International Journal of Hybrid Information Technology
Vol. 9, No.6 (2016), pp. 51-66
http://dx.doi.org/10.14257/ijhit.2016.9.6.05

Pre and Post Silicon Validation Automation with Graphics Behavior Checker

Imran Ahmed

SCSE VIT University, Vellore, Tamil Nadu – 632014
Imranahmed2k9@gmail.com

Abstract

Pre and Post silicon validation automation is used for testing graphics in the computers, tablets and smart phones. A graphics driver is software that allows operating system and programs to use computer graphics hardware. To test the performance of this display system pre and post silicon automation is used such that end users get the best high definition experience. Pre silicon automation is based on emulation of graphics hardware and testing on it before it is actually build. Post silicon automation is testing of actual graphics hardware after it has been manufacture. Post silicon automation works to detect and fix bugs in graphics hardware of the system after it has been manufacture. Due to complex design of the hardware it is impossible to fix all bugs before it is manufacture. Pre silicon automation is an attempt to rectify any bug that can occur before the manufacture of graphics hardware such that it is way to reduces number of cases of bugs in graphics hardware that can occur after it is produce. Graphic Behavior in which each section is to assigned team of architects and micro-architects whose expertise targets a particular functional area for both hardware and software. It is record of all information that is kept for registers used in any sequences such that all information generated after running from driver is used to cross verify.

Keywords*: Pre Silicon, Post Silicon, Graphics Driver, Testing Automation, Graphics Behavior Checker.*

1. Introduction

Graphics driver is a program that controls how graphics components communicate with rest of the computer like other software, monitor and so on. A graphics card is also known as a display adapter, video card, or graphics controller, is a card plugged into a computer to create signal that are displayed on monitor. Graphics drivers are the software that runs graphics card, connecting them to operating system. There are different types of drivers for each graphics card and are most often provided by the manufacturer of the graphics card. It is important to keep these driver updated to get the best performance of computer.

Before anyone can use this graphics hardware, it has to be tested. Method of testing can be manual as well as automation. Manual testing requires lots of work force as compare to automation as well as manual testing is time consuming and uses improper allocation of resources. Automation testing can be of two type's pre silicon automation and post silicon automation.

Pre silicon automation is the case where whole these graphics hardware are emulated in the virtual machine for testing before they come to existing situation. All the test cases are run on the virtual graphics hardware to come across the feasibility of the system and to gain knowledge about such type of graphics hardware is adequate and meeting the requirements and effectively is reducing the cost.

Post silicon automation is the case where actual graphics hardware is tested and is being verified and ready for use. Various kinds of test cases are run on this hardware to make hardware prominent in use and make it authentic.

For the pre silicon automation, the main motive is reducing maximum number of bugs present in the graphics hardware. A generic tool has to be implemented such that it can detect bugs on any operating system. To implement this, a tool is developed in the python which accept test cases from the server and implement. It can be used to run test cases and gain information about the virtualized graphics hardware.

For post silicon there is a tool which takes input the required number of test cases has to be run and it gets the required driver that need to be installed as prerequisite, after successful installation of drivers it divided the test cases in the machine pool to get the result.

Graphics behavior checker is a tool that cross check all the functionality provided by the graphics driver. It tries to make graphics driver robust such that best performance is obtain from the from graphics card. It runs various sequences on graphics driver to validate each component models are working as it was desired.

2. Related Work

Since the 1970s, few crucial thoughts in formal and semiformal confirmation have advanced from hypothetical idea to mechanical practice. The apparently most essential thought is that of formal details – affirmations. Transient rationale [2][3] and comparable automata-based detail dialects have advanced into institutionalized modern utilization [1][10]. Declarations are integral to post-silicon approval as they give an exact technique for determining application prerequisites and framework usefulness that could be unfavorably influenced by post-silicon bugs. Algorithmic check methods, taking into account progresses in Boolean thinking, for example, Boolean Satisfiability (SAT) [4][11][13] also Binary Decision Diagrams (BDDs) [5][6][9], are likewise focal to post-silicon acceptance. Maybe one of the greatest modern triumphs of formal equipment confirmation is combinational proportionality checking. What's more, late advances in SAT-based calculations and circuit improvement are scaling up consecutive proportionality checking enormously, as showed by the ABC framework [16][18][19]. Going past proportionality checking, model checking [7][8][12], i.e., choosing whether a framework fulfills a property determined normally in worldly rationale, is currently a key part of all mechanical formal confirmation instruments. Formal methods are frequently consolidated with conventional irregular recreation strategies. At last, hypothesis demonstrating routines are additionally standard modern practice today, especially in chip configuration and check (e.g., [14][15][17]). One of the real drivers for formal check going ahead, both utilizing hypothesis demonstrating and model checking, will be satisfiability modulo hypotheses (SMT) solvers [20][22][23]. It is clear from the above exchanges that central examination empowered huge advance in the reception of organized techniques for assembling testing and configuration confirmation. With developing multifaceted nature of post-silicon[21][24][25] acceptance, our trust is that new thoughts will rise that will change post-silicon acceptance from gifted workmanship rehearsed by a couple of experienced designers to an order with solid establishments empowered by organized methodologies and outline robotization[28][31][32]. Such new systems can have broad effect on different fields identified with post-silicon approval. Illustrations incorporate dependable framework configuration, installed frameworks [29][30], and programming test and check.

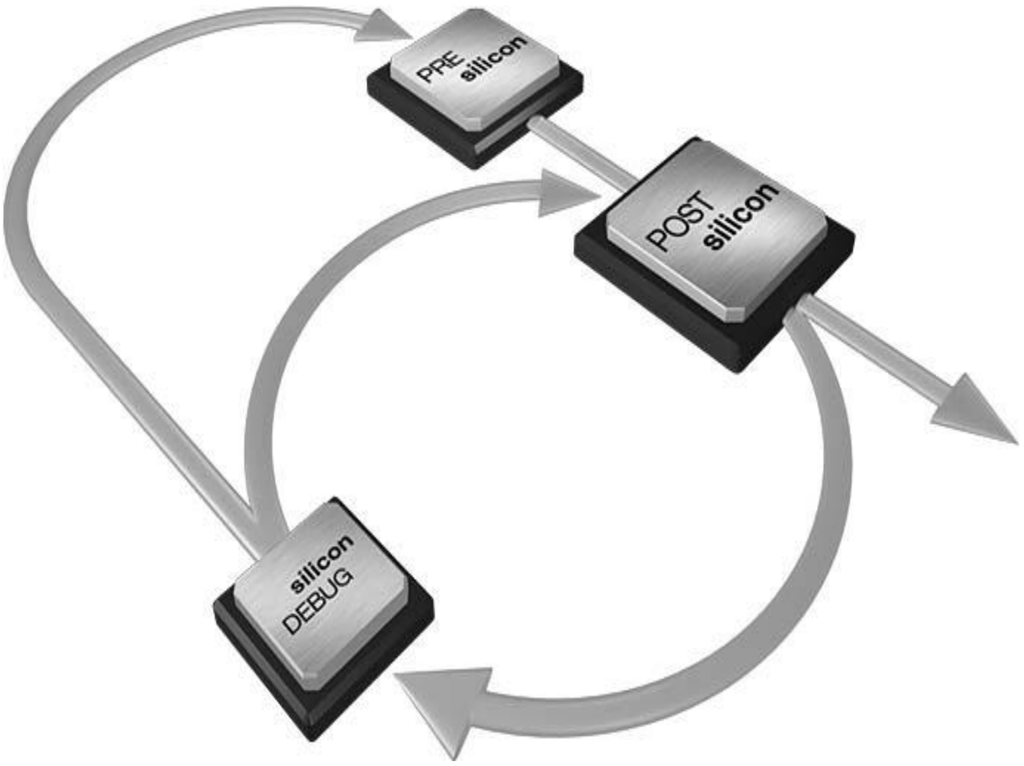


Figure 1. Silicon Validation Automation

A great part of the testing happens in programming, before a physical segment is ever made. Before reaching acceptance testing keeps amid the post-silicon periods of improvement. When a part achieves the commercial center, the outline has been approved over more than one hundred trillion transport cycle [49][52] mixes. By utilizing a portion of the business' most developed recreation devices, it starts accepting part outline much sooner than the first silicon [40][41][42][43] models are delivered. The advanced configuration is initially reproduced in programming, and afterward tried under a colossal [48] mixed bag of working conditions. Tests are intended to give exhaustive approval scope at all levels. At the Unit Level [33][36][37] first its approval specialists test the inner workings of every significant subsystem inside the segment. Utilizing modified programming devices [50][51], they can control and screen all subsystem interfaces, empowering a larger amount of testing than is conceivable in equipment. Issues can be identified prior and determined all the more rapidly to quicken item advancement. At the Chip Level [34][35][38][40] Segment execution is likewise tried with all subsystems working all the while. Here once more, all interfaces are firmly controlled and checked, and usefulness is accepted under a colossal mixed bag of conditions. At the framework level [39][44][46], the reenacted segment[45][47] is tried in a full working environment to guarantee that it meets expectations impeccably in coupled with other stage parts. Operation is likewise tried against industry transport models, to approve similarity in for all intents and purposes any environment that sticks to proper gauges.

3. Methodology

The challenge is to provide a tool such that it is platform independent and is able to track to bug in the graphics hardware whether it is in virtual state or it is being used after it has been manufacture therefore it provide better end user experience. After running test cases on the graphics hardware it’s task is to produce result in form of html, xml and in log file such that tester can verify the result or error occurred during execution can be corrected. Therefor for both pre and post silicon it is important to generate log files in running various test cases on various operating system. This graphics testing tool provides testing the graphics in its emulation and also after it has been manufacture and it also enables logging facility for the software tester and it also provides a very important feature of scheduling of test on machine pools. It supports of simulation environment as well as seamless integration with other environment, including simulation, emulation and actual silicon. It also facilitates screen capturing such that software tester get better perspective of the error occurred or result obtain.

3.1 Pre-Silicon Automation

3.1.1 Preparation of Tool

First need is to prepare tool such that it can start executing test case, but before that one need to check is it the latest tool on which test cases will be running. For this one need to check the version file located in both server and local tool. A comparison is made between these two files and if the server has the latest version then tool is downloaded to local client. If the both have same version then nothing has to be done everything is up to date.

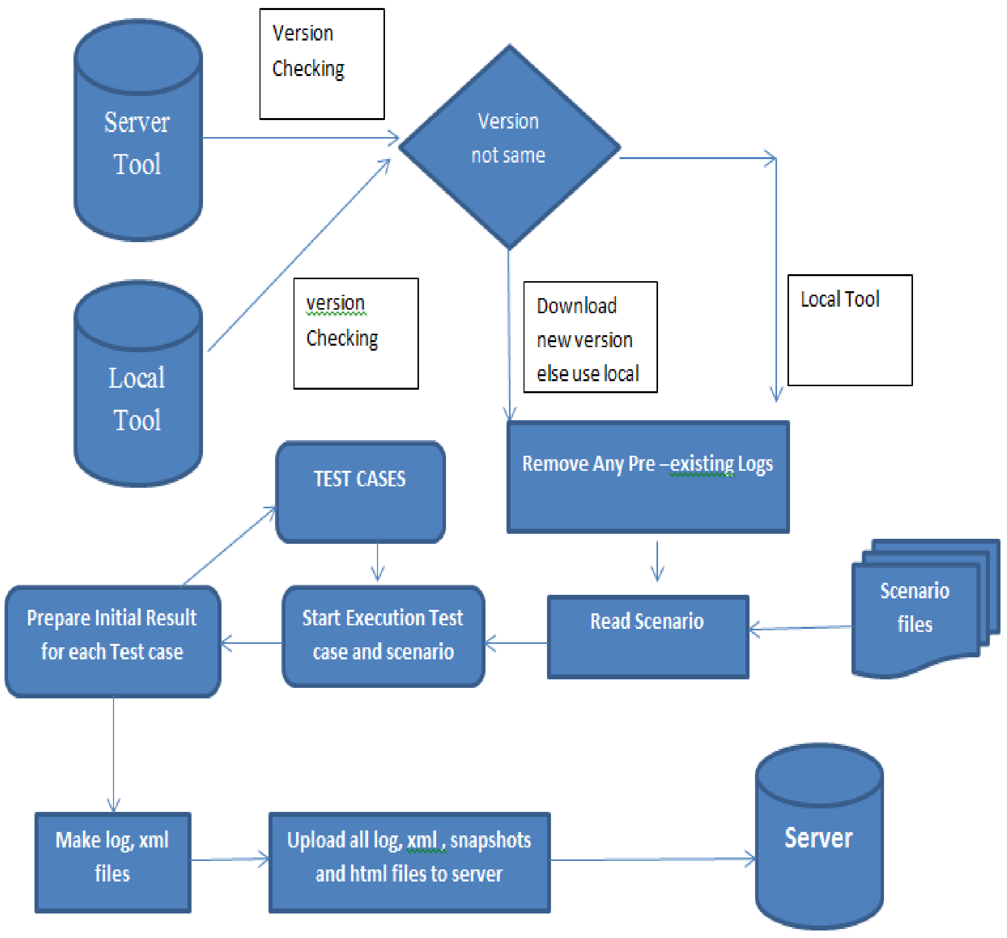

Server Tool
Version Checking
Version not same
Local Tool
version Checking
Download new version else use local
Local Tool
TEST CASES
Remove Any Pre –existing Logs
Prepare Initial Result for each Test case
Start Execution Test case and scenario
Read Scenario
Scenario files
Make log, xml files
Upload all log, xml , snapshots and html files to server
Server

Figure 2. Pre Silicon Automation

3.1.2 Remove logs

After the preparation of tool one has to look for the any pre –existing logs are there in the server, if there are any logs in the server then remove all logs because it create problem when new test case run and produces the same log, so it might be problem when dealing with logs. Best option to delete the logs as soon as tool is ready. After the removal of logs, one can proceed as there might not be any previous result stored.

3.1.3 Read Scenario Script

After removing logs one has to read the scenario script which the list of parameter commands that needs to executed, as passing different parameter to test the performance of the graphics hardware. The all parameter are read and converted into list such that it can be made available for further execution

3.1.4 Execution of Commands

Reading the scenario script, a temporary file is created such that each command execute independently. Taking the first commands line it is passed to the tool to start executing. Tool takes the command as the parameter and start running and it will produce result which can be pass, fail and sporadic. It identifies that particular test passed or failed. Therefore it creates a snapshot of each case such that they know the result and what happens during the execution.

After that it create a log file in which store the information about the location where what happens during execution of code. It generates an html file with result and specifying what happens during execution and xml file with desire result.

3.1.5 Preparing Initial Results

After the execution of commands as a parameter for the tool, the execution test return the result in form of list, such that final result layout can be prepared such that for all test cases and information can be shown to software tester. It starts creating log and xml file. Log file contain information regarding the total number of test cases passed or failed during execution. A xml file that contain same information but detailed description what happen during execution and run time of each test case.

3.1.6 Uploading all Information

After the whole processing information has to be made available to the software tester person, therefore all log, xml, html are uploaded to the server such that it is available to software tester. The files are kept in the server for future reference such that they can look out for any information they need in near future.

3.2 Post Silicon Automation

Post silicon automation tool is an automated test environment developed by Intel Corporation, for validation of graphics hardware. Automation test on this tool can initiate through submission of test request or scheduled test request. Execution on this tool requires Shared binary, Test Version, Test Suite and Test Request.
SB – Shared Binary will hold the details about Test Binary
TV – Test Version will hold the details about Test CMD Lines
TS – Test Suite will hold the details about Test Component
TR – Test Request will hold the details about Test Configuration
Shared Binary (SB) will be imported into Test Version (TV)
Test Version (TV) will be imported into Test Suite (TS)
Test Suite (TS) will be imported into Test Request

Requirements
1. SB – Shared Binary
2. TV – Test Version
3. TS – Test Suite
4. TR – Test Request

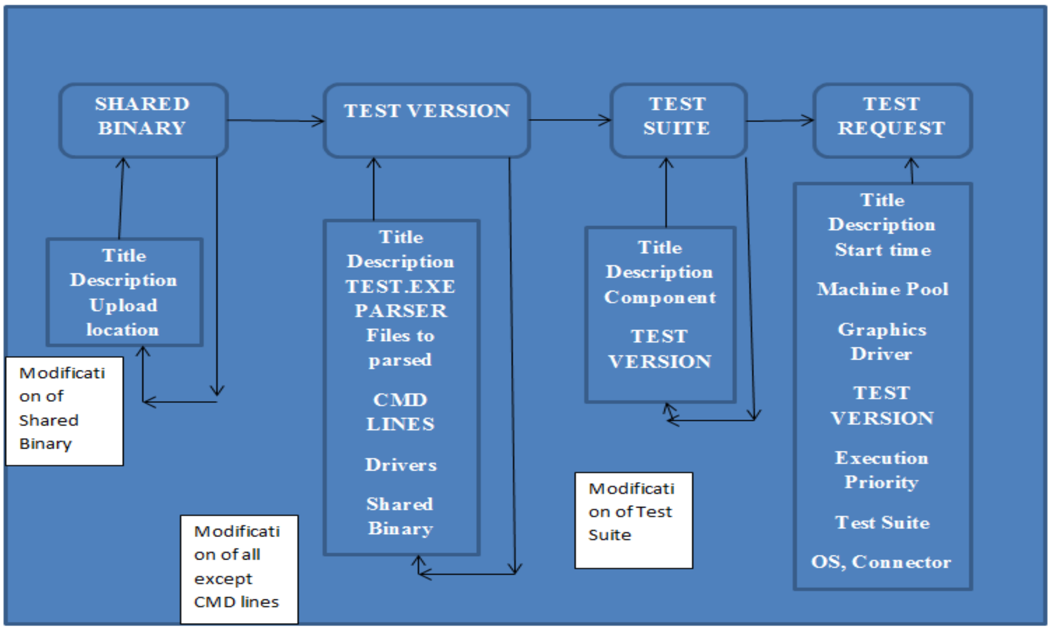


Figure 3. Post Silicon Automation

3.2.1 Creation of Shared Binary

The Shared Binary form is used to allow the user to upload files (executables, libraries) that can be shared by multiple tests alleviating that need to load the libraries with each and every test. There is also a Link Shared Binary form that has similar characteristics to the Shared Binary except for how the content is uploaded.

The Shared Binary form allows a user to upload a folder of files that can be shared across multiple tests. The folder and files are stored in a network share known as the Shared Binary store and created as an item in the database. Users creating a new Test Version can reference as many Shared Binaries as needed from the Shared Binaries tab. The process of creating a Shared Binary is equivalent to a configuration step.

The process of creating shared binary

1. One has to keep the copy given test binary on common share location so everybody can access on same network
2. Open the user interface where has to provide all information.
3. Create New Shared Binary and provide following details like
   - Title
   - Description
   - Upload Folder location
4. Once upload is done, there will be a confirmation pop-up and it shows that upload has been done.
5. Click OK pop-up window and Click on Submit button. Then Note down the Shared Binary number for future reference.
6. For modification to the shared binary one has to provide different share location make it synchronize.

7. In this making of shared binary one has to keep account of reference number.

3.2.2 Creation of TEST VERSION

The Test Version form allows the user to upload a folder of files that will be used within the test, often including a file that will be executed by the test. The user must specify the file to be executed, information on log files, and a Test Template to use (currently the Windows Test Template is used by all requests). The user can also specify any number of shared files to be used by the test. Shared files, known as a Shared Binary, allow one set of files to be used across multiple tests without having to be repeatedly uploaded the content. In addition, the user must specify at least one command line argument to be used with the test file to be executed. The process of submitting a Test Version only saves the Test Version and Test Variation data for future reference within a Test Suite. The process of creating a Test Version is equivalent to a configuration step.

Test Version (TV) will hold the details about test CMD lines which have to be passed as parameter and compatible platform or operating system.

1. Keep copy given CMD lines in a text file and make it as single line by separating it with colon (:), such that it will easy to split them based upon colon as parameter.
2. Create new Test Version and it will hold following information
   - Title
   - Description
   - Operating system template
   - Timeout in minutes
   - Log parser to be use
   - Test exe that has to be executed
   - Log files which has to be parsed
   - Deleting log files before leaving
   - Providing colon(:) as parameter to separate the CMD lines.
   - Graphics Driver and Operating System has to be selected for execution
3. Import Shared Binary which is already by using its reference number.
4. Ensure all fields are filled and properly and submit TV.
5. Note down the TV for future reference.
6. Modification is possible everywhere except in CMD lines that have been already given for execution.

3.2.3 Creation of TEST SUITE

The Test Suite form allows users to group Test Versions (and their referenced Test Variations) together into a single plan for execution within a Test Request. The Test Suite compiles data from the Test Variation and selected Test Template from the Test Version and creates an Execution Plan XML with Operation nodes for each Test Variation that contain an expanded Test Template populated with data from the Test Version and Test Variation. The process of creating a Test Suite is equivalent to a configuration step.

Test Suite (TS) will hold the details about test CMD lines and Test Component.

1. Create new TEST SUITE and it will hold following items.
   - Title
   - Description
   - Component on which has to be tested
2. Import Test Version in the Test Suite by using its reference number.
3. Submit the Test Suite and note down the reference number for future use.
4. Modifying Test Suite like blocking or removing any particular CMD line or Importing new Test Version into Test Suite can be done.
5. Whenever modifying Test Version, synchronization should be done on Test Suite accordingly. Otherwise Test Suite will have older values of Test Version.

3.2.4 Creation of TEST REQUEST

The Test Request form is the basic entry point for executing a test in tool. The Test Request form must specify a title and at least one Test Suite to be executed. The Test Request can also include any number of compatible Components and Configuration items. Various test execution options can be set to alter how the test is executed. Tests can be submitted to execute immediately or scheduled for the future. When a Test Request is submitted, it is transitioned to a SUBMITTED workflow state. Test Requests that have a scheduled start time that is current or in the past are picked up by the Scheduler service and processed further. Test Request (TR) will hold the details about test CMD lines, Configuration on which test should run, Machine pool and Time frame as when it should run.

1. Submit Test Request and it will contain the following things.
   - Title
   - Description
   - Start time or Scheduled time
   - Machine Pool
2. Import graphics driver by using it code.
3. Select the execution priority for the test case ranging 1 to 10.
4. Import Test Suite by its reference number which has been already created
5. Check the split test, because it will divide the test cases for many machines to run simultaneously and parallel.
6. Import Operating System, driver for execution of the test cases.
7. Now ensure all the required fields are filed. Then submit the Test Request and note down the Test Request number for future reference.
8. Now you will receive a mail from software with Test Request number detail. Now refresh the browser and confirm the Test Request is become to scheduled state. Because if something wrong on Test Request creation, automatically it will get cancel once it submitted.

In pre and post silicon automation the test cases are taken from the server and are distributed in the machine pool. Each computer takes the test case and executes it independently and produces result.

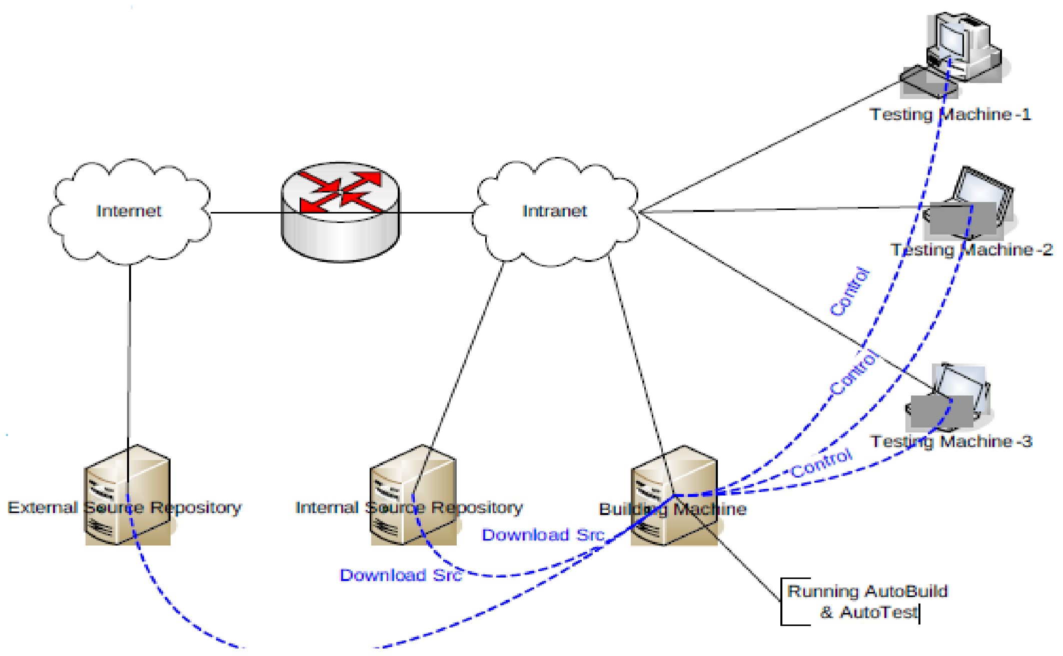


Figure 4. Test Cases Distribution and Execution

In the above figure it shows how the test cases located on server and local are compared and checked which version is greater according to that tool is downloaded or used local one. The test case that has to be executed is passed on to the testing machine for testing graphics hardware.

System design for pre and post silicon automation dependent upon the system resource required which test case has to be run and which operating is required for running test cases. To follow this a simple procedure need to be followed in creating shard binary, test version , test suite and test request.

3.3 Graphics Behavior Checker Tool

Till now validation testing was either black box or white box testing, a new approach has been proposed such that after running test cases either in case of pre-silicon or post silicon, driver logs are collected to further analyzed to identify error not only by just seeing display the graphics driver show but also knowing the status of each register. If the driver follows the predefined sequences and display is coming on machine then driver is in pass state else it failed. Even if the display is coming but the driver does not follow the predefined sequence then driver status is failed. To rectify the problem semi-white box testing has been implemented.

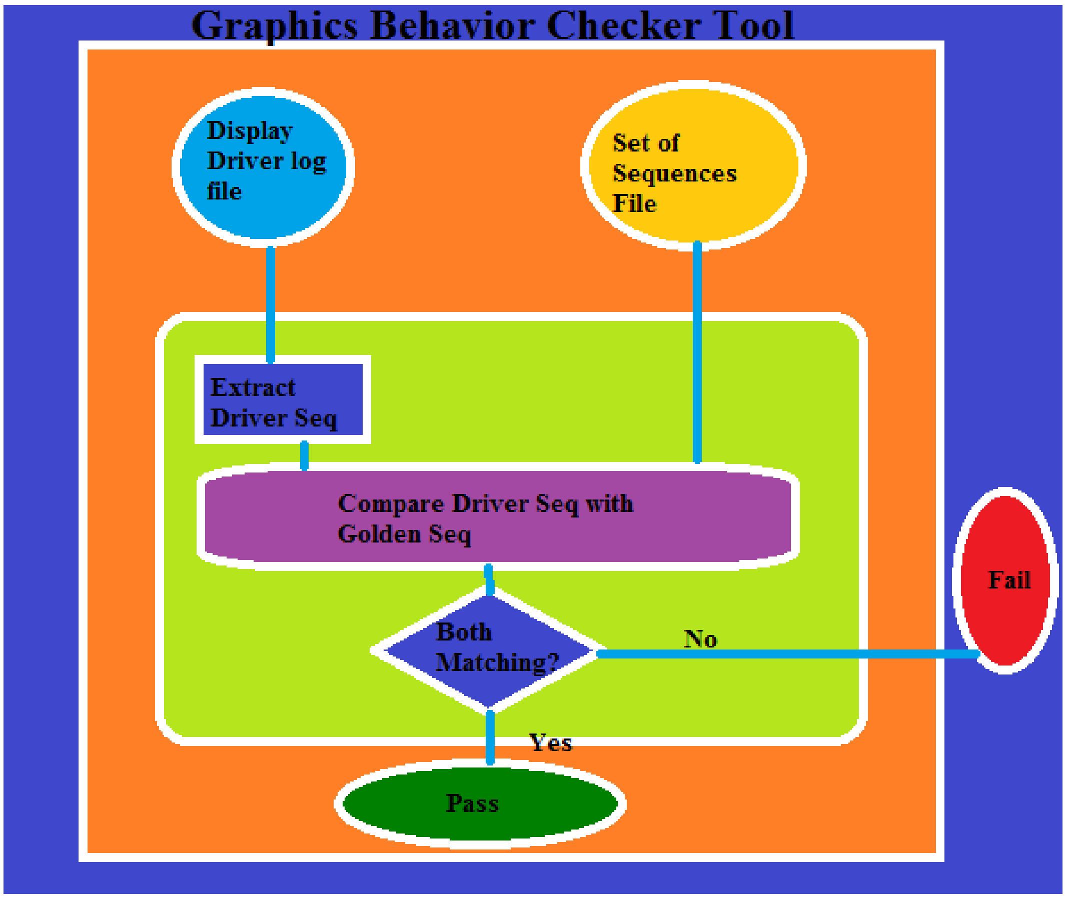


Figure 5. Graphics Behavior Checker Tool

Current validation status mostly cover user lever validation, end users scenarios, minimal predefined test case on display devices or predefined test case at component level.

Graphics Behavior Check Tool works with driver log and cross verify it with predefined sequence rules. For display devices there are many sequences which involve cloning display, HDMI, extended display such that if all these operation perform by the graphics driver is it working as required format. Two input files are passed to graphics behavior tool, one is from driver log obtain after running test case either in pre-silicon or post- silicon environment and another is developed as a golden sequence file as defined in graphics driver normal behavior. Tool need to have mechanism to parse and extract driver logs and match with the sequences to report out the results as pass or fail.

Golden Sequence file: It will help validation team to run a standalone sequence flow file as per sequences defined already and it will easy for validator to tell exact that has occurred in graphics driver testing.

Display Driver log: Getting a driver log files obtained after running graphics driver in display system.

This approach is going to assure on solving all the gaps one have in display validation, but creating a framework to enhance validation approach from black box to system level and white box validation. This will also help us to communicate right issues across teams to narrow down the problems. Sequence analysis will help both development and validation team to reduce the debug time. Using these type of exposure in pre-silicon and post-silicon will create better coverage and to callout clear indicators to maintain the component health.

3.4 Test Failure Analysis Tool

If the test case fail one has to analysis the cause of failure that has occurred therefore, reference number has to be passed to test failure analysis tool such that validator don't have to look into files to get information. By providing the tool a reference number of the test case, it scans whole logs and provides the reason behind the failing of test case.

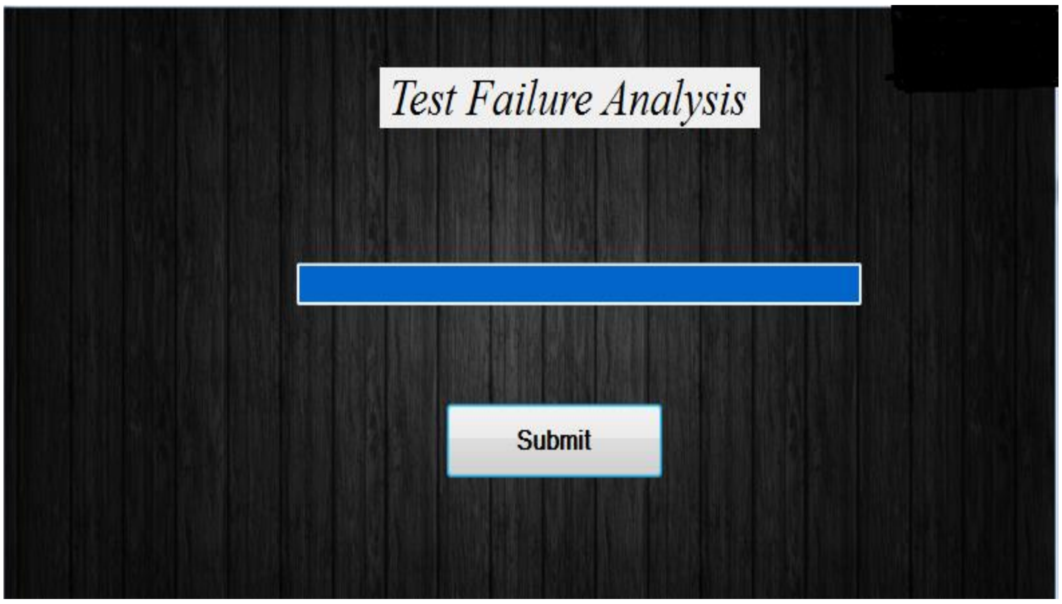
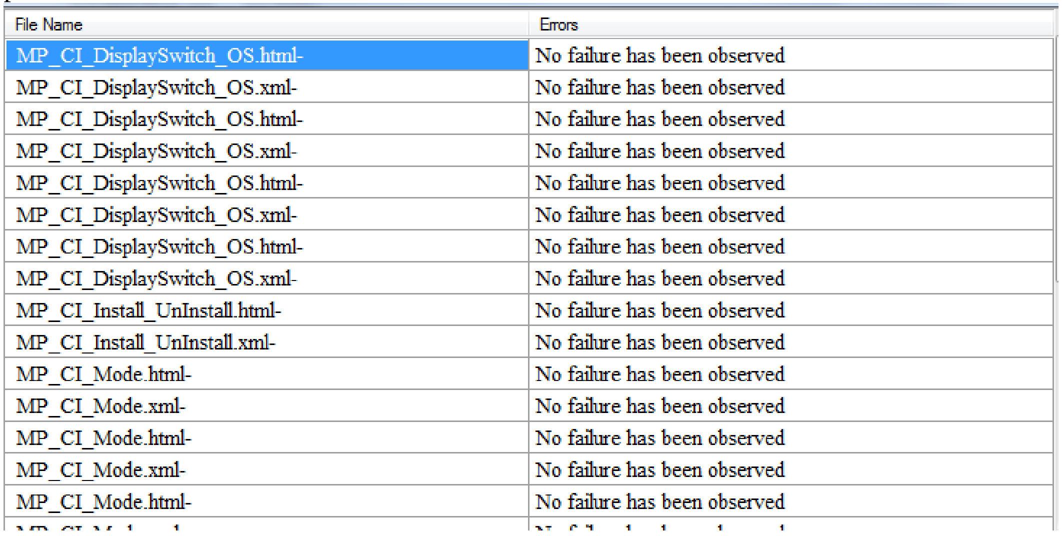


Fig 6. Failure Analysis Tool

If the failure has not occurred in test case, still user passes reference number tool. It will generate downloaded report file specifying no failure has been observed.
Such that with help of this tool a fully functional automatic test failure has been developed which will not only help user to understand but also help to rectify the particular error occurred.



File Name	Errors
MP_CI_DisplaySwitch_OS.html-	No failure has been observed
MP_CI_DisplaySwitch_OS.xml-	No failure has been observed
MP_CI_DisplaySwitch_OS.html-	No failure has been observed
MP_CI_DisplaySwitch_OS.xml-	No failure has been observed
MP_CI_DisplaySwitch_OS.html-	No failure has been observed
MP_CI_DisplaySwitch_OS.xml-	No failure has been observed
MP_CI_DisplaySwitch_OS.html-	No failure has been observed
MP_CI_DisplaySwitch_OS.xml-	No failure has been observed
MP_CI_Install_UnInstall.html-	No failure has been observed
MP_CI_Install_UnInstall.xml-	No failure has been observed
MP_CI_Mode.html-	No failure has been observed
MP_CI_Mode.xml-	No failure has been observed
MP_CI_Mode.html-	No failure has been observed
MP_CI_Mode.xml-	No failure has been observed
MP_CI_Mode.html-	No failure has been observed

Figure 7. Output of Failure Analysis Tool

4. Tools Required

Following are the tools required to build up whole setup of application.

Pre-Silicon Requirements

Hardware Specification

1. Cherry or Bay trail dedicated processor
2. RAM - 4GB
3. Memory – 20 GB

Software Specification

1. Python Interpreter
2. Server Based Software
3. Shared Executable Binaries

Post –Silicon Requirements

Hardware Specification

1. Machine pool of bay trail
2. RAM for each – 4 GB
3. Memory for each -30 GB

Software Specifications

1. Intel HD Graphics Setup
2. Machine Pool Server
3. Tool for sending request to run test cases

Bay Trail- Intel's existing 22nm Bay Trail CPUs for use in 64-bit tablets and laptop with dedicated graphics processor.
Cherry Trail- 14nm Cherry Trail Atom CPUs with graphics processor aimed at tablets and low-end PCs/convertibles, should allow Intel to take a manufacturing process lead over ARM-based mobile processor rivals. 14nm ARM-based parts are expected to arrive in late 2015.

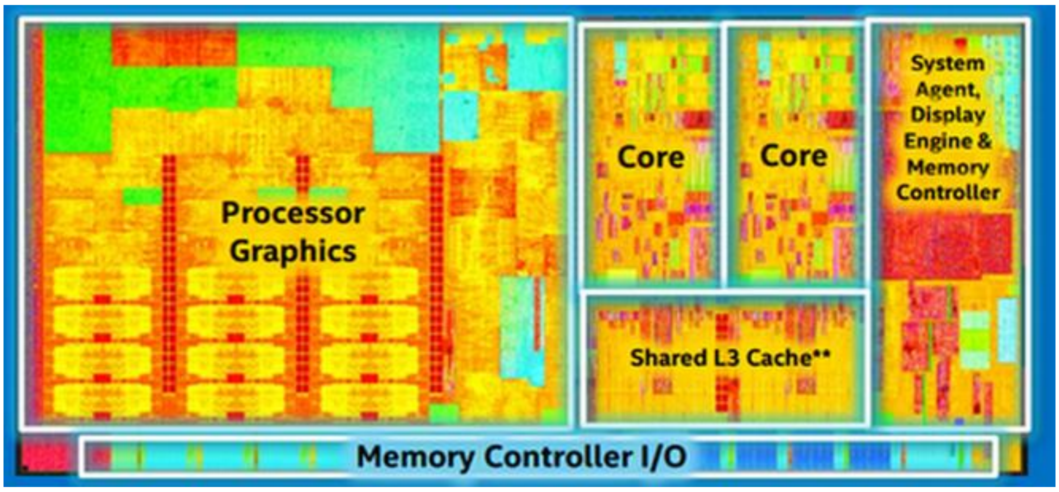


Fig 8. Design of Processor with Graphics

Graphics Behavior Checker Tool
Hardware Specification

1. Machine pool of machine
2. RAM for each – 1 GB
3. Memory for each -10 GB

Software Specifications

1. Intel HD Graphics Driver Log File
2. Golden sequence Rule File
3. Python Interpreter

Failure Analysis Tool
Hardware Specification

1. Dedicated Server of Reference Number Storage Files
2. RAM for each – 4 GB
3. Memory for each -20 GB

Software Specifications

1. Perl software
2. Visual Studio 2013
3. C# Encryption Package

5. Conclusion

A fully functional pre-silicon and post silicon environment is setup such that will handle test case running in pre and post silicon mode. Further, a sequence checker will ensure that the driver is following the predefined log and failure analysis will help to debug the system if any error occurred. Therefore it presents a robust framework of pre-silicon and post-silicon validation automation framework. It will improve the performance of the all over test procedure in any environment.

References

[1] Abramovici, M., P. Bradley, K. N. Dwarakanath, P.Levin, G. Memmi and D. Miller, “A Reconfigurable Design-for-Debug Infrastructure for SoCs,” Proc. Design Automation Conf.,pp. 7-12, 2006.
[2] Anis, E., and N. Nicolici, “On Using Lossless Compression of Debug Data in Embedded Logic Analysis,” Proc. Intl. Test Conf., 2007.
[3] Anis E., and N. Nicolici, “On By-passing Blocking Bugs during Post-silicon Validation,” Proc. European Test Symp., pp.69-74, 2008.
[4] Bardell, P.H., W.H. McAnney and J. Savir, Built-In Test for VLSI: Pseudo-random Techniques, John Wiley & Sons, 1987.
[5] Barrett, C., R. Sebastiani, S. A. Seshia, and C. Tinelli, “Satisfiability Modulo Theories,” Handbook of Satisfiability, IOS Press, 2009.
[6] Bayazit, A.A., and S. Malik, “Complementary Use of Runtime Validation and Model Checking,” Proc. Intl. Conf. CAD, pp. 1052-1059, 2005.
[7] Boule, M., and Z. Zilic, “Incorporating Efficient Assertion Checkers into Hardware Emulation,” Proc. Intl. Conf. Computer Design, pp. 221-228, 2005.
[8] Brayton, R.K., et al., Berkeley Logic Synthesis and Verification Group, ABC: A System for Sequential Synthesis and Verification. http://www.eecs.berkeley.edu/~alanmi/abc/
[9] Bryant, R.E., “Graph-based Algorithms for Boolean Function Manipulation,” IEEE Trans. Computers, Vol. C-35, No. 8, pp. 677-691, Aug. 1986.
[10] Caty, O., P. Dahlgren and I. Bayraktaroglu, “Microprocessor Silicon Debug Based on Failure Propagation Tracing,” Proc. Intl. Test Conf., pp. 293-302, 2005.
[11] Chang, K.H., I.L. Markov and V. Bertacco, “Automating Post-silicon Debugging and Repair,” IEEE Computer, Vol. 41, No.7, pp.47-54, July 2008.
[12] Clarke, E.M., O. Grumberg and D. Peled, Model Checking, MIT Press, 2000.
[13] Dahlgren, P., P. Dickinson and I. Parulkar, “Latch Divergency in Microprocessor Failure Analysis in
[14] MicroprocessorFailure Analysis,” Proc. Intl. Test Conf., pp. 755-763, 2003.
[15] de Paula, F.M., M. Gort, A.J. Hu, S.E. Wilton and J.Yang, “BackSpace: Formal Analysis for Post-Silicon Debug,”Proc. Intl. Conf. Formal Methods in CAD, 2008.

[16] Eichelberger, E.B., and T.W. Williams, “A Logic Design Structure for LSI Testability,” Proc. Design Automation Conf., pp. 462-468, 1977.

[17] Eldred, R.D., “Test Routines based on Symbolic Logic Statements,” Journal ACM, Vol. 6, No. 1, pp. 33-37, Jan. 1959.

[18] Hamzaoglu, I., and J.H. Patel, “Reducing Test Application Time for Full Scan Embedded Cores,” Proc. Intl. Symp. Fault-Tolerant Computing, pp. 260-267, 1999.

[19] Heath, M.W., W.P. Burleson and I.G. Harris, “Synchro- Tokens: Eliminating Nondeterminism to Enable Chip-Level Test of Globally-Asynchronous Locally-Synchronous SoC’s,” Proc. Design,
Automation and Test in Europe, pp. 1532-1546, 2004.

[20] Iyer, R.K., N. Nakka, Z. Kalbarczyk and S. Mitra, “Recent Advances and New Avenues in Hardware-Level Reliability Support,”IEEE MICRO, Vol. 25, No. 6, pp. 18-29, Nov.-Dec. 2005.

[21] Josephson, D., S. Poehlman and V. Govan, “Debug Methodology for the McKinley Processor,” Proc. Intl. Test Conf., pp.451-460, 2001.

[22] Josephson, D., “The Good, the Bad, and the Ugly of Silicon Debug,” Proc. Design Automation Conf., pp. 3-6, 2006.

[23] Keshava, K., N. Hakim and C. Prudvi, “Post-Silicon Validation Challenges: How EDA and Academia Can Help,” Proc.Design Automation Conf., 2010.

[24] Ko, H.F., and N. Nicolici, “Automated Trace Signals Identification and State Restoration for Improving Observability in Post-silicon Validation,” Proc. Design Automation and Test in Europe, pp. 1298-1303, 2008.

[25] Ko, H.F., A.B. Kinsman and N. Nicolici, “Distributed Embedded Logic Analysis for Post-silicon Validation of SOCs,” Proc.Intl. Test Conf., 2008.

[26] Koenemann, B., “LFSR-Coded Test Patterns for Scan Designs,” Proc. European Test Conf., pp. 237-242, 1991.

[27] Krstic, A., W.C. Lai, K.T. Cheng, L. Chen and S. Dey, “Embedded Software-Based Self-Test for Programmable Core-Based Designs,” IEEE Design and Test of Computers, Vol. 19, No. 4, pp.
18-27, July-Aug. 2002.

[28] Li, Y., Y.M. Kim, E. Mintarno, D.S. Gardner and S. Mitra, “Overcoming Early-Life Failure and Aging Challenges for Robust System Design,” IEEE Design and Test of Computers, Vol. 26, No. 6,
pp. 28-39, Nov.-Dec. 2009.

[29] Li, W., A. Forin and S. A. Seshia, “Scalable Specification Mining for Verification and Diagnosis,” Proc. Design Automation Conf., 2010.

[30] Liu, X., and Q. Xu, “Trace Signal Selection for Visibility Enhancement in Post-Silicon Validation,” Proc. Design Automation and Test in Europe, pp. 1338-1343, 2009.

[31] Ma, S.C., P. Franco and E.J. McCluskey, “An Experimental Test Chip to Evaluate Test Techniques: Experimental Results,” Proc. Intl. Test Conf., pp. 663-672, 1995.

[32] Malik, S., and L. Zhang, “Boolean Satisfiability: From Theoretical Hardness to Practical Success,” Communications of the ACM, Vol. 52, No. 8, pp. 76-82, Aug. 2009.

[33] Mitra, S., and K.S. Kim, “X-Compact: An Efficient Response Compaction Technique for Test Cost Reduction,” Proc. Intl. Test Conf., pp. 311-320, 2002.

[34] Mitra, S., and K.S. Kim, “X-Compact: An Efficient Response Compaction Technique,” IEEE Trans. CAD, Vol. 23, issue 3, pp. 421- 432, March 2004.

[35] Mitra, S., “Robust System Design,” Proc. Intl. Conf. VLSI Design, pp. 434-439, 2010.

Moore, J., T. Lynch, and M. Kaufmann, “A mechanically checked proof of the AMD5 K86(TM) floating-point division program,” IEEE Trans. Computers, Vol. 47, No. 9, pp. 913-926, Sept. 1998.

[36] Naffziger, S., B. Stackhouse, T. Grutkowski, D.Josephson, J. Desai, E. Alon and M. Horowitz, “The Implementation of a 2-core, Multi-threaded Itanium Family Processor,” IEEE Journal Solid-State Circuits, Vol. 41, No. 1, pp. 197-209, Jan. 2006.

[37] Park, S.B., and S. Mitra “IFRA: Instruction Footprint Recording and Analysis for Post-Silicon Bug Localization in Processors,” Proc. Design Automation Conf., 2008.

[38] Park, S.B., T. Hong and S. Mitra, “Post-Silicon Bug Localization in Processors using Instruction Footprint Recording and Analysis (IFRA),” IEEE Trans. CAD, Vol. 28, Issue 10, pp. 1545-1558, Oct. 2009.

[39] Park, S.B., and S. Mitra, “Post-silicon Bug Localization for Processors using IFRA,” Communications of the ACM, Vol. 53, No. 2, pp. 106-113, Feb. 2010.

[40] Park, S.B., A. Bracy, H. Wang and S. Mitra, “BLoG: Post-Silicon Bug Localization in Processors using Bug Localization Graphs,” Proc. Design Automation Conf, 2010.

[41] Parvathala, P., K. Maneparambil and W. Lindsay, “FRITS – A Microprocessor Functional BIST Method,” Proc. Intl. Test Conf., pp. 590-598, 2002.

[42] Patra, P., “On the Cusp of a Validation Wall,” IEEE Design and. Test of Computers, Vol.24, No.2, pp.193-196, March- April 2007.

[43] Pnueli, A., “The Temporal Logic of Programs,” Proc. Symp. Foundations of Computer Science, pp. 46-57, 1977.

[44] Sarangi, S., B. Greskamp and J. Torrellas, "CADRE: Cycle-Accurate Deterministic Replay for Hardware Debugging," Proc. Dependable Systems and Networks, pp. 301-312, 2006.
[45] Seshia, S.A., W. Li and S. Mitra, "Verification-Guided Soft Error Resilience," Proc. Design Automation and Test in Europe, pp. 1442-1447, 2007.
[46] Shen, J., and J. A. Abraham, "Native Mode Functional Test Generation for Processors with Applications to Self Test and Design Validation," Proc. Intl. Test Conf., pp. 990-999, 1998.
[47] Silas, I., I. Frumkin, E. Hazan, E. Mor and G. Zobin, "System-Level Validation of the Intel Pentium M Processor," Intel Technology Journal, Vol. 7, No.2., pp. 37-43, May 2003.
[48] Siewiorek, D.P., and R.S. Swarz, Reliable Computer Systems: Design and Evaluation, A.K. Peters, 1998.
[49] Vardi, M., "From Church and Prior to PSL," Proc. 25 Years of Model Checking, pp. 150-171, 2008.
[50] Wagner, I., V. Bertacco and T. Austin, "Using Field- Repairable Control Logic to Correct Design Errors in Microprocessors," IEEE Trans. CAD, Vol. 27, Issue 2, pp.380- 393, Feb 2008.
[51] Williams, M.J.Y., and J.B. Angell, "Enhancing Testability of Large Scale Integrated Circuits via Test Points and Additional Logic," IEEE Trans. Computers, Vol. C-22, Issue 1, pp. 46-60, Jan. 1973.
[52] Yerramilli, S., "Addressing Post-Silicon Validation Challenge: Leverage Validation and Test Synergy," Keynote, Intl. Test Conf., 2006.

Author

Imran Ahmed, He has done post-graduation at VIT University Vellore, Tamil Nadu in Computer Science and Engineering stream. He has done Bachelor of Technology in Computer Science and Engineering from University Institute of Engineering and Technology, CSJM University, Kanpur. His major interest work area is Automation, Computer Graphics, Cloud Computing, Artificial Intelligence and Web development.