A Functional Coverage Approach for Direct Testing: An Industrial IP as a Case Study

Sameh El-Ashry
Mentor Graphics
Alexandria, Egypt
Sameh_Mahmoud@mentor.com

Khaled Salah
Mentor Graphics
Cairo, Egypt
Khaled_Mohamed@mentor.com

***Abstract*— Tracking the test-plan progress of the direct testing methodology is a manual process. If the test-plan is complicated, the manual tracking effort is huge. In this paper, an automated functional coverage method is proposed to be used along with direct testing in order to automatically track the progress of the test-plan. To the best of our knowledge, it is the first time that functional coverage is proposed to be used with direct testing. Besides, this methodology may predict uncovered corner cases scenarios. The whole universal verification methodology (UVM) hasn't been used because it is a time-consuming process and an infrastructure for direct testing already exists. The time efforts to implement functional coverage only with direct testing are negligible compared to the time efforts needed to implement the whole UVM testbench. An industrial intellectual property (IP) model is used as a case study to demonstrate our findings.**

Keywords*— *Functional Coverage, Verification, Direct testing, Verilog, System Verilog, Regression testing, Perl, UVM.

I. INTRODUCTION

At present, the verification phase carries an important role in the design cycle of a System on Chip (SoC). The verification represents the biggest part of the total manufacturing cost of the devices. The approximated and estimated cost of this phase is about 70% of the total cost of the digital system design. The number of verification engineers is more than the one of the design engineers, so the challenge of verifying a large design is growing exponentially. There is a need to define new methods that makes functional verification easier. On the other hand, unlike software, the errors in hardware design are very expensive as it may require redesigning and physically replacing the failed device [1].

Furthermore, attributable to the increase in the complexity of digital systems lately, the importance of creating efficient designs and reducing their verification time has generated the need to create more efficient design verification techniques. These techniques reduce the time and increase the test coverage percentage.

Different methods have been developed to perform the verification processes; also new software tools have been produced to ease verification process such as VEasy tool [2]. However, these software platforms have been used with different digital systems, and the results obtained were to show that it is necessary to develop more efficient methods that may cover the hardest design cases [3].

The furthest goal of functional verification is to prove the preservation of the design of the device under verification in its implementation and specifications. Thus, functional verification of the chip may be divided into two parts: design verification and hardware or software co-verification methodologies. The following major factors drive the verification challenge.

High design complexity protocols, such as almost industrial applications Memory/USB/PCI, need huge numbers of direct test cases, the presence of both the purchased design IPs and internally developed blocks, high device programmability, and speed Verilog simulator [4].

In directed verification, a separate test-case file created for each feature in the design for verification. Counting the number of test cases is one of the manual processes to know how many features are verified. Verification is done when all tests are coded and passing alone with 100% code coverage. Missing any test scenario in the verification environment is a higher probability because it is a manual process. Functional coverage plays a deep role to discover these types of scenarios.

In constraint random verification, all the features are generated randomly. Verification engineer needs a mechanism to know the information about the verified features of design under test (DUT) [5]. SystemVerilog provides a mechanism to know the untested feature using functional coverage.

Functional coverage is better than code coverage; the code coverage reports what was exercised rather that what was tested.

Code-level coverage measures such as statement/ branch coverage are often used for simplicity. They provide information on excitation of functions, or relative frequency of excitations, which are important in attaining high confidence in the verification results. However, excitation of an erroneous statement doesn't necessarily mean that the incorrect value will manifest to an observation point during simulation [6].

An industrial pseudo code practice for an Intellectual property (IP) model that have two bus modes; one-bit bus width and four-bit bus width as shown in TABLE I. The host side sends commands to the device side to execute part of the device operations. As a sample investigation, maybe the code coverage are 100% for the test cases applied but there are missing hidden test cases in a complex industrial IPs as mentioned in TABLE II.

In this work, a new methodology which uses a functional coverage as main property with an existing directed verification

978-1-4799-8569-2/15/$31.00 ©2015 IEEE

TABLE.I
OPCODE PSEUDO CODE EXAMPLE TO DEMONSTRATE THE DIFFERENCE BETWEEN THE CODE COVERAGE AND THE FUNCTIONAL COVERAGE

Verilog Pseudo-Code Example

If (bus_width = 1bit || bus_width = 4bit)
begin
if (Opcode == CMD1)
{ Execution syntaxes };
 	
else if(Opcode == CMD2)
{ Execution syntaxes } ;
 	
else if(Opcode == CMD3)
{ Execution syntaxes };
End

TABLE.II
COMPARSION BETWEEN FUNCTIONAL COVERAGE AND CODE COVERAGE

Test cases	Code Coverage Result	Functional Coverage Result	Missing Test cases
1- CMD1 and CMD2 sent in 1bit mode. 2-CMD3 sent in 4bit mode.	100%	50%	1- CMD1 and CMD2 not sent in 4bit mode. 2-CMD3 not sent in 1bit mode.

environment infrastructure for eMMC memory protocol is proposed. First version of eMMC is released before the appearance of UVM methodology, so direct verification was the available solution at that time.

The rest of the paper is organized as follows. Section II introduces the proposed methodology to implement functional coverage. Section III analyzes the results that appeared before and after functional coverage based on a sample of coverage report. Finally, Section IV discusses the conclusion of the proposed new approach.

II. THE PROPOSED METHODOLOGY

The design verification environment intent is normally stated in the design specification. A design model is needed to capture the design intent. In theory, there can be many implementations of the same design verification environments. The verification team created predictor and checker models from the design specification to provide self-checking of the design and implementation intent. The verification team started with the development of existing direct verification environment in the company. The intent verification was finished upon the completion of all verification tests and the achievement of functional coverage criteria.

In this case, the coverage criteria include both the functional coverage and Verilog code coverage data. When either functional coverage or code coverage results were found unsatisfactory, more tests are written.

This section summarizes the types of different environments; it starts with a comparison between the traditional direct environment and randomization environment in section II.A. In the following section II.B, the proposed methodology is discussed. The functional coverage proposed methodology is discussed in section II.C. Section II.D discusses the code reusability.

A. Traditional Direct Environment Vs Randomization UVM Environment

In a direct environment, the task of checking all input vectors in the device is infeasible because the coverage space increases when the complexity of the device becomes bigger. Generally, the main objective of the informal verification techniques is to increase design space coverage and changes for finding errors in the digital system design. When the verification of an implemented device is performed in a hardware description language such as Verilog, VHDL, and SystemVerilog [7], the verification engineer needs to use specialized software tools (QuestaSim, ModelSim) [8].

The directed functional verification has an important role to meet the conditions. It has been found that it is hard to cover all corner cases by using the pseudo-random test generation methods, it is necessary to propose new test vector generation methods as shown in Fig 1.

In UVM Randomization environment, UVM represents the latest member of a family of methodologies and their associated base class libraries for using SystemVerilog for functional verification of digital hardware [9].

The coverage collector in the UVM environment entity measures the advance of the verification process by registering the kind of tests and results that occur over a functional coverage model. The scoreboard analyzes the functional correctness of the test outputs comparing them to a reference model. Both coverage collector and scoreboard depend on data provided by a monitor, which is best kept as a separate entity for re-usability [10].

Monitor components are usually specific to particular protocols and sensible to signal level parameters, but independent of the application. In contrast, coverage collectors and scoreboards code are usually highly application-specific and less affected by the interface protocols and timing.

One of the most challenging tasks is to build the first UVM working testbench [11], as depicted in Fig 2. The different components could not be tested individually, so all the environment has to be assembled in order to test the components; making it hard to trace the source of errors. According to the current time market, it is an industrial decision to use the current infrastructure with the functional coverage method.

B. The Proposed Methodology

Fig.4 shows the new proposed verification model hierarchy for direct testbench that consists of three levels: verification environment, code coverage, and functional coverage.

The new proposed environment is based on a layered approach where each layer provides a generic technique. For the following layer, every test consists of a direct Verilog environment test layer, a coverage observation layer and a

Verilog application programming interface (API) layer which represent the upper layer.

The SOC simulation configuration, which specifies how each block in the verification environment is modeled, is described in a Perl hash structure [12]. It instructs the build script how to compile the functional coverage code with the implemented direct test environment.

The main goals of the testbench design are to reduce the test development cycle, facilities the process of debugging, find out the hidden bugs, increase verification code reusability, and increase the level of functional coverage. The testbench implementation is based on a pure direct Verilog environment mixed with SystemVerilog functional coverage mechanism Fig 3.

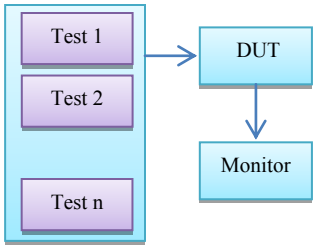

Fig.1 The direct testbench environment.

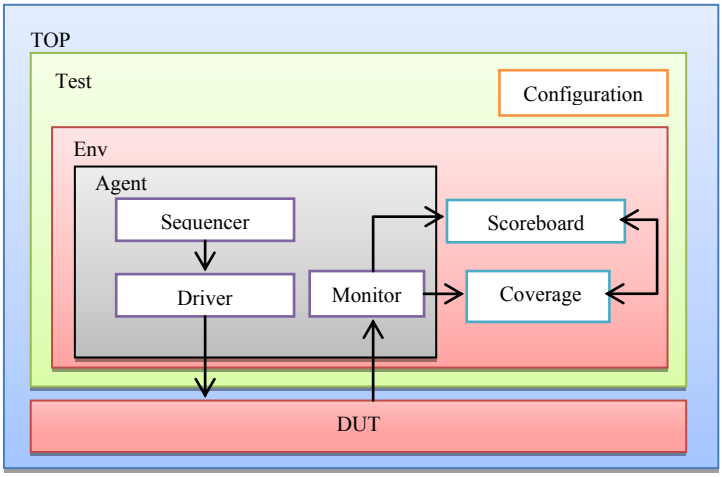

Fig.2 The Complex UVM environment.

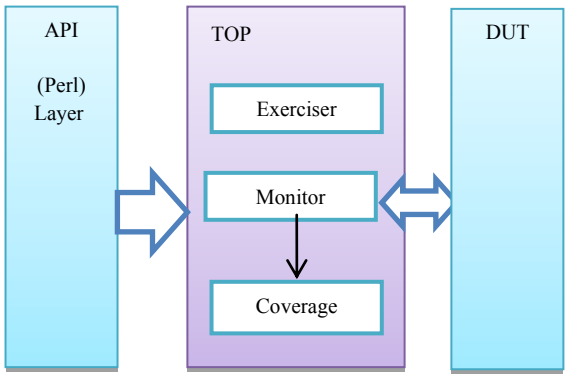

Fig.3 The proposed environment with direct test, API Perl interface and Coverage block.

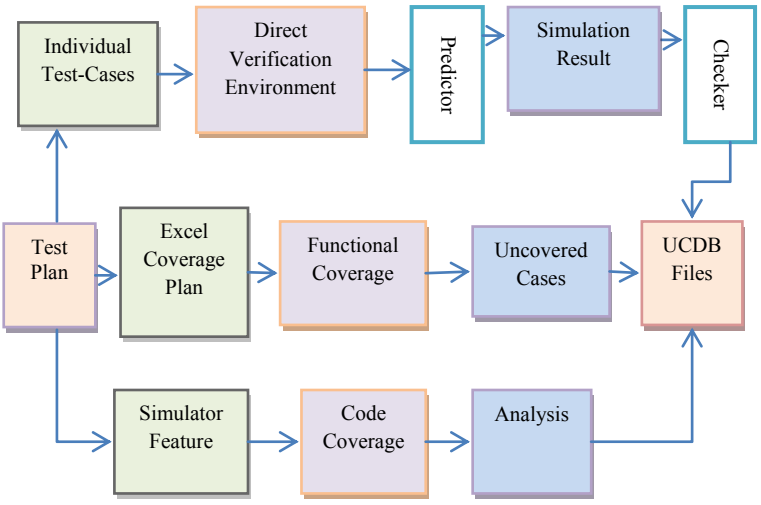

Fig.4 The new proposed verification model hierarchy from the test plan to the generated UCDB file.

C. *Functional coverage implementation*

We should start with the Excel functional coverage plan. Then, start implementing the coverage plan using System Verilog coverage mechanism. The coverage.sv file should be included in appreciate place through the verification environment to discover the uncovered cases from the implemented coverpoints and crosspoints which covers the signals in design or verification environment, the coverage may be showed in covergroup window or by generating a coverage detailed report from the QuestaSim simulation tool. The next step is to open the testcase.ucdb file which is generated from the last step [13].

Finally, The Excel test plan should merged with the coverage result in the tool besides applying the coverage commands to generate reports and graphs [14].

Coverage Excel plan should map a standard format that's mentioned in QuestaSim manual because it would be converted to XML format type which will converted to UCDB format through QuestaSim TCL commands to help the tool understand and merge the coverage plan with the results is shown in Fig 5.

The development of a functional coverage model is done thorough study of high level design specification documents.

A list of design features extracted after that and the main steps of coverage process as summarized in TABLE III.

Each of these features is interpreted in terms of a relation between signals in the executable design description. These relationships are expressed in terms of coverage monitors written in a hardware verification language. These coverage monitors keep track of coverage of design functionality and provide a numerical measure of it.

A complete functional coverage model must consider the details of the important signals in the implementation side because the implementation side may include relationships between signals that are not described in the specification or not implemented explicitly in the verification environment.

As referencing in Fig.5 and according to the coverage format, Section column in test coverage plan represents a number of

each scenario in the coverage plan. The title should be readable and describe the coverpoint explicitly to help the verification team in preparing the analysis. From the coverage report, Wight and Goal values are determined according to the importance of cover or cross point.

An example of the industrial protocol, the goal of address signal for memory interface is 30% to cover certain areas in the addresses only but another side for important signals; goal should be 100% percentage.

Moreover, the UCDB provides the administration of the coverage database for code and functional coverage in order to store collected functional coverage information. The database has to fulfill two requirements. It has to capture already sampled coverage information prior to the next run and to save this data after running all test cases.

D. Code Reusability

Reuse has been an industry buzzword for years now. When it comes to IP, reuse can be extremely powerful way of saving resources and shortening project timescale.

Code reuse in the industrial protocol verification environment is achieved primarily by developing the existing infrastructure direct verification environment by System Verilog functional coverage mechanism based on industrial situations to save the verification time and cost.

In the proposed technique, a set of coverage groups that collectively describe the functional coverage model. Each coverage group specifies a function of some signals in the design. The function is evaluated at each clock cycle during simulation.

Functional Coverage is the fraction of coverage groups whose functions evaluate to true sign at some points during simulation. The coverage groups are defined so that satisfying all coverage functions during simulation indicates that the test cases thoroughly explore the functionality as it related to the given signals.

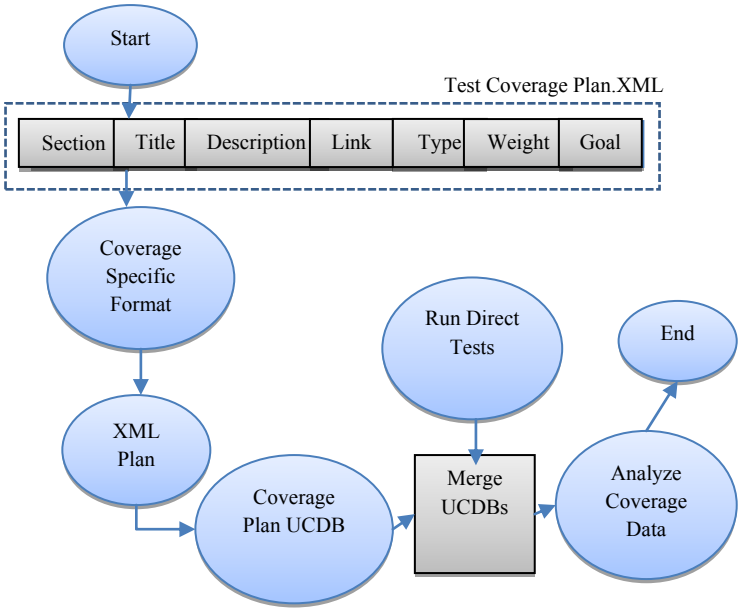

Fig.5 Flowchart for the proposed methodology.

TABLE.III
FUNCTIONAL COVERAGE PROCESS CYCLE

1.	Set the functional coverage plan in excel spreadsheet file.
2.	Start implementing the coverage plan using SystemVerilog coverage mechanism in a separate file.
3.	Include the coverage.sv file in your testbench environment.
4.	Run the Verification environment, check the coverage in covergroup window or generate a report.
5.	Close the tool and open the testcase.ucdb file which is generated from the above step in QuestaSim view mode.
6.	Merge the excel testplan with the coverage result in the tool.
7.	Apply coverage commands and generate coverage reports.

TABLE.IV
REPRESENTING OF DIFFERENT BINS IN THE INDUSTRIAL IP

Bins for ranges	Represent the ranges for addresses.
Default bins	Represent specific values in signals.
Illegal bins	Represent illegal commands at states.
Transition bins	Represent the transitions between states for IP.
Cross bins	Represent the crossing for coverpoints in different modes.
Ignore bins	Represent the ignored values for signals.

Functional coverage with the proposed technique is well correlated to error detection in early verification stages.

The covergroups consist of coverpoints and crosspoints which contains parameters called bins as summarized in TABLE IV.

The set of coverage monitors will be divided into two groups. Good bins these are evaluations which legal and are allowed to occur during simulation. A coverage monitor will be created for each good bin to detect whether or not it occurs during simulation. Bad bins these are valuations which are illegal and must never occur [15]. A coverage monitor will also be created for each bad bin, but if a bad bin is found to occur then simulation is halted because an error has been detected, is shown in Fig 6.

Once the good bins and bad bins have been identified, the SystemVerilog language is used to implement coverage monitors using its coverage group construct. The functional coverage value is the fraction of good bins whose coverage groups have detected at least one occurrence during simulation. The coverage points allow us to measure the degree of exploration of design behavior pertaining to associated signals.

Furthermore, the covergroups supports the definition of bins to collect hits in variable ranges of the coverpoints and bins for each value in the coverpoint, the proposed method discovered that the second type of bins is more effective and found more uncovered values directly.

Example for the original SystemVerilog notation for ignore and illegal bins are displayed in TABLE V. Ignore bins specify values or ranges that shall be ignored while cover the expression, even if they are included in the declaration of other bins. Illegal bins specify values on which the simulation shall be halted.

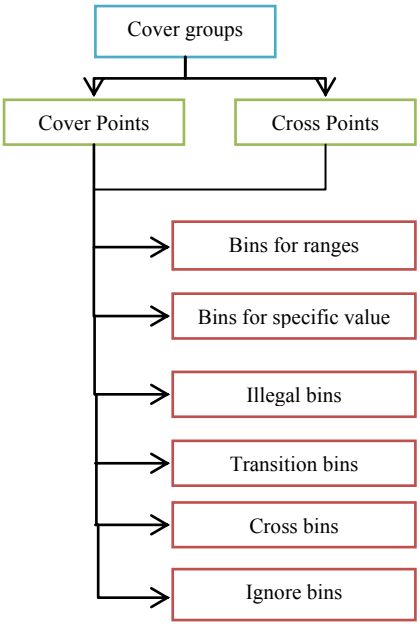

Fig.6 Tree of covergroups with coverpoints, crosspoints, and different bins.

TABLE.V
SAMPLE OF FUNCTIONAL COVERAGE CODE

System Verilog Cover Point Sample

```
covergroup  cg @ (posedge clk);
    C3_BUS_WIDTH: coverpoint   bus_width[2:0]
    {
        bins SDR_1bit = {3b000};
        bins SDR_4bit = {3b001};
        bins DDR_1bit = {3b011};
        bins DDR_4bit = {3b010};
        ignore_bins   reserved = {[3'b100:3'b111]};
    }
endgroup
   cg   cg_new = new;
```

III. RESULTS

This section summarizes the coverage results through the implemented method. It starts with missing verification scenarios in section III.A. In the following section II.B the functional coverage statistics is discussed.

A. Findings: Missing Verification Scenarios

Covergroup window in simulations of the proposed methodology compared with the traditional verification shows that some important findings appeared after merging the functional coverage.

As a result, there are some commands missing and they are discovered by the coverpoints. Additionally, some transitions states are missed which guide the verification team to realize that the functional coverage is a good promising method to expect these hidden states.

TABLE VI shows that the industrial IP consists of different bus modes, commands and different device states. The missed column in the table represents the missing values or scenarios for each signal in the covergroup, covered column represents the signal how many times is covered.

Through crossing feature of bins, few commands are not sent in all operation modes of the industrial IP, Signal mode_buswidths is a smart cross point to cover more scenarios in the IP by crossing between two coverpoints.

B. Functional Coverage Statistics

Regression testing in direct testing environment is modeled as a number of individual test cases which running at the same time through automated scripts. The value of regression testing for finding bugs is often overlooked. Random testing, where input stimuli, test parameters, and test scenarios are generated Pseudo-randomly through the Perl script by randomizing the direct tests, greatly improve the verification quality by generating interesting verification scenarios. Though the majority of regression failures are not real design bugs, close to 10% of those failures can be described as either design limitations needed to be documented or interesting, hard to imagine test scenarios that had to be fixed in the design.

The overall number of chip test cases is around 100 test cases that represent a regression testing, and each test case shows individual scenarios. The regression test-cases are optimized for 50 coverage test case of smart scenarios.

On average, the total number of critical design bugs is 5% percentage of coverage test cases after running the regression and applying the coverage mechanism.

The total number of uncovered scenarios that discovered after applying the SystemVerilog mechanism is 100 scenarios, so around 20 test cases created and added to the environment and the result passed.

TABLE VII discusses the regression mechanism that compare two files related to each test case to print the result for each test case by passing or failing, the first file is a golden file that's produced from a manual process at the first stage of verification process and the second file is just a file which is produced from a regression testing process to check the testcases at different scenarios.

TABLE.VI
SAMPLE OF FUNCTIONAL COVERAGE REPORT

Signal Variable	***Covered***	***Missed***	***Type***	***Goal***	***Weight***
mem_addr	307	717	cover point	30	1
cmds	42	2	cover point	100	1
mode_sdr	1	0	cover point	100	1
mode_ddr	2	0	cover point	100	1
bus_widths	3	0	cover point	100	1
device_states	40	5	cover point	100	1
cmds_ddr_4bit	200	30	cross point	100	1
modes_buswidths	4	2	cross point	100	1

Script mechanism is prepared to adopt a randomization through the proposed environment.

TABLE.VII
GENERIC PERL SCRIPT FOR REGRESSION

Generic Perl Script For The Proposed Direct Testing Environment

```
@scenario = ("scenario1"," scenario2", ……);
   #Each scenario have different idea
@testlist = ("test1.v","test2.v", …….);

for ( $j=0;   $j<scalar(@scenario); $j++ )
{
     for($i=0; $i<scalar(@testlist); $i++ )
     {
          system("make –f  Makefile  TESTCASE=$testlist[$i]
                                     LOGFILE=$test[$i].log ");
          if (system(cmp    –s    test.log     test.golden) == 0)
            { print "Pass"}
          else
            {print "Fail"}
     }
}
```

TABLE.VIII
FUNCTIONAL COVERAGE STATISTICS RESULTS

Direct Testing	**Coverage statistics**		
	Before functional coverage	***After adding functional Coverage***	***After Rewriting the missing testcases***
Percentage of Coverage	unknown	70%	100%
Covered Scenarios	unknown	2026	2536
Uncovered Scenarios	unknown	510	zero
Critical Missing	unknown	5%	zero
Expected uncovered Scenarios	10 %	253	Zero

TABLE.IX
COMPARISON BETWEEN DIFFERENT VERIFICATION ENVIROMENTS AND OUR PORPOSED METHODOLOGY

	Direct Verilog	UVM	The Proposed Environment
Number of tests	Large	Small	Large
Time effort to build the environment	Small	Large	Small
Randomization	No	Yes	Yes
Reusability of stimulus	No	Yes	No
Scalability of testbench	No	Yes	No
Functional Coverage reusability	No	Yes	Yes

The effort of this work takes two weeks from the whole project period, Compared to the UVM environment that takes months to build the components and layers.
TABLE VIII shows the distribution of coverage before and after adding the coverage mechanism.

The comparison among direct environment, randomization environment and proposed environment is shown in TABLE IX.

IV. CONCLUSION

In this paper, an automated functional coverage method is proposed to be used along with direct testing in order to automatically track the progress of the testplan. Compare to using the whole UVM testbench, the proposed methodology saves time efforts especially in our case where the direct testing infrastructure already exists. An industrial IP model is used as a case study to show the efficiency of our methodology which not only automatically tracks the testplan progress, but also predicts uncovered corner cases scenarios.

REFERENCES

[1] A. Martinez, R. Barron, and H. Molina, "Automated functional coverage directed for complex systems,".VLSI-SOC ,2014.

[2] Samuel Nascimento Pagliarini, Paulo Andr´e Haacke and Fernanda Lima Kastensmidt,"Evaluating Coverage Collection Using the VEasy Functional Verification Tool Suite,". LATW, 2011.

[3] M. keaveney, A. McMahon, N. Okeeffe, "The development of advanced verification enviroments using system verilog,". ISSC, 2008.

[4] G. Defo, C. kuznik, W. Muller, "Verification of a CAN bus model in SystemC with functional coverage,". SIES, 2010.

[5] G. Zhong, J. Zhou, B. Xia, "Parameter and UVM, Making a Layered Testbench Powerful,". ASICON, 2013.

[6] Khalil Arshak, Essa Jafer, Christian Ibala , "Power Testing of an FPGA based System Using Modelsim Code Coverage capability,".DDECS, 2007.

[7] Jonathan Bromley, "If SystemVerilog Is So Good, Why Do We Need the UVM?,". FDL, 2013.

[8] MentorGraphics, Questa® SIM Reference Manual, 2011.

[9] Hung-Yi Yang, "Highly Automated and Efficient Simulation Environment with UVM,".VLSI-DAT, 2014.

[10] Young-Jin Oh , Gi-Yong Song ,"Simple Hardware Verification Platform using SystemVerilog,". TENCON, 2011.

[11] Yingke Gao, Diancheng Wu1, Quanquan Li1, Tiejun Zhang1, Chaohuan Hou1 "Design and Implementation of Transaction Level Processor based on UVM,". ASICON, 2013.

[12] Daniel L. Moise ,Kenny Wong,"Extracting Facts from Perl Code,". WCRE, 2006.

[13] MentorGraphics, Questa® SIM Verification Management User's Manual, 2011.

[14] MentorGraphics, Questa® SIM User's Manual, 2011.

[15] Verma, S, Harris, I.G. , Ramineni, K "Automatic Generation of Functional Coverage Models from CTL,".HLVDT,2007.