FoCuS: A Dynamic Regression Suite Generation Platform for Processor Functional Verification

Amir Hekmatpour
IBM Microelectronics
ASIC Development
Research Triangle Park, NC U.S.A 27709
rima@us.ibm.com

James Coulter
IBM Microelectronics
Processor Development
Research Triangle Park, NC
jcoulter@us.ibm.com

Azadeh Salehi
IBM Microelectronics
ASIC Development
Research Triangle Park, NC
asalehi@us.ibm.com

Abstract

Functional verification of complex integrated circuits such as processors utilize a large number of tests. To manage the verification resources and reduce redundancies optimized regression suites are required. We describe a platform for generating coverage-based regression test suites that can be simulated to verify the changes and enhancements to a system (hardware, software) to ensure that the design does not violate the desired functionality. The web-based platform called FoCuS provides integrated regression suite generation, optimization and management. Application of FoCuS to advanced processor verification has shown consistent regression suite quality and size improvement over existing greedy and ad-hoc methods.

1 Introduction

Verification of a complex integrated circuit design is an iterative process where the major features are tested on a continuous basis for the duration of the design. To alleviate the burden and to manage the increasing task of verification, regression testing is adopted. Regression testing involves the repetitive testing of a design or an application's major features with a suite of tests. Regression test suites are necessary to ensure that changes to the system made as a result of bug fixes or design changes have not "broken" something previously tested. Developing an optimized regression suite is desirable to provide the highest verification coverage with minimum simulation time and resources. Such a regression suite would allow design and verification engineers to rapidly reverify the design changes and quickly respond to new function requirements and/or what-if analysis.

A platform is presented for generating an optimized regression suite based on a set of user defined functional coverage attributes and simulation characteristics. We provide a language for modeling the desired regression attributes and selecting one or more regression algorithms for optimizing the generated regression suite. As such, the platform and its associated methods and algorithms can be described as a coverage-directed system for algorithmic regression test suite generation and optimization. Although most references and examples used in this paper are related to processor and integrated circuits functional verification, the techniques, algorithms, and methodology are applicable to any field that requires functional and integrity validation during the design and development.

An instance of the described regression system has been developed and deployed at the IBM PowerPC Embedded Processor design group at IBM RTP. An overview of the platform system architecture known as FoCuS is shown in Figure 1. The FoCuS platform is used for collection, analysis and optimization of randomly generated test programs utilized in the functional verification of advanced PowerPC Embedded Processors. The results show consistent regression suite quality and size improvement over the traditional ad-hoc regression generation. The major advantages of the platform described in this paper are reduced processing cost and improved user control over regression customization. The regression modeling language enables managing the size of regression suites and provides an effective method for influencing and customizing the content of the regression suites based on the important characteristics of the system.

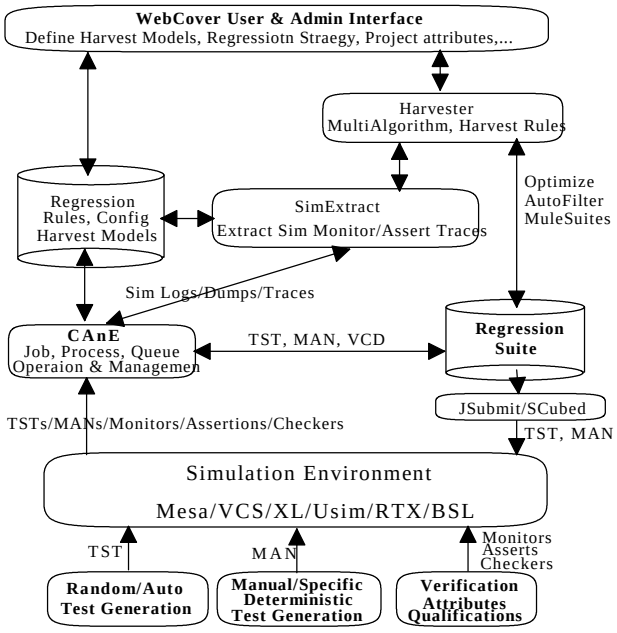

Figure1: Platform System Architecture

The FoCuS platform, its associated algorithms, and control and management utilities are designed for operation in a 24/7 environment where new tests are continuously generated and simulated. The system will collect and analyze the simulated tests based on the

harvesting models and regression strategy and then optimize the resulting regression suite based on the selected algorithm(s), as shown in Figures 1, 2 and 3.

Users can customize their regression strategy by selecting the desired algorithms, the maximum size of the regression suite, the density threshold, as well as the regression optimization attributes (Figure 5). The data flow and functional block diagram of the regression generation and optimization process described in this paper are shown in Figure 1. The tightly coupled regression, harvesting, job operation and management described in this paper can support a diverse combination of regression and harvesting strategies (Figures 2 and 3) and can be easily extended to any iterative design and verification field.

2 Previous and Related Work

Functional verification is the major task in integrated circuit design. It is estimated that over 70% of the resources and development-cycle of a processor is used for its functional verification. Due to increasing market pressure on the design turn around cycle, the functional specification, architectural definition and design and verification are conducted in parallel, or at least with a large overlap. Therefore it is not practical to generate a regression suite from specification as suggested by [1]. Optimization methods applied to static regressions such as [2] and [3] are not applicable here either. The platform described in this paper does not test a specific hardware or software design as is the case with [4] or compact existing regression suites as reported in [5] and [6].

Considering all the above, it is reasonable to say that regression generation and management in the field of processor verification is one of the most demanding and time/cost sensitive applications of the regression test suite concept. In addition, considering the large number of EDA companies involved in development and support of functional verification tools and methodologies, it is a good benchmark to use for understanding the current state of the art and for comparison to the described platform. In the remainder of this section we will review the state of the art tools and techniques available for generating, optimizing and managing regression suites.

2.1 Meteor from IBM Haifa Research Lab

Meteor and MeteorLite coverage analysis and regression generation tools developed at IBM Research Lab in Haifa provide a greedy regression test suite generation feature. They can be classified as an “All Hit Greedy” algorithm similar to [6] and [7]. Any test that hits one or more of the predefined coverage metrics is added to the regression suite. Meteor does not provide any information on the signature of the regression - what test hit what and when. No regression optimization is provided other than some test sequence reordering similar to [8]. Once you turn the regression on, any test which hits a coverage model is added to the regression suite. Meteor and MeteorLite are better suited for functional coverage modeling and analysis rather than regression suite generation.

2.2 BugSpray

BugSpray is a verification and coverage analysis environment developed by IBM server group in Austin [9]. The user defines design attributes deemed important or interesting to be monitored. During the simulation, these user defined assertions create messages that identify whether the event defined occurred or the design behavior was exercised. BugSpray parses these messages and based on user specification, decides whether or not to keep the test. No optimization or analysis of the regression suite is provided. Since assertions and their corresponding regression attributes are defined within the design HDL, it requires some level of familiarity with the design to make any changes and/or turn it off. BugSpray regression features are not extensible and can not be used outside of BugSpray environment. The regression strategy is fixed and not accessible to the user.

2.3 Specman

Specman from Verisity [10] provides means for a user to define characteristics of test benches required. It then generates the tests, runs them through simulation and generates coverage reports telling the user which of the attributes and events were covered. Specman does not provide a set of tests but rather a set of test specifications. To regenerate the same test or set of tests, their corresponding test specification and generation seeds have to be rerun. So, the entire regression suite has to be regenerated each time it is run. Specman does not provide any mechanism for the user to define functional verification regression generation criterion or any way to optimize it. As such, Specman and its regression methods do not apply to the repetitive functional testing required by the incremental specification and design of complex integrated circuits.

2.4 0-In Checklist/CheckWare

0-In does not provide any specific regression test suite generation or optimization capability but it advertises that if the user defines a set of interesting assertions, he can use 0-In structural coverage to analyze an existing regression suite against the assertions and identify missing tests. 0-In, like several other EDA vendors promotes a combination of semi-formal and assertion-based verification for functional verification of integrated circuits but does not provide any regression capability in order to reduce the task in many iterations of design/verification required for any design of reasonable complexity. [11]

3 Coverage-based Regression Test Suites

FoCuS generates and optimizes a regression test suite from among all tests simulated in the verification of a hardware or software system to maximize the overall density and effectiveness of the test suite with the smallest number of tests as possible. The density of a regression suite is measured as its functional coverage or the total number of interesting design attributes that are verified when the regression is simulated. In addition to the coverage density, the resources required for generating, optimizing, storing and re-simulating the regression are also taken into consideration. The regression suite with the maximum coverage and minimum cost is desired in order to quickly and inexpensively verify a design after each major upgrade, redesign or design check point.

The FoCuS platform and its associated methods work well with environments where a large number of random or automatically generated tests are simulated and the quality and progress of the verification are measured based on the actual functional coverage of the design. This platform is also suitable for any verification environment where the design attributes to be verified (or attributes that are observed when a verification task is successful) can be defined and some form of test program or stimulus is available to drive the model of the design or emulate its function. Figure 2 shows a sample processor functional verification regression environment deployed with FoCuS.

3.1 Regression Suite Granularity

FoCuS supports generation of a regression suite based on any number of design attributes and verification environment characteristics. For example, one team may want to maintain separate regressions for each unit or sub-unit and even apply different algorithms to each strategy. In some cases it might be necessary to collect and maintain separate regressions based on the type or source of tests. This might be necessary if the simulation environment is different for each test type. Yet in another case, a regression might be necessary for each design level in order to ensure the design integrity before promoting the design to the next level.

The tightly coupled regression, harvesting and simulation job management supported by FoCuS (Figure 1) in conjunction with its customizable regression environment (Figure 5) and the extensible harvesting language (Figure 6) can support a diverse combination of regression and harvesting strategies. The following are some examples of the regression granularity supported for one of the IBM Embedded PowerPC projects. (as shown in Figure 2)

- Unit (IPU, IFU, LSU), Sub-Unit (IFU-IEA)
- Test Type (TST, MAN, ...)
- Simulation model (major model releases) or test generators (GPro, FPGen, Piparazzi, ...)
- Design check points, RIT (Pass1, Pass1.1,...), tool releases (start a new regression for major release of the test generator).

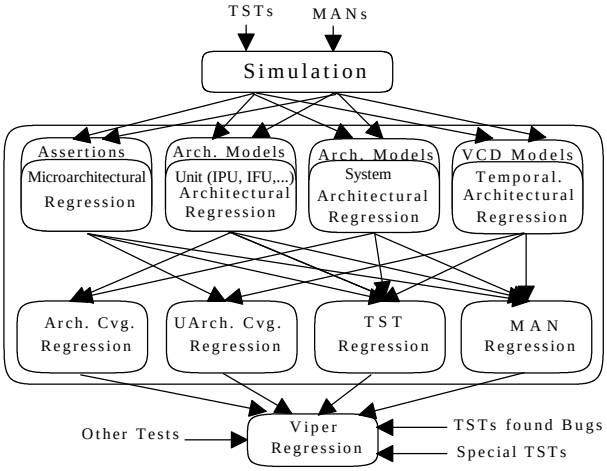

Figure 2: Hierarchical Regression Generation

4 Multi-Algorithm Hierarchical Regression

In complex design environments, different test generators and perhaps different simulation environments are utilized at different levels or parts of the design. This requires the flexibility to define different regression strategies for each unit verification (Figure 2). A high volume, highly automated test generation and simulation environment might require rapid and continuous test analysis at the first level of regression generation which would include a large number of tests. A more focused and elaborate selection and optimization could be deployed as the various mini-regression suites are merged and there is more time for analysis and optimization.

When devising a regression strategy, FoCuS allows taking into consideration the verification environment characteristics and how the regression suite will be used in the verification process. Various regression strategies can be applied at different levels of design hierarchy. As the design progresses, different regression algorithms may be utilized to better manage the verification needs. One such multi-algorithm hierarchical strategy is shown in Figure 3.

4.1 Regression Algorithms

The FoCuS platform supports a number of customized regression algorithms, as well as other optimized general purpose regression algorithms. Different versions of each algorithm (Normal, Vectorized, Sorted,...) are available which can be evaluated to identify the version suitable to a specific verification task based on the processing speed, storage requirement, and number of tests. For example, the overall size of the algorithm database tables and/or corresponding tests, database query and access delays, network load, and the overall algorithm processing time

can be taken into consideration in deciding which algorithms and what version of each, to be utilized in the regression strategy. The platform currently supports the following algorithms and their variations.

- Optimized Greedy Algorithm
  - First hit single pass greedy regression
  - Vectorized first hit single pass greedy regression
  - Sorted first hit single pass greedy regression
- Backward elimination Algorithm
  - Best hit multi-pass backward elimination
  - Vectorized best hit multipass backward elimination
  - Sorted best hit multi-pass backward elimination
  - Sorted best hit sum backward elimination
- Weighted hit algorithm
  - Normalized weighted hit
  - Sorted normalized hit
- Maximum likelihood (proximity) algorithm
- Normalized weighted hit sum algorithm

5 Regression Strategy

A regression strategy is the combination of harvesting models, regression algorithms and regression suite build/merge/reset choices. A team lead defines a regression strategy for a project via the FoCuS web interface WebCover (Figure 1). In making the selections that define the overall regression strategy, design attributes, design verification environment characteristics, simulation environment attributes, server farm capacity, network and disk storage capacity, as well as the frequency of design releases (requiring re-simulating the regression suite) have to be taken into consideration to define an appropriate overall regression strategy.

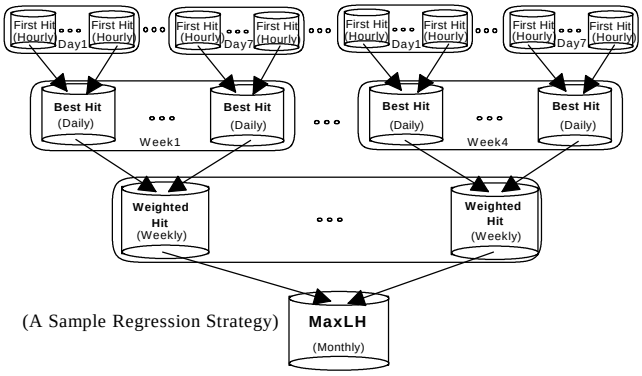

Figure 3: Hierarchical Regression Suite Generation/Optimization

Any such strategy should be revisited regularly and adjusted based on the actual behavior of the design and its simulation environment. Various reports are available from WebCover to get a better insight into the progress of the verification, as well as resource utilization. The following sections describe some of the major aspects of defining a regression strategy and give some sample data for tradeoff analysis.

5.1 Regression Suite Size Considerations

When choosing a regression algorithm and defining the overall harvesting and regression strategy, several factors have to be considered, such as the frequency of incoming tests, the average size of each test, the number of regression attributes to be analyzed, the frequency of regression suite merges, and the size and number of simulation traces to be maintained, etc..

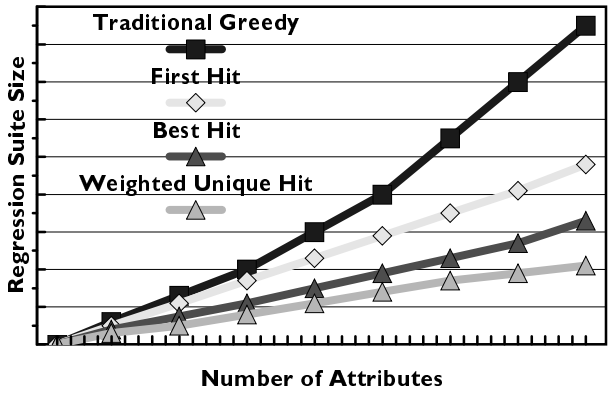

Figure 4. Regression Suite Size vs. # Regression Attributes

5.2 Algorithm Selection Considerations

Selecting which algorithms to run and how frequently and at what level of regression hierarchy depends on many factors such as storage requirements for each level of regression, processing resource requirements for each collection and whether there is any interdependency between the regression attributes (i.e. a test dropped at one level may become relevant in another level or with a different algorithm). In addition to the specific requirements of a project, the regression algorithm behavior, the processing cost as a function of test suite size and the number of optimization attributes as shown in Figure 4, have to be taken into consideration.

The FoCuS platform monitors and collects a variety of statistics on the actual behavior of the active algorithms, the simulation environment, regression suites and utilized resources. These reports, as well as the actual regression suite profile are available to the user during the strategy evaluation and tradeoff (what-if) analysis. Project operations and throughput statistics are also collected and various reports are available to users for strategy planning and optimization. Users can access these reports from WebCover to make strategy modifications according to the actual project behavior.

5.3 Algorithm Evaluation Examples

Users can provide project information to the regression front-end (WebCover) by completing predefined project templates and then selecting an initial regression strategy based on the overall disk storage, CPU processing resources and statistics generated by WebCover. An example of one such overall cost (storage + processing

requirements) evaluation process for three different algorithms is provided in this section. The sample project utilizes automatic random test generation/simulation where random and targeted tests are generated and simulated on a dedicated server farm. It is assumed that the verification environment can generate and simulate 100K random tests each day. In choosing the most appropriate regression algorithm, not only the final size of the regression is taken into consideration but also the amount of file transfers between the simulation environment and the regression suite, as well as additional processing required for regression optimization and management..

Test generation = 100K tests/day = ~4K tests/hour
Average size of each test program = ~200KB
Average simulation log size = ~500KB
Test transfer stage1 = simulator => Harvester
= 100K * 200KB = 20 GB/day = ~ 1GB/hour
Sim log transfer stage1 = simulator => Analyzer
= 100K * 500KB = 50 GB/day = ~2 GB/hour
Test generation time (average) = 30T (T= Time unit)
Test simulation time (average) = 10T
Test file transfer delay (simulator => harvester) = 1T

In the following sample scenarios, the optimization percentages (how much of simulated tests are discarded) and the required processing time for each algorithm are estimates. The actual numbers are dependent on test size, simulation model and verification environment. Each scenario includes seven daily regression suites, which are combined and optimized at the end of the week to generate a single weekly regression suite. This second round optimization rate is usually much lower than the first round. Regression processing time is total CPU time used for analysis and optimization of daily and weekly suites.

5.3.1 First Hit Greedy Algorithm

- Regression optimization = 50%
- Daily file transfer = 70 GB/day * 7 days =490 GB/week
- End of week file transfer=490 GB* 50%=245 GB/week
- Total file transfer = 735 GB/week
- Regression suite size = ~300K tests, 60 GB/week
- Regression processing time=1 T/tst*1M=1M T/week
- Regression suite simulation time = 1 day

5.3.2 Best Hit Greedy Algorithm

- Regression optimization = 65%
- Daily file transfer = 70 GB/day * 7 days=490 GB/Week
- End of week file transfer=490 GB* 35%=170 GB/week
- Total file transfer = 660 GB/week
- Regression suite size = 200K tests, 40 GB/week
- Regression Processing time=2T/tst* .9M=1.8M T/week
- Regression suite simulation time = 2/3 day

5.3.3 Normalized Weighted Hit Algorithm

- Regression optimization = 80%
- Daily file transfer= 70 GB/day * 7 days =490 GB/Week
- End of week file transfer= 490 GB * 20%=98 GB/week
- Total File transfer = 588 GB/week
- Regression suite size = 100K tests, 20 GB/week
- Regression processing time=3T/tst*.8M=2.4M T/week
- Regression suite simulation time = 1/2 day

6 Harvesting Strategy and Modeling Language

Identifying, gathering and collecting tests based on some criterion defined by the user is the process of harvesting. Harvesting allows a project lead to apply a global filter to all regression strategies. Harvesting can be turned on or off for a project from the WebCover Regression Admin area. The selections are added to the project regression configuration file which is utilized by the platform back-end regression job logistics and management CAnE (Coverage Analysis Environment) to generate and schedule harvesting and regression jobs for the project. A sample configuration profile is shown in Figure 5.

```
/* VProcessor random + test benches regression */
root_dir=/afs/peps/p/coverage/viper
project=vprocessor
strategy=vprocessor1
strategy_abbrev=vp
sub_strategy_name1=ipu
sub_strategy_name=ifu
sub_strategy_name=lsu
attributes=cmonitors, vmonitors, booke_arch
test_extension=tst, man, vcd
tst_sendsimdata2coverage=true
man_sendsimdata2coverage=true
harvest=true; harvest_models=active
regression=true
regression_dir=/afs/viper/regression/p1/vpr/vp1.23
harvest_merge_alg=weighted_fit
harvest_day2week_alg=best_fit
harvest_hour2day_alg=greedy_hit2
harvest_5min2hour_alg=greedy_hit
harvest_merge_want=100
harvest_day2week_want=100
harvest_hour2day_want=50
harvest_5min2hour_want=10
log_invalid_attributes=true
harvest_queue=cov1_vp
regression_queue=cov2_vp
monitor_db_merge_frequency=30m
tst_db_merge_frequency=2h
. . .
```

Figure 5: A Sample Regression Strategy Profile

To facilitate a standard and formal method for defining harvesting criterion, a modeling language is provided, as described in the following sections. The language attribute library can be augmented for a project to provide additional and specific modeling attributes and variables.

6.1 Harvest Modeling

The regression modeling language (aka harvesting language) provides constructs for describing the harvest attributes and specific conditions and features to be

considered for ranking and harvesting tests. This model serves as a filter and is used to prune the tests forwarded to the active regression algorithms for analysis. Figure 6 shows a sample harvesting model template that includes a partial listing of modeling constructs, operators and attributes supported by FoCuS. The harvesting language allows a user to define global criterion for test collection. Once a test has passed the simulation, it may or may not be of much value to a regression process. As shown in Figure 1, harvesting models are applied to the simulated tests to prune the rather large collection and eliminate those with little or no regression value.

```
harvest_model $MODEL_ID;
   name = $MODEL_NAME;
   version =$VERSION;
   project =$PROJID;
   user = $USERID;
   date_created = $DATE_FIRST;
   date_last_updated = $DATE_LAST;
   description = $COMMENT;
   include {
      test.type = ({$Test_Type_List});
      test.target.type = ({Target_Type_List});
      test.target.name = ({TARGET_NAME_LIST});
      test.target.count >= $COUNT;
      test.sim_env = (SIM_ENV1) or (SIM_ENV2); }
   exclude {
      test.type=($TEST_TYPE1) & (sim=$SIM1);
      test.type=($TEST_TYPE2) & (test.target.count < N);
      test.name=(WILD1*) | (*WILD2*);  }
end $MODEL_ID;
```

Figure 6: Harvest Modeling Language Syntax & Attributes

In addition to a set of language constructs, variables and operators defined, each project can extend the harvesting language attribute library. This allows domain and tool specific attributes and properties to be added to the library in order to define focused and domain-specific harvesting models. The next section shows a sample harvest model and how pruning and optimization may be conducted.

6.1.1 Harvest model as a density filter

This harvest model collects all tests that hit more than five monitors, excluding MAN tests hitting a Verilog monitor.

```
harvest_model hvm1;
    name=tst_hit_5_monitors;
    Project=vprocessor;  user=jcoulter;
    date_created=0428031609;
    date_last_update=0428031615;
    description="Harvest tests with hits> 5 monitors";
    include {
        test.type = (tst, man);
        test.target.type = (c_monitor, v_monitor);
        test.target.count >= 5; }
    exclude {
        (test.type=man;) &(test.target.type=v_monitors;)
end hvm1;
```

7 Conclusion

We described a programmable regression platform for compiling custom regression suites. The platform also known as FoCuS is tightly integrated with the test generation and simulation environments. It collects qualified tests used in the verification of a design for future regression testing. FoCuS provides a language and mechanism for modeling desired design, verification and verification environment attributes for harvesting tests. The platform provides several algorithms for optimizing the size and coverage density of the suite. A regression strategy can be defined as the collection of harvesting models, optimization algorithms, regression attributes, test generation and retiring preferences. Regression strategies can be applied to the entire design verification processes or specific design units or specific verification tasks. Application of FoCuS to advanced embedded processor verification at IBM Research Triangle Park has shown consistent regression suite quality and size improvement over other greedy and ad-hoc methods.

8 References

[1] K. Singh, J. Striegel, P. Yu, "Automatic Generation and Maintenance of Regression Test Cases from Requirements, " US Patent US6415396

[2] A. Barret, J. Baumgartner, S. Mandyam, R. Raghavan, "Efficient Method to Reduce Regression Test Suite, " IBM Technical Reports AT896-0259 v40 n3 3-97.

[3] O. Goldschmidt, D. Hochbaum, G. Yu, "A modified greedy heuristic for the Set Covering problem," Info. Processing Letters, 48(6), PP 305-310, December 1993.

[4] Y. Chen, D. S. Rosenblum, K. Vo, "TESTTUBE: A System for Selective Regression Testing," Proc. 16th International Conference on Software Engineering, IEEE Computer Society, May 1994, pp. 211-220.

[5] I. Pomeranz, L.N. Reddy, S. M. Reddy, "COMPACTTEST: A method to generate compact test sets for combinatorial circuits," Proc. International Test Conference, pp. 194-203, 1991.

[6] E. Buchnik and S. Ur., "Compacting regression-suites on-the-fly," Proceedings of the 4th Asia Pacific Software Engineering Conference, pages 385-94, December 1997.

[7] Yih-Farn Robin Chen, et al., "System and Method for Selecting Test Units To Be Rerun in Software Regression Testing, " US patent US5673387.

[8] A. Offutt, J. Pan, J. Voas, "Procedures for Reducing the Size of Coverage-based Test Sets," 12th Int. Conf on Testing Computer Software, PP. 111-123, June 1995.

[9] J. Ludden, W. Roesner, et. Al, "Functional verification of the POWER4 microprocessor and POWER4 multiprocessor systems," IBM Journal of Research and Development, Volume 46, issue 1, 2002.

[10] http://www.verisity.com/products/specman.html

[11] http://www.0-in.com/products_archer_cdv.html