Comparing Two Testbench Methods for Hierarchical Functional Verification of a Bluetooth Baseband Adaptor

Edgar L. Romero Marius Strum Wang Jiang Chau
Polytechnic School - University of Sao Paulo
Av Prof Luciano Gualberto
Trav 3, 158
+55 11 30915259
eromero@lme.usp.br strum@lme.usp.br jcwang@lme.usp.br

ABSTRACT
The continuous improvement on the design methodologies and processes has made possible the creation of huge and very complex digital systems. Design verification is one of the main tasks in the design flow, aiming to certify the system functionality has been accomplished accordingly to the specification. A simulation based technique known as functional verification has been followed by the industry. In recent years, several articles in functional verification have been presented, focusing either on specific design verification experiments or on methods to improve and accelerate coverage reaching. In the first category, the majority of the papers are aimed to processors verification, while communication systems experiences were not such commonly reported. In the second category, different authors have proposed methodologies, which need an extensive and complex work by the verification engineer on tuning the acceleration algorithms to the specific design. In the present paper, we present a functional verification methodology applied to a Bluetooth Baseband adaptor core, described in SystemC RTL. Two techniques are considered, one following the traditional framework of applying random stimuli and checking functional coverage aspects; in the second one, a simple acceleration procedure, based on redundant stimuli filtering, is included. For both solutions, a hierarchical approach is adopted. We present several results comparing both solutions, showing the gain obtained in using the acceleration technique. Additionally, we show how results on a real testbench application environment correlate to the hierarchical verification approach taken.

Categories and Subject Descriptors
B.5.2 [**Register-Transfer-Level Implementation**]: Design Aids – *optimization, simulation, verification*

General Terms
Design, Reliability, Experimentation, Verification.

CODES+ISSS'05, Sept. 19–21, 2005, Jersey City, New Jersey, USA.

Keywords
Coverage Analysis, Functional Verification, Hierarchical Verification, Optimization, Verification Strategy.

1. INTRODUCTION
The continuous improvement on the design methodologies and processes has made possible the creation of huge and very complex digital systems, with a diversity of components and sub-systems involved, spurring the facets of the known design productivity challenge [1]. One of the critical aspects in a complex system development is the assurance of design quality, which has a direct impact of in its success. One of the most crucial activities in the design flow is the design verification, the certification on the correctness of a circuit in respect to its specification, i.e., its intended behavior. In the context of Intellectual Property (IP) cores design, the verification is a key element for their success in being reused [2].

Design verification has been accomplished following two principal techniques known as formal and functional verification [2]. The functional verification, also known as simulation-based type, plays a major role in the current industrial practice. Functional verification is carried out by implementations of testbench models as the one shown in Figure 1, where results of simulating a design under verification (DUV) description are compared against the results of simulating a higher level reference model, also known as golden model. This traditional test framework makes simulation-based verification easy to manage, but system designers have reported it as the most resource and time consuming phase of the design flow. The reason is the generalized adoption of random stimulation as a mandatory approach in verification; in these cases, reaching high levels of confidence in automatic testbenches requires the execution large number of testcases until some coverage criteria is complied.

Considering the importance of the functional verification and its weight in the design flows, there is a need for the development of new strategies that make possible to ease and accelerate this phase. One alternative in this direction is the coverage analysis technique, as reported by Rosenberg[3], where the testbench execution speedup is achieved by the identification of the less stimulated coverage aspects and, then, indicating the manner of changing the stimuli generation pattern. Since determining the correlation between applied stimuli and coverage aspects is not a trivial task, techniques natural from the artificial intelligence area

have been proposed to automate this analysis. Techniques based on genetic algorithms [4], Bayesian networks and Data Mining [5] have showed to reduce the stimuli application time, but needed an extensive and complex work by the verification engineer on tuning the acceleration algorithms to the specific design.

In the present paper, we propose a simple method, based on redundant stimuli filtering. It allows reducing execution time of testbenches with random stimuli, without resorting to complex algorithms. This filtering technique is based on avoiding unnecessary RTL simulation whenever a testcase generated by a stimuli source is known to be redundant.

We compared the proposed filtering method to the traditional verification framework[6] of Figure 1, by developing and executing testbenches on a complex design, a Bluetooth Baseband layer adaptor core, described in SystemC RTL. We opted for performing the functional verification on a network communication core in order to cover an application area with little verification results reported. For years, basically experiments on verification of general purpose processor had been published[4][7]; presently, the activities in other application areas has been also reported, but being only a few related to communication systems [8,9].

Besides focusing on the acceleration of the verification methodology of a Bluetooth Baseband adaptor core, a hierarchical approach is explored to increase the efficiency of the functional verification task. We present results on a verification process running large number of testcases, showing the advantages for easing of the bug finding and managing the circuit complexity. Regarding the quantitative analysis on the advantages of the proposed acceleration technique, numbers on coverage by execution time are obtained for all modules in the Bluetooth adaptor hierarchy, with time saving up to 66%.

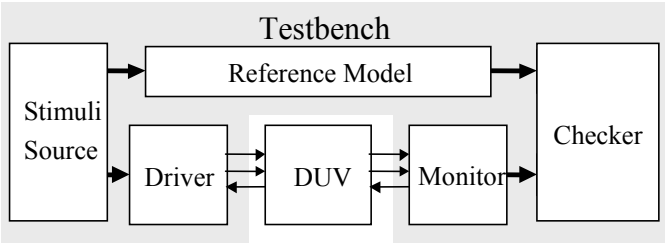

Figure 1. Basic testbench model

The subsequent sections are organized as follows: in the section 2, we present the functional verification methodology used here, with the two testbench techniques to be compared; in section 3, the Bluetooth adaptor core and its hierarchical composition is presented; in section 4, the verification strategy for the Bluetooth adaptor is presented, along with the results and discussion. Then, in section 5, we present our conclusions.

2. FUNCTIONAL VERIFICATION

2.1 Basic Functional Verification Model

In this section, we introduce some basic concepts on functional verification; more detailed description can be obtained in specific literature about this subject, as in [2]. The functional verification is based on the idea that some specification implemented at two different levels of abstraction may have its behaviors compared automatically by a tool called testbench.

The testbench architecture used in this paper follows the model presented in the Figure 1, characterized for modularity and reusability of its components. The testbench model comprises all the elements required to stimulate and check the proper operation of the Design Under Verification (DUV); in the context of this work, the DUV is a RTL description. The stimuli source, in general based on aid tools, applies pseudo-random generated testcases to both the DUV and the Reference Model, also known as Golden Model, a module with behavioral description at higher level of abstraction. The Driver and Monitor are blocks aimed to convert transaction level data to RTL signals and vice-versa. Finally, in the Checker, outputs from the simulation performed on both the Reference Model and RTL modules are compared and figures on coverage are computed and presented.

Being the functional verification an application dependent task, the designer must plan carefully aspects as the coverage model and the stimuli source. The stimuli can be classified in the following categories: directed cases, whose responses are previously known (e.g. compliance test); real cases dealing with expected stimuli for the system under normal conditions of operation; corner cases, aimed to put the system on additional stress (e.g.. boundary conditions, design discontinuities, etc.); and random stimuli, determined by using probability functions.

The coverage is an aspect that represents the completeness of the simulation, being particularly important when random stimuli are applied. Coverage may be of different types as, for instance, code, structural, functional, etc.[10], depending on which different system characteristics are to be included. Functional coverage is usually considered the most relevant type because it represents directly the objectives of the verification process. However, since it is infeasible to explore all the functionality of a complex circuit through simulation, there is not a firm definition for the verification community on what amount of simulation on a testbench may correspond to an acceptable coverage. Sometimes the coverage is limited by project deadlines.

Knowing which metrics should be included as coverage measure is dependent on the verification engineer expertise. To deal with the complexity of the problem, some generic steps may be followed, in an approach known as item functional coverage. First, a judicious selection must be made on a set of parameters associated to input and output data, for instance, the size of packets, words with specific meaning, as keys, passwords, etc. Then, for every selected parameter, the designer must form groups defined by ranges of values it may assume, following a distribution considered relevant. The 100% coverage level is established by a sufficient amount of items per group, i.e., testcases, whose corresponding applied stimuli and observed responses match the parameter group characteristics. The larger the number of items is considered, the stronger the functional verification process will be.

2.2 Coverage Analysis

Even though there is no standard method to perform the functional verification and every tool provider establishes its own approach, some main activities may be considered in the verification process. Hekmatpour[11] suggests that the functional

verification should be carried out by following several phases as planning, reference model implementation, coverage analysis, among others. The coverage analysis, to be done after or during the testbench application, is an important phase for certifying the testbench robustness. After the testbench application, in case of evidence of coverage holes, the stimuli generation should be redirected and the verification should restart until no missing coverage aspects are found.

Coverage analysis activities may be done during the testing in order to speed up the coverage fulfillment. It is known that, under random stimuli, the coverage evolution, in terms of time, presents a fast growth in the initial phase of the testbench application, followed by a saturation tendency when higher levels of coverage are reached, due to increased occurrence of redundant stimuli. In this context, redundant stimulus is defined as a pattern that does not represent any increment to the defined coverage; for the case of item functional coverage, it is part of parameter groups, which have already reached 100% coverage

The functional coverage saturation effect has motivated, in recent years, researches on using the information on functional coverage status to adjust the stimuli generation, in techniques known as "closed-loop testbenches" or "reactive testbenches". The establishment of the relationship between the reached coverage and the stimuli generation for a DUV is a complex task, requiring the application of techniques natural of the artificial intelligence area, such as genetic algorithms [4], data mining and Bayesian networks [7]. Although these approaches have had excellent reported results regarding the testbench execution time, they also have showed to require an extensive work to be implemented and set up.

2.3 Stimuli Filtering Technique

In order to reduce the testbench execution time, under random stimuli, without resorting to complex methodologies, we developed a simple closed-loop technique based on stimuli filtering. This technique is based on the observation that simulating the Reference Model is much faster than performing it on the RTL model of the DUV. The long RTL simulation may be avoided if it is known that redundant testcases were generated by stimuli source. This is accomplished as follows: first the reference model is executed and, then, by analyzing the results, it is verified if the coverage parameter groups associated to each testcase have already reached to the assigned total number of items. If the testcase is not a redundant one, the DUV is simulated.

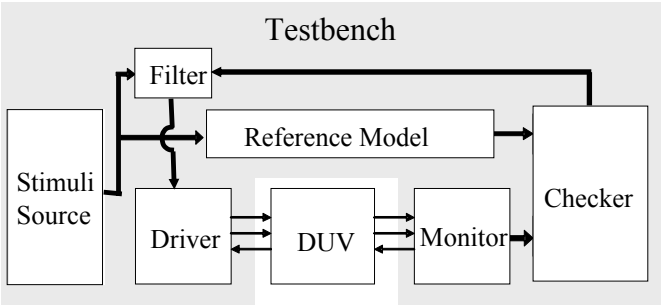

Figure 2. Filtering testbench model

The implementation of this idea consisted in modifying the basic testbench model to the one presented in the Figure 2. Other than the inclusion of the filtering block, the only change needed on the original testbench model is the implementation of an early checking of the reached functional coverage.

In the new model, the stimuli are applied to the Reference Model, whose outputs are analyzed by the Checker. The latter block sends the information about testcase´s redundancy back to the Filter, which, then, is responsible for blocking or allowing the RTL simulation. Actually, the fast Reference Model simulation and the slow RTL simulation are done serially. This method is very simple if compared to other closed-loop testbenches since there is no need to model the usually complex relationship between the coverage characteristics and the stimuli generation (source).

3. BASEBAND LAYER ADAPTOR

3.1 The Bluetooth Standard (version 1.1)

Bluetooth is an established standard for short distance wireless communications developed by the Bluetooth Special Interest Group[12]. The Bluetooth Stack, shown in Figure 3, comprises the protocol layers from the high level SDP and RFCOM to the low level radio frequency. The Bluetooth standard operates at 2.4 GHz in the ISM band (Industrial Scientific Medicine) and uses the GSFK modulation (Gaussian Frequency Shift Keying). The data transmitted has a symbol rate of 1 Ms/s. The FHSS (Frequency Hopping Spread Spectrum) technique is used to reduce the effect of radio frequency interferences on transmission quality.

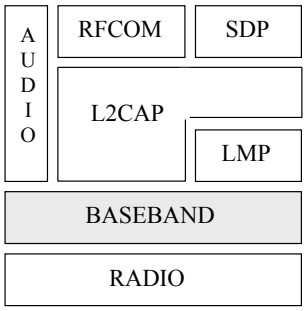

Figure 3. Bluetooth stack

The Bluetooth devices sharing the same channel form a network called piconet, where only one of them acts as master while the rest act as slaves, up to maximum of seven. The channel represents a pseudo-random hopping of RF frequencies, being the sequence determined by the Bluetooth device address of the master. The Bluetooth protocol establishes a packet with a 72 bit access code, which identifies the Bluetooth receiver unit of the packet, a 54 bit header carrying information about the packet and the payload (variable number of bits), which are information for the upper layers.

3.2 The Bluetooth core implementation

The Bluetooth core described with SystemC® 2.0.1. at RTL level implements all the Baseband functionality required to establish Asynchronous Connection-Less (ACL) links. The core scheme is presented in Figure 4; there can be observed how the modules are

organized in a hierarchical manner according its functional affinity.

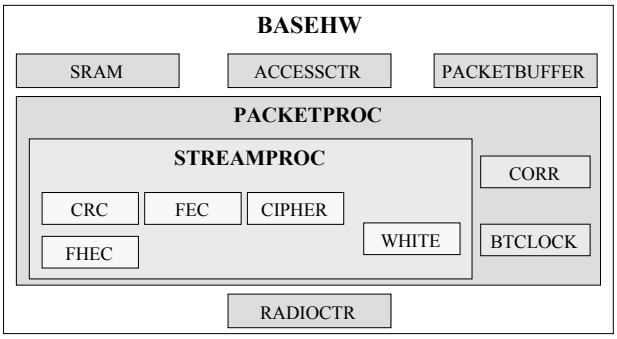

Figure 4. Bluetooth Baseband core

Each module communicates directly with modules of the same level and with the module that contains it. The inner level of the hierarchy (less global) corresponds to the bit processing modules. The CRC module detects the presence of errors in the payload bits, the FEC module corrects errors in the payload bits, the FHEC module realizes the error detection and correction in header bits, the CIPHER module encrypts the payload bits and the WHITE module reduces the DC bias of the whole packet.

At the stream level, the bit level modules are used in the STREAMPROC module to compose and decompose packet streams, the CORR module identifies the packets directed to the Bluetooth unit from all of those that come through the RF channel and send the access code to the other units and the BTCLOCK module set the clock references required to synchronize the Bluetooth unit according to the connection temporization.

In the next level, the PACKETPROC module together with CORR and BTCLOCK modules insert the streams in to the Bluetooth timing requirements. Finally, the top level includes: the PACKETBUFFER which, stores the information required to the operation of the stream and bit processes, the SRAM that acts as the configuration memory of the core and the ACCESSCTL which deals with the interfacing with both the LEON2 processor and the radio driver RADIOCTL. In the present work, the core was designed to communicate with a LEON2 processor through an AMBA™ APB interface and to a radio controller for the Bluetooth transceiver Phillips™ BGB101.

4. THE ADAPTOR VERIFICATION

4.1 The Verification strategy

In order to deal with the complexity of the whole verification process, the strategy was planned to match the Bluetooth adaptor features. Therefore, two aspects were considered for the functional verification: firstly, the development of different testbenches focused on the most relevant characteristics of the Bluetooth adaptor modules, such as the bit processing operations, the stream handling, the packet handling and the interfacing procedures; secondly, the testbenches application followed the hierarchical structure of adaptor implementation in a bottom-up way. Once less global (more local) modules have been debugged and the functionally classified as correct, the verification proceeded to more global modules.

Although the main source of applied stimuli was random, we have also considered compliance test stimuli, found on the Bluetooth specification, and real cases stimuli; however, the latter ones were only applied for the CRC and FEC verification. The random sources were strongly considered because they allow the fast creation of large number of stimuli, what is desirable for exploring of the huge number of possible combinations of the Bluetooth adaptor functionality.

The coverage approach followed in this work was the item functional coverage, as described in Section 2.1. As examples of input coverage parameters for the CRC module, we may have the packet size or the CRC initialization value, and, of output coverage parameter, the parity bits.

Based on the above strategy, the verification was performed in two different batches with different objectives: 1) to compare the basic testbench and the filtering testbench models presented in Section 2; 2) to analyze the benefits of hierarchical verification for managing complexity, by applying testbenches with very large number of stimuli.

4.2 Results on Comparing Verification Models

In order to compare the testbench model proposed for redundant stimuli filtering (closed loop) against the basic one (open loop), random stimuli based verification was performed on all RTL modules and sub-modules of the Bluetooth adaptor. Full coverage (100%) per group in each chosen parameter was set to a relatively medium number, 10,000 items, only for the purpose of this study. Results on a real testbench application environment, what took longer times, will be given in the next section.

Table 1 shows a summary of the numbers we obtained in simulating all modules under both testbench models, considering 100% coverage. Column 1 lists the tested DUVs; column 2 shows the total number of stimuli generated by the stimuli source in both models, while column 3 presents the number of redundant stimuli, which were filtered out with the acceleration method. In column 4, we have the total number of items to be applied (input parameters) and observed (output parameters) in order to reach the established 100% coverage. The last 3 columns present data on execution time. Column 5 and 6 indicate the time consumed by the testbench application under the basic and filtering models, respectively. The last column shows the relative gains in execution time. The time measures presented in Table 1 were taken running both models on a PC with a Pentium IV HT processor and with 1GB of DRAM memory.

From the last column in Table 1, we can see a strong relationship between the relative time gain and the ratio of the amount of blocked redundant stimuli in respect to the total number of generated stimuli. However, this relationship is discrepant in a few cases -StreamProc (interpret), Corr (correlate) and BaseHw- because the time gain by the filtering testbench model actually depends on the relative weight of the avoided unnecessary RTL simulation time compared to the execution times of the stimuli generation and of the reference model. The discrepant tendency is presented strongly when the reference model is complex, StreamProc (interpret) and BaseHw, or when the DUV operation is simple, Corr (correlate).

Table 1. Verification with basic and filtering testbench models, under random stimulation (10,000 items coverage)

Design under Verification	Total # Stimuli	Total # Redund. Stimuli	100% Cov. (Items)	Time1 (s)	Time2 (s)	Time Gain (%)
CRC (cod)	237,282	142,988	170,000	2,295.33	1,015.94	55
CRC (decod)	212,055	117,210	170,000	4,708.78	2,116.47	55
FEC (cod)	107,403	18,045	160,000	56.92	50.53	11
FEC (decod)	106,576	17,143	160,000	72.46	52.99	26
FHEC (cod)	267,562	173,558	250,000	247.79	101.08	59
FHEC (decod)	81,792	1,793	160,000	44.47	43.28	3
White	387,567	255,141	220,000	2,119.50	723.09	66
Cipher	171,466	82,001	500,000	10,486.64	5,648.57	46
StreamProc (compose)	75,971	12,378	300,000	19,951.62	17,733.48	11
StreamProc (interpret)	124,984	55,870	300,000	37,021.93	24,302.70	34
Corr (correlate)	53,612	10,864	200,000	5,941.51	4,914.27	17
Corr (send)	40,497	223	120,000	130.79	129.24	1
PacketProc	116,105	41,280	340,000	98,625.76	69,432.49	30
BaseHw	117,998	43,210	340,000	186,882.31	131,953.00	29

A second interesting observation from Table 1 is that there are cases where the gain is close to 50-60%, other ones about 25% and, finally, some below 10%, meaning there are relatively more redundant stimuli in some cases than others. This can be better understood with the help of figure 5, showing curves on number of redundant stimuli by (normalized) time, up to 100% coverage. It makes evident that there has been less redundant stimuli during the test of BaseHw module than during the verification of the CRC module. This fact is closely related to the amount of processing and data transformations involved, which are reflected to the modules output ports. Whenever the output data are highly correlated to the input data, the relative gain is smaller. This can be confirmed in cases like CRC, FHEC, White and Cipher, where the amount of processing is high. Another related issue is that the coverage parameters considered for a more local module will be left aside in the testbench development of a more global module which uses the first one. When disconsidering those parameters for more global modules, these ones tend to present simpler coverage aspects, less redundant stimuli and lower gains, as it is the case of modules at the bottom of the Table 1.

The base for the filtering method is the elimination of unnecessary RTL simulation cycles, due to redundant stimuli. The RTL simulation applied to the DUV is the most time consuming task in the testbench application, so its weight has led us to some other observations. As expected, more complex modules, particularly the global ones, present longer simulations. As an example from Table 1, the number of stimuli applied to the CRC modules was higher than the total number applied to the BaseHw module, although the execution time for the last one was two orders higher. Another related aspect is the importance of considering RTL simulation time in absolute values. Although there may have a relative time gain reduction for more global modules, as shown in Table 1, the absolute time gain in those cases is very significant. While for more local modules the RTL simulation time has taken about minutes, for more global modules, the simulation has taken hours. These numbers turn to be more critical considering that, in real testbench application environments, the number of items are much higher - in next section, we show testbenches with 100% coverage set as 100,000 items for each parameter group. Therefore, the absolute gain may be days and weeks in real test environments.

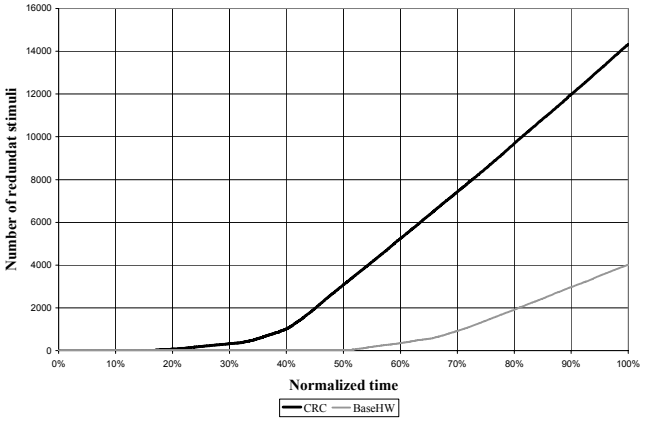

Figure 5. Redundancy comparison CRC and BaseHW

Regarding the implementation time for the closed loop testbench, the first filtering module (the CRC module) took about one day to be developed and coded. Once this module was implemented, it took about half day for each additional module. This extra time is very small if compared to the time required to the development of each module under the basic open loop model. The inclusion of automation and design aids like templates may make this task even faster.

4.3 Results on Testbench Application

In this section, we present the results of functional verification with the basic testbench model, applied to the Bluetooth adaptor modules, following a bottom-up hierarchical approach. The verification was performed with stringent coverage restrictions: under the random stimuli application, the 100% coverage was set as 100,000 items for every parameter group. The results are listed in Table 2: column 1 lists the DUVs, column 2 shows the number of random stimuli applied, column 3 shows the number of found errors and, finally, in column 4, the time required for the development and execution of the testbenches is presented. For presentation purposes, only the cases with design errors detected are listed. It is important to notice that the small number of detected design errors is due to the fact before the execution of this random stimuli testbench, cases with compliance test and real cases stimuli had already been run.

Table 2. Verification results with basic testbench model, under random stimulation (100,000 items coverage)

Module	# Stimuli	#Errors	Time (weeks)
FEC (code and decode)	2,305,455	1	2
StreamProc (compose and interpret)	1,161,392	4	4
Corr (correlate and send)	538,133	1	2
PacketProc	797,458	4	6
BaseHw	797,458	2	6
TOTAL	5,599,896	12	22

The increasing complexity and execution time for more global modules makes evident the relevance of the hierarchical verification approach as a form to fight the complexity. Let us consider that finding and correcting one design error, as in the FEC, took 2 weeks for developing testbenches and applying 2,305,455 stimuli; then, in order to find/correct the same design error using the whole adaptor as DUV, it would take a much bigger effort. As result, this strategy allowed speeding up the verification process, with errors detected in a faster and more accurate manner. The hierarchical approach also shows benefits on the coverage analysis aspect- as cited in previous section, the coverage parameters considered for a more local module will, eventually, be left aside in the testbench development of a more global module which uses the first one. It means less stimuli to be applied to the more global RTL module, reducing the time consumed by testbenches application and RTL simulation. None of the errors found during the any module verification occurred in the previously verified modules or had influence from them. This shows that the progressive scheme allows the designer to gain confidence on the behavior of the core as the verification evolves and, also, opens the possibility of reusing the inner modules for future implementations of cores.

5. CONCLUSIONS AND FUTURE WORK

The present work shows the importance of the conception of the functional verification strategy in the success of the verification process. From the coverage point of view, random stimulation it is a big source of redundant cases, i.e., stimuli that do not increase coverage. Consequently, the effective and appropriate use of random stimulation requires using techniques to modify the generation patterns according to the desired coverage. Techniques such as stimuli filtering, presented in this paper, show that important time saving can be achieved even without much computational expense or development effort.

We have presented several results of a verification plan applied to a Bluetooth Baseband adaptor core, comparing the coverage evolution under two testbench models and also reporting results in found design errors. The results have showed us that techniques to increase functional verification efficiency must consider the functional and implementation characteristics of the design, as the hierarchical structure.

As future work, the acceleration method presented in this paper should be compared to other closed loop methodologies. One may also develop verification tools as automatic verification modules generation in a template based fashion.

6. ACKNOWLEDGMENTS

This work has been supported by the Ministry of Science and Technology, through the CNPq agency, from Brazil.

7. REFERENCES

[1] Silicon Industry Association, *Silicon Technology Roadmap for Semiconductors*, 2001.

[2] J Bergeron, *Writing Testbenches: Functional Verification of HDL Model*. 2nd ed, Kluwer Academic Publishers, Boston, 2003.

[3] S. Rosenberg, *Combined Coverage Verification Speeds Verification*. *EEdesign*, 2003.

[4] M. Bose, J.Shin, and E. Rudnick, *A Genetic Approach to Automatic Bias Generation for Biased Random Instruction Generation*. Evolutionary Congress Proc on, pp 442- 448, 2001.

[5] M. Braun, W. Rosenstiel, and K. Schubert, *Comparison of Bayesian Networks and Data Mining for Coverage Directed Verification*. High Level Design Verification and Test Workshop, pp 91-95, 2003.

[6] K. da Silva, E. Melcher, G. Araújo, "Automatic Testbench Generation Tool for a SystemC Functional Verification Methodology", Proceedings of the SBCCI2004, 17th *Symposium on Integrated Circuits and Systems Design,* pp 66 –70, 2004.

[7] Z.Gu, Z. Yu, and Q.Zhang, *Functional Verification Methodology of a 32-bit RISC Microprocessor*. IEEE international conference on circuits and systems, pp 1454 – 1457, 2002.

[8] Y. Mathns, and A. Châtelain, *Verification Strategy for Integration 3G Baseband SoC*. Design and Automation Conference, IEEE, Chicago, pp 7-10, 2003.

[9] V. Fernandez, L. Berrojo, and A. Jalon, *Design, Functional Verification and Test of a MPEG2-TS Multiplexer for an On-Board Satellite Processor*. Proceedings of the XVII Conference on Design of Circuits and Integrated Systems (DCIS2002). 2002.

[10] S. Tasiran, K. Keutzer. *Coverage Metrics for Functional Validation of Hardware Designs*. IEEE Design and Test of Computers, vol 18, pp 36 – 45, 2001.

[11] A. Hekmatmpour, and J. Coulter, *Coverage-Directed Management and Optimization of Random Functional Verification*. Proceedings International Test Conference, vol 1, pp 148 – 155, 2003.

[12] *Bluetooth Specification*, version 1.1 , 2001. www.bluetooth.org (at March 2005).