE Std 610-1990 defines test case "A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement" [4]. IEEE Std 829-1983 defines test case "Documentation specifying inputs, predicted results, and a set of execution conditions for a test item" [5].

Although there are several ways of defining a test case practically all the definitions imply that a test case will either pass or fail that is how the tester verifies it as per the requirement.

The Socket is a script written using scripting languages, Perl, Tcl, It is intended to generate the test cases automatically which are of 2 types. Firstly test cases generated based on usage, secondly test cases based on connectivity. The purpose of Socket is to reduce manual intervention, decrease testing time, reduce testing complexity and thus increase the productivity of the automation tool.

The remaining of the paper is organized as follows: Section II provides a brief overview of Socket, Section III presents a case study of Socket on an automation tool. Section IV reports the experimental results, Section V draws the conclusion.

II. Related Work

In common design flows of system-on-chip (SoC) designs functional verification requires 70% of the entire design effort and most of the effort for functional verification is spent on finding and creating adequate test cases to verify that the modeled design corresponds to its specification[6].But the Socket proposed does automatic test case generation. Regression testing method used earlier to generate test cases manually is an expensive testing process and it is used to validate new features, detect regression faults, which occurs continuously during the development lifecycle. Due to time and resource constraints, it may not be possible to execute all the test cases on every testing iteration [7], [8].

Since test case prioritization is vital and there are numerous techniques in this regard [9]–[15], the Socket prioritizes the test cases automatically. After test case prioritization is completed then test cases are prepared automatically by the Socket. There are some other systems which make the test cases automatically based on specification of software and requirement [16]–[18]. But according to description provided by testers, system could not understand the different information which expresses the same means in the function description and hence manual revisiting after the generation of test case was necessary [19].

Manuscript received December 25, 2012; revised January 5, 2013.

©2013 Engineering and Technology Publishing
doi: 10.12720/joace.1.2.177-180

Industrial automation systems are tested nowadays mainly via system tests at a very late stage of development and these tests are conducted manually, are time-consuming and cost-intensive [20]. But the Socket proposed does testing of the automation tool very early in the development stage by parallel firing of the auto generated test cases and hence reducing manual intervention and testing time.

The Socket initially prioritizes the test cases which are generated automatically, then fires those auto generated test cases in parallel and finally generates a consolidated report automatically and gives a pass/fail status depending on the test case conditions.

III. A Case Study

The ABC Automation tool generates an output file based on the inputs provided to it. The Socket uses the inputs given to the Automation Tool for testing. The most basic unit that this tool considers is called a cell. The different types of cells are Combinational and Sequential. There are also certain cells which should not be considered and these are called Cell not be tested. The cell not be tested are those cells having only inputs and no outputs and also if the cells will be not be considered in the next version of the tool then they will be considered here.

The combination of all these cells together forms a Reference library. The input files of this Automation tool will contain

- *Reference cells:* The most reliable cell in terms of power and leakage is considered as a Reference Cell used in Reference library.
- *Connectivity file:*Connectivity information of various cells is provided in Connectivity file.
- *Configuration file:* The path for the Reference Libraries, Reference Cells is provided in Configuration file.
- *Group file:* This will characterize the cells of similar type together. All the Combinational Cells would belong to one group.
- *Attribute file:* This gives the attributes of the cells and also of the pins considered in each cell.
- *Controller unit:* Checks how long the exact functionality is maintained as per the specified architecture when extreme conditions are applied like temperature, pressure is applied.
- *Interface units:* The Interface Units are placed at the inputs and at the outputs.

The input file also contains information regarding the number of repetitions for each group. This Automation tool is intended to provide output file containing the exact connectivity of all the cells present in the Reference library based on the specification, cell list formed contains the total number of cells present in the library and finally it should also generate all the intermediate output files.

A. Auto Test case Prioritization

Test case prioritization schedules the test case in an order that increases the effectiveness in achieving some performance goals. Socket prioritizes the test case automatically such that the most severe faults are detected at the earliest in the testing life cycle. The prioritization is divided into 4 steps.

- Check whether the tool completes its entire run on different systems.
- The tool needs to run until specified break points.
- The connectivity information present in the output file must be as specified at the input.
- The tool needs to generate a suitable error/warning when certain input files are not available or the path specified for it is incorrect.

B. Auto Test case Development Technique

After the prioritization, the Socket generates the test cases automatically. To increase the testing coverage and efficiency the Socket divides the auto generation of test cases into four phases

Test cases for Phase 1:

- If a cell contains two outputs then the cell name should be repeated twice at the intermediate output files but the cell list should contain it only once.
- The file that contains the Reference library path should be exactly same as the Reference library present at the input.
- Certain cells which are not be tested as should not be considered from phase 1 itself.
- Phase 1 end report should contain the details of the cells used in Reference library and also details of the pin attributes for each cell.

Test cases for Phase 2:

- The different types of cells present in the library together need to form a group based on the information in the group file.
- The intermediate file that is formed at this stage should contain information regarding the inputs and outputs of the cells present in the Reference library which needs to be in accordance to attribute file.
- The Connectivity information for each cell should be passed appropriately.
- The Reference cells needs to be picked appropriately from the specified Reference library.
- Merging of similar types of cells between the two libraries need to be possible.

Test cases for Phase 3:

- If the paths present in the configuration file are incorrect then errors needs to be displayed in log files.
- If any input files were not provided then suitable error needs to be given.
- If a Reference cell mentioned in any of the input files is not present at that particular library then it is give a fatal error.
- The number of times the each group is repeated should be as per the information given at the input else error msg.

Test cases for Phase 4:

- The sourced version of the core assembler needs to be compatible with the tool.
- All the intermediate log files need to be consolidated.
- Certain environment variables needed to run the tool if not set appropriately then it should display a suitable error.
- Final output file should have exact connectivity for all the cells and needs to match with the specified architecture.

IV. Results and Discussion

The auto generated test cases needs to be fired parallel. For each test case fired there would be corresponding job id. This job id is stored and only when this job is completed the status of the test case is passed to the final auto generated report.

If these test cases are fired serially then it would consume large amount of time, typically it is about 12 to 14 days but auto generation of test cases and parallel firing of test cases the task is completed in 1 to 2 hrs.

A. Results for phase 1

The inputs given for phase 1 Socket are auto generated test cases.

- FD2TQX4 is a sequential cell having two outputs Q and QN.
- DE.lib is a Reference library which is at present at /home/stevent/DE.lib
- Cell not be tested CAP12

TABLE I. Output for Phase 1 Socket

Cell list	Reference library path	Cell not be tested
FD2TQX4	/home/stevent/DE.lib	CAP12

Phase 1 end report that contains the details of the cells used in Reference library and also details of the pin attributes for each cell is checked manually.

B. Results for phase 2

The inputs for phase 2 Socket auto generated Socket test cases.

- Library DE.lib containing Combinational cells AND2VT, AND3VT, AND4VT. Sequential cells FD2TQX4, FD2STQX4, FD2TQX2.
- The Reference cell is AND2VT.The group G1 needs to contain all Combinational cells and group G2 all Sequential cells.

TABLE II. Output for Phase 2 Socket

G1	G2	Reference cell path
AND2VT	FD2TQX4	/home/stevent/DE.lib
AND3VT	FD2STQX4	/home/stevent/DE.lib
AND4VT	FD2TQX2	/home/stevent/DE.lib

For merging of similar type of cells we consider a library FG.lib containing cells Combinational cells OR2VT, OR3VT. Sequential cells FD2QX4, FD2QX8.After merging all the Combinational cells need to be in G1 and all the Sequential cells in G2 and the Reference cell path is given

TABLE III. Merging of Groups In Phase 2 Socket

G1	G2	Reference cell path
AND2VT	FD2TQX4	/home/stevent/DE.lib
AND3VT	FD2STQX4	/home/stevent/DE.lib
AND4VT	FD2TQX2	/home/stevent/DE.lib
OR2VT	FD2QX4	/home/stevent/FG.lib
OR3VT	FD2QX8	/home/stevent/FG.lib

C. Result for phase 3

The inputs given for phase 3 Socket is the paths of the configuration file and also an input file which contains details regarding the Reference cells and the Number of repetitions for each group. If there is any mismatch in any of these files or if these files are not existing then the log file generated will have the following errors

- *E The configuration file path incorrect.
- *E The input file does not contain information about the number of repetitions.
- *E The reference cell is not present in the path specified.
- *E The reference library does not exist.

D. Result for phase 4

Inputs given to phase 4 Socket involve sourcing the Core Assembler, setting the environment variables used to run the tool and intermediate log files generated. The final output generated at the end of phase 4 should contain the exact connectivity for all the cells.

V. Conclusion

As technology continues to grow and become more complex, testers will be faced with tougher challenges to fully test the automation tool within the time given to them. In this paper, we develop a Socket which is adaptable for any version of the automation tool, since it is a generic script. The final Socket output would depend on the following conditions. If there exist an error in any of the log files and if the final output file is not generated because of the errors then its status would be pass else it would be fail.

The output of the automation tool should have the connectivity as per the design specified. To validate the connectivity another tool is invoked which would generate a connectivity based output file and this file is used for comparison. Hence the second pass/fail status would be based on connectivity.

By developing Socket the testing time of the automation tool was reduced from 8-10 weeks to 4-5 hours. The manual labor was also reduced from a team of 8 members to 2. The accuracy of the automation tool tested was increased from 45% to 92%. It also reduces testing complexity and manual intervention to a large extent. All these factors increase the overall productivity of the automation tool to 99.8%.

References

[1] J. Karlsson and K. Ryan, "A cost-value approach for prioritizing requirements," *IEEE Software*, vol. 14, no. 5, pp. 67-74, Sep-Oct 1997.

[2] M. Harrold, "Testing: A roadmap," in *Proc. International Conf. on Software Engineering*, Limerick, Ireland, 2000, pp. 61-72.

[3] L. Tahat, B. Vaysburg, B. Korel, and A. Bader, "Requirement-based automated black-box test generation," in *Proc. 25thAnnual International Computer Software and Applications Conf.*, Chicago, Illinois, 2001, pp. 489-495.
[4] *IEEE Standard Computer Dictionary*. A Compilation of IEEE Standard Computer Glossaries.
[5] *IEEE Standard for Software Test Documentation*.
[6] C. Kirchsteiger, J. Grinschgl, C. Trummer, C. Steger, R. Weiss, and M. Pistauer, "Automatic test generation from semi-formal specifications for functional verification of system-on-chip designs," in *Proc. Systems Conf., 2008 2nd Annual IEEE*, May 2008.
[7] H. Leung and L. White, "Insights into regression testing," in *Proc. International Conf. on Software Maintenance*, Miami, Florida, U.S.A., Oct 1989, pp 60-69.
[8] K. Onoma, W. Sai, M. Poonawala, and H. Suganuma, "Regression testing in an industrial environment," *Communications of the ACM*, vol. 41, no. 5, pp. 81-86, May 1998.
[9] S. Elbaum, A. Malishevsky, and G. Rothermel, "Prioritizing test cases for regression testing," in *Proc. International Symposium on Software Testing and Analysis*, Aug 2000, pp. 102-112.
[10] S. Elbaum, A. G. Malishvesky, and G. Rothermel, "Test case prioritization: A family of empirical studies," *IEEE Transactions on Software Engineering*, vol. 28, no. 2, pp. 159–182, 2002.
[11] S. Elbaum, G. Rothermel, S. Kanduri, and A. G. Malishevsky, "Selecting a cost-effective test case prioritization technique," *Software Quality Journal*, vol. 12, no. 3, pp. 185-210, 2004.
[12] D. Jeffrey and N. Gupta, "Test case prioritization using relevant slices," in *Proc. 30th Annual International Computer Software and Applications Conf.*, 2006, pp. 411-420.
[13] J. Kim and A. Porter, "A history-based test prioritization technique for regression testing in resource constrained environments," in *Proc. 24th International Conf. on Software Engineering*, May 2002, pp. 119-129.
[14] Z. Li, M. Harman, and R. Hierons, "Search algorithms for regression test case prioritization," *IEEE Transactions on Software Engineering*, vol. 33, no. 4, pp. 225-237, April 2007.
[15] A. Malishevsky, J. Ruthruff, G. Rothermel, and S. Elbaum, "Cost-cognizant test case prioritization," *Technical Report TR-UNL-CSE-2006-0004*, March, 2006.
[16] K. Wee, C. KhooSiau, and Y. Sun, "Automated generation of test programs from closed specifications of classes and test cases," in *Proc. 26th International Conf. on Software Engineering*, 2004, pp.96-105.
[17] L. Peng, "Test case generation for specification-based software testing," in *Proc. Conf. of the Centre for Advanced Studies on Collaborative Research*, 1994, pp. 41.
[18] M. Fisher, M. Cao, R. Gregg, R. Curtis, and M. Margaret, "Automated test case generation for spreadsheets," in *Proc. 24th International Conf. on Software Engineering*, 2002, pp.141-153.
[19] Z. Ning, G. Yang, "An automate test case generation approach: using match technique," *Computer and Information Technology*, pp. 922-926, 21-23 Sept 2005.
[20] Krapfenbauer, H. Ertl, D. Zoitl, and A. Kupzog, "Improving component testing of industrial automation software," *Computing in the Global Information Technology*, pp. 259, 2009.

Steven Fernandes is a first author. He received his M.Tech in Microelectronics from Manipal Institute of Technology, Manipal, India, 2011. He is a research scholar in Department of Electronics and Communication Engineering at Karunya University, Coimbatore, India. His field of interest includes face detection and recognitions using genetic algorithms.

osemin Bala is a corresponding author. She received her Ph.D. from Anna University, Chennai, India, 2008. She is Professor and Head of Department, Electronics and Communication, Karunya University, Coimbatore, India. Her areas of interest are digital image processing.