SOC Verification:Approach & Strategy

Devansh SharadKumar Mehta [1] Mrs. Falguni Sharma [2] Mr Bhavesh Soni [3]
[1] Post Graduate Student of Electronics and Communication [2] Senior Tech Lead [3] Assistant Professor
[1,3]U.V Patel College Of Engineering, Ganpat University Mehsana Gujarat [2] e-Infochips Pvt Lmt, Ahmedabad Gujarat.

***Abstract*—** A VLSI system plays an important role among all other system and Verification plays important and a huge role in a VLSI life cycle. SOC is now a days very popular due to its reusability. So this paper gives a guidance related to SOC Verification and practical approach for SOC Verification which includes fusion of verification Environment with a mix C tests for debugging embedded processor and Verilog test bench for monitors and checkers. This paper gives an idea of how a SOC is going to verify covering all its functionality using verification methodology's parameter such as functional coverage and code coverage. Using Verification Techniques and verification approach we can verify system level SOC.

Keywords:- Verification,System On Chip,UVM Methodology

I. INTRODUCTION

Semiconductor technology has progressed to the point where it is now impossible to implement system level verification on a single LSI chip. However traditional LSI verification becomes less and less powerful as the scale and complexity increases. In Fact more than half of time required to develop a system on chip is used for functional verification. A new verification methodology for SoC's should therefore be established. Today's highly complex System-on-a-Chip (SoC) designs demand a comprehensive and integrated design, verification and application development environment [4].

This paper starts by giving introduction to SOC and verification. In this paper we will look after practical approaches and strategy followed for SOC verification. We will look after the importance of verification along with its role in the ASIC flow and the importance of SOC in industrial level market. Here in this paper our prime aim is to showcase the industrial approaches carried out for an SOC to be verified and the flow for the verification. This paper will showcase the practical approach and criteria required to be matchup. Most importantly it will also give guidelines for the SOC verification. This paper also describes the importance and things included in preparing verification plan.

This paper discusses about the combined approach of simulation based verification and coverage driven verification. Simulation based verification environment means verification done by running well targeted simulation in self-checking verification environment and coverage driven verification environment means verification done with randomization and considering all corner scenarios.

As to reach to any destiny which way to select is most important similarly it is our belief that creating good verification environment and choosing verification approaches is very important as that will give direction to your verification goal. Verification environment involves some degree of programing and programing can be in any language. Most popular programing language specially designed for verification is system Verilog but a verification engineer can use any of Verilog, system Verilog, system C, VHDL, C/C++/PERL languages as per the tool on which he is going to work[1].

II. OVERVIEW OF SOC

A. What Is Soc

A system on a chip or system on chip (SoC or SoC) is an integrated circuit (IC) that integrates all components of a computer or other electronic system into a single chip. It may contain digital, analogy, mixed-signal, and often radio-frequency functions—all on a single chip substrate. Whereas the term SoC is typically used for more powerful processors, capable of running software such as the desktop versions of Windows and Linux, which need external memory chips (flash, RAM) to be useful, and which are used with various external peripherals.

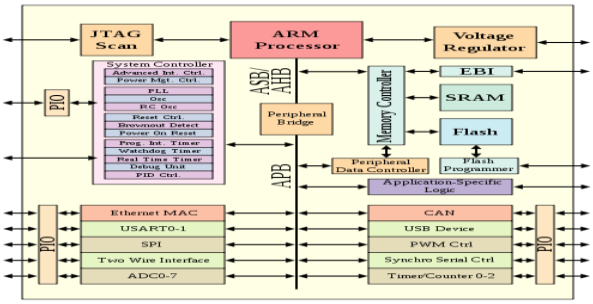

Fig. 1: SOC Design [2]

In short, for larger systems, the term system on a chip is a hyperbole, indicating technical direction more than reality: increasing chip integration to reduce manufacturing costs and to enable smaller systems. [2]

B. Pros & Cons of Soc

The benefits of SoC are self-evident: Everything needed to run the computer is contained in that one chip - the smaller the better. The chip has all that is needed to run even detailed computer functions [6].

The main obstacle to a final version of SoC continues to be the laws of physics. When you start mixing hardware and software, the demands on the chip and its silicon can be tremendous, sometimes conflicting or impossible with current technology. In the end, SoC might not be so far around the corner.1) higher performance, since all the circuits will be on a single chip; 2) smaller space requirements; 3) lower memory requirements; 4) higher system reliability; and 5) lower consumer costs [11] . The challenges posed by SoC technology include: 1) larger design space; 2) higher design and prototyping costs; 3) longer design and

All rights reserved by www.ijsrd.com

prototyping cycle time; 4) more complex debugging; 5) lower IC yields and higher wafer fab costs due to the relatively larger die sizes involved; 6) integration of intellectual property from multiple (and possibly independent) sources. [10]

III. VERIFICATION

A. What Is Verification

In general, Verification is one technique to verify something. In ASIC division Verification means to verify the design all possible scenario and to verify the purpose of the chip integrated .Through verification technique we can verify all the functionality to check that chip is working proper in all possible situation and there is no issue in our design [7].

In one sentence we can define that verification is a process used to demonstrate the functional correctness of a design. [3]

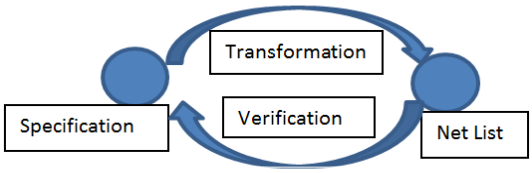

Fig. 2: Verification Transform [7]

B. Where Verification Plays Role?

In Chip Design flow verification plays an important role. Over 70% of time is consumed by verification only. According to the requirement specification of any chip are prepared and according to that specification designer develop design code for it and to ensure that design is prepared well according to specification is the role of verification engineer. Verification engineer exercise this design in all possible ways and check whether the specification is transformed well into a design which works well in all erroneous and possible scenarios.

Verification consists of:-

- Identifying the complete set of conditions to be checked in the design
- Generating the stimuli test cases to be applied on the design to check for these conditions
- Ascertaining that stimuli when applied to the design, results in correct behaviour.

IV. SOC VERIFICATION

In many ways verification of SOC is similar to verification of any ASIC: you need to stimulate it, check that it adheres to the specification and exercise it through a wide set of scenarios. However SOC verification is special and it present some special challenges [1].

All the IPs,Cores and other components put together to build a SoC are verified at standalone level So,the big question are:

Why then we need SoC verification?

SoC Verification is about verifying the complex SoC design together. A typical SoC verification flow consists of three major tasks: modify, test and evaluate. The diagram below depicts the flow and the various processes carried out during SoC verification. Designers follow this iterative loop of modification, testing and evaluation until verification objectives are met. The size of the iterative loop circumference indicates time, which is the key in managing the throughput. The aim of SoC Verification is to focus on three primary area of SoC verification to ensure device success. These areas are

- Connectivity:(Is everything is connected and connected correctly.
- Functionality:(Does each IP block, subsystem and system function as expected.)
- Performance**(**Does the system perform as per required.**)**

A Verification environment with a mix of C tests for debugging (embedded Processor) and Verilog test bench is for monitors and checkers is used to verify an SoC. Particular environment is made to newer technique, including system-level design approach, code coverage, directed random testing ,functional coverage etc. Using verification environment developed an SOC verifies connection between the blocks, its reusability, its functions and performance. Along with this some more things to keep in mind are mentioned below:-

A. Reuse Of IP Blocks

The most advantage of an SOC verification is its reusability purpose. The reuse of many hardware IP blocks in an mixed and match style reduces Verification time and efforts. Many companies treat their verification IP as valuable asset and sometimes valued more than hardware IP[13].

B. Integration

The primary focus in SOC is checking the integration between various components. The underlying assumption is that each component was already checked by itself. This special focus implies a need of special techniques[13].

C. Complexity

The combined complexity of the multiple sub-systems can be huge and there are many seemingly independent activities that need to be closely correlated. As a result we need a way to define complicated test scenarios as well as measure how well we exercise such scenarios and corner case [9].

D. Unique Bugs

Here are some typical unique bugs which may exists in an SOC which one should take care[8]

- Interactions between blocks that we assumed verified
- Unexpected access conflicts between shared resources.
- Cache Coherency in multi-core system.
- Interrupt connectivity and priority scheme.
- Arbitration problem and deadlocks
- Priority conflicts in exception handling.
- Multiple power domain region, clock domain crossing
- Unexpected HW/SW sequencing

All the challenges above indicated the need for rigorous verification of each of the SOC components separately and for very solid methodology and tool for the verification of full system [1]. This requirement for executive verification indicate the need for a high level of automation, otherwise the task of verification will simply become impractical [1].

All rights reserved by www.ijsrd.com

1) Comparison Of SOC Level Verification And Block Level Verification[7]

Verification is of two levels one is SOC level verification and another is Block level verification. In Block Level verification each block is verified at block level with its each supporting features of block or IP. In SOC level verification each block are verified at system level only means each blocks features which are required to be verified.

Block level verification are done with RTL code only but in SOC level verification we have an advantage of verification done with behavioral model of that RTL Design or can have verification done with any other VIP. Block level verification is done prior of integrating that block with the system and then SOC level verification is done to verify connectivity and performance of each block and overall system. SOC level verification is a top level approach of verifying system while Block level verification is a bottom level of verification. Sometime SOC level verification becomes more complex to integrate IP with the system and creating a top level test bench implementing verification methodology.

2) Key Concern For Verifying SOC[10]

While verifying our design the main question that arises is "*What to verify at SoC level*?" For this, one must understand the basic difference between SoC verification and intellectual property (IP) verification. While designing a SoC, IP is generally delivered by a third party or designer we can assume it has been verified by the IP level verification test-suite. The SoC team integrates the IP as per the defined functionality in the specifications. Then, the SoC verification team has to verify the chip level functionality, which mainly focuses on the integration of IP.

Some important concerns during SoC level verification include:

- *Pin Muxing In The Chip*: The number of pins in a SoC is directly related to the chip cost and determined by customer requirements, so the SoC is restricted with limited pins as compared to the number of pins required by IP to interface to the external world. In order to use the SoC pins efficiently, they are interfaced to the internal IP via complex muxing logic at the SoC level. It is very important to verify this logic.
- *Protocol Checks*: Protocol checkers should be added at module boundaries.
- *Initial Sequences for The Soc Bring Up*: This mainly includes the reset and boot-up sequences. Apart from reset sequence checking, a typical SoC can have multiple ways to boot-up, Interrupts, Times testing e.g. from internal or external memory. Checking the boot from all possible sources can be one area of focus.
- *Memory And Register Access:* Focus here can be to ensure that design can configure registers and memory to enable various SoC level scenarios.
- *Control And Data Paths In The Design*: The focus here is to cover data paths that involve configuring transactions with the external world (interface protocols at SoC like LIN, CAN, UART, etc.), configure masters of transactions to work in parallel (processor, DMA etc), and excitation of functional paths for interrupts.
- *Low Power Feature*: Low power is an important feature in the SoC because it is a key requirement for applications like energy metering, smart phones or tablets. Tests should check low power entry and exit.
- *Polarity Check*: Polarity of the connection of the output port of one module to the input port of the other module.
- *Gate Level Simulation*: The point here is to catch timing related issues, constraint validation glitches, and combinational logic on reset path, etc.
- *System Level Use Cases*: Special consideration needs to be given to the application-level use cases that excite the control and data paths for multiple applications running in parallel. These are the application scenarios that the customer will finally run on the SoCs and adds confidence in verification. This can be targeted to find any bottleneck in the design, such as processor latency, DMA path latency, and/or interrupt priorities in customer use cases.

3) Efficient Verification Strategy

As per the VLSI industry trend, the complexity of the chips is increasing while cycle time is reducing. So, even after so much planning on "what to verify," it may still not be possible to verify upcoming complex designs. As a result, designers need to plan for automation needs in the verification cycle to help verify the entire design while maintaining quality standards and on time delivery[9]. The following steps can help to make the verification process more efficient:

4) Effectiveness of the Test-Suite

The verification plan should be made from the system level architecture document (Chip Spec) so that each feature mentioned in the Chip Spec is mapped to at least one test-pattern in the verification suite. The test patterns must be reviewed and checked for the feature (verified by that pattern) and map it to the corresponding features in the Chip Spec. The test-patterns must contain proper comments, as per the pattern intent.

1. *Automatic Test-Pattern Generation*: Tests that check basic features (like the memory and the register access) can be generated through automation.
2. *Self-Checking Patterns*: Test patterns must check the intended functionalities and should indicate pass or fail, depending on the checks intended. Some of the checks could be to verify the number of interrupts occurred vs expected or expected values of status registers at end of a transaction. Having automated checks can save a lot of debugging time on failures.
3. *Automated Checks in Test bench*: Monitors and assertions should be enabled in the test bench to perform automated checks for key functionalities. Clock monitors and low power mode entry sequences are a few of the checks that can be helpful.
4. *Reusable Test-Suite*: A verification test-suite should be made in such a way that it is fully reusable. That way, for every new SoC it is simply a plug and play with some minor adjustments in case there are some

All rights reserved by www.ijsrd.com

new features of that IP that is to be used in the SoC. Test cases should be coded in a high level language so that they are independent of the processor and easily reusable for different SoCs. This avoids the extra effort of test-suite creation every time an SoC changes and reduces the verification cycle time.

5. *Randomization*: Use randomization in the test bench for hitting the corner cases in the design. Some of the parameters that can be randomized are:

- Values written in the register's read/write patterns
- Number of resets to be given in the patterns
- Pin used for wakeup from Low-power mode
- Latency in the wakeup from low-power mode
- Wakeup pulse duration

The following help hit the corner cases that may be difficult to target in a directed scenario:

– *Regressions*: SoC level regressions must be enabled at very early stages in the verification cycle to review progress against overall test completion.
– *IP Level Regressions*: An IP level regression suite must be run on IP releases delivered to the SoC team, so that the SoC team can focus on SoC level defects.
– *Formal Verification of Iomuxing*: Efficiently verify the IO path. As there are limited pins on chip and with increasing logic in chip, the number of functionalities muxed on each pin is increasing. So, covering all of the paths by directed patterns might not be possible. Formal checks can be done for IOMuxing verification as a quick and efficient method.
– *Coverage*: Design coverage (Toggle, Code, FSM) must be enabled to collect the coverage data in each regression run so as to see the coverage holes and cover them. This ensures that you are progressing towards the end goal. Add the cover points and cover groups for those cover points in the test bench and collect the functional coverage to check whether all the assertions and monitors are covered as per the test-suite.

5) Code And Functional Coverage[1]

Coverage is an important tool for identifying areas that were never exercised. But there are two other important benefits it can give:

– Identify areas that were sufficiently exercised, and therefore need not be exercised any further
– Replace the need to write a lot of deterministic, delicately

The reason these two are much less recognized is that most people think of code or toggle coverage when they speak of coverage, while the biggest value can be found, in fact, in functional coverage. Both code coverage and toggle coverage are good "first level indications" for areas that were never accessed. However, they can never tell you if you have exercised "enough". Exercising enough means that your entire test plan was executed, and all interesting scenarios were exhausted. Code and toggle coverage do not give you any indication of "functionality" covered. For example, code coverage can not tell you if all types of cells were not received on all ports of an ATM switch, nor can it tell you which sequences of opcode were executed by a CPU. You can achieve 100% code coverage, and still miss key areas where bugs can be hiding. Functional coverage, on the other hand, allows you to define exactly what functionality of the device should be monitored and reported. This means you can make your functional test plan executable. You can get reports that will measure exactly what you care about, and describe that Information in your own terms.

Looking at functional coverage reports, you may conclude that certain features or scenarios of interest in a certain area were already exercised to the full extent. Seeing that, you could stop running tests in that area, and focus your efforts on the areas that were neglected. A functional coverage tool can also be used as a query mechanism, to investigate further into what was or was not exercised. You should be able to interactively explore more combinations of events. For example, if you have coverage metric on a state machine, and a metric on the opcode a CPU has processed, you may be able to combine the two and see what opcodes were processed while the state machine was in each possible state.

But most significant impact of functional coverage in context of SOC verification is in eliminating the need to write many of the most time consuming and hard to write tests. Using functional coverage, you can describe complicated scenarios of interest as coverage metric. Instead of spending days on crafting a few directed tests to hit these corner cases, you can write one generic test aimed at the whereabouts of those corner cases. Such a test can run with some variance factors multiple times, for a full day, and flood the area of interest. The test might hit the corner case only occasionally, but at the end of the day, you may open the coverage report and see that the corner case was encountered several dozen of times, in a variety of scenarios. You may have achieved, in less than an hour, more than you would have achieved in days of writing directed tests. Using the coverage report you can also choose the best test instances and add only those to the regression suite you run periodically.

V. SOC VERIFICATION FLOW

Companies and design groups around the world have many different approaches to verification and especially to SOC verification. Thus it is quite impossible to say "this is how things are typically done". However, there are several characteristic that can be seen quite often. Let's take a look at some trends in traditional SOC verification or have a look in the SOC verification flow

A. Verification Strategy

There is a traditional flow which verification teams flows for success of system to be verified. Everything for a success starts with proper planning. So same with verification proper planning is done. For going for verification plan team discuses internally the scope, possibilities, assumptions, approach and all possible thing which helps them later. Team analyse the top level design and specification and try to map that with their verification models and prepare a rough sketch for verification environment.

Verification plan and test plan are prepared considering area of focus and boundaries for verification

All rights reserved by www.ijsrd.com

along with complexities involved in the verification. Then strategy and verification environment design is prepared along with this connectivity matrix is prepared which helps each blocks visibility and way of accessing them. Connectivity matrix is a matrix prepared which show's each blocks connectivity with the other blocks. Reusable Test bench are prepared along with the integration of model required for verification. According to the test plan verification is done to check the functionality and connectivity with model or RTL for defect and tracking of bug. If any bug is found verification engineer try to fix that issue otherwise inform to designer to fix it[5].

Fig 3 Verification Flow

B. Verification Plan

Verification plan is a part of design report. Verification plan is an important aspect to start with the Functional verification. Before going to develop test cases and start verifying functionality we need to develop verification plan. In verification we decide the approach require to verify SoC, complexities involved, challenges which we might face, blocks required to be verified with considering different and corner scenarios, level of abstractions, assertions required.

Verification plan includes:-

- Test Strategy for both and top level module
- Testbench Components-BFM,bus monitors.
- Required Verification Tools
- Simulation environment including block diagram
- Key features needed to be verified in both levels
- Regression test environment and procedure
- Criteria have to be defined for Verification Closure.

Verification Plans helps in developing test bench environment early. It also helps in developing test scenarios and corner test. It helps in developing verification environment parallel with design task by separate team.

C. Test Plan

Many companies apply the same techniques they used in ASIC verification to SOC verification. These typically involve writing a detailed test plan with several hundred directed tests and describing all sorts of activates and scenarios the designer and architects deem important[1]. While these testplan are important and useful plan are prepared considering two important things, one is considering connectivity and configuration verification and functional verification.

D. Debugging Techniques

The main objective of debugging is to ensure that behavior of the design is according to the specification. Following are some of the techniques which are used for debugging test cases:

– *Manual Checking*

Manual checking done on the bases of the waveform. Here in manual checking we are generating stimulus input to our design and check the behavior of our design with the actual protocol and specifications provided. Here we are generating stimulus with the help of testcase created using virtual sequences which help for configuration as well as for targeting to specific scenario. With the help of waveforms and state of operation for specific scenario which we have targeted, we come to know if there is an unexpected behavior of our design which helps us to find out bugs. Waveform analysis will help us in debugging our test cases or targeted functionality as shown in below figure.

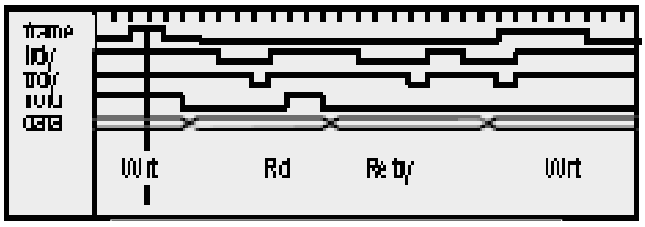

Fig 4: Simulation Waveform

– *Golden Model*

In Golden Model we are comparing out testing model with golden model and that comparison results us that our verifying model are working well or not as shown in below figure.

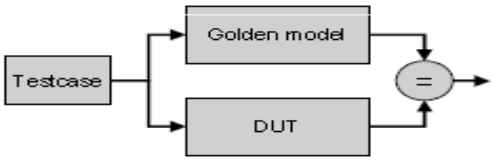

Fig. 5: Golden Model Approach

While discussing about the golden model the first question comes in our mind is what golden model is. Golden model is a model which has all the capabilities of model. Golden model means VIP (verification Intellectual Property). One can verify their design with the help of VIP also. Here the approach of verification is slightly modified. Here test bench is developed in such a way that our design outcomes are matched or compared with the golden model i.e. VIP and then scoreboard will result the comparison of both the outcomes. Here we target the scenario with random stimulus input to both and then check the result.

– *Assertions*

By keeping assertions we can track or monitor signals and it behavior while running test cases. Assertions will automatic result in error if our signal is not working as per our expectation and we can easily justify where an error is.

E. Verification Testbench Approach

Functional verification tests the functionality of the DUT using the testbench. The testbench is created based on the specifications of the design. There are mainly three functional verification approaches: black-box, white-box, and gray-box. The functional verification is performed using event-based and cycle-based simulators.

1) Black-Box Verification Approach [7]

In this approach, the design is treated as a black box, and the internal design details are unknown for verification. The test bench is created based on the block specification. The errors in the design can be detected only at the output, since the approach does not provide insight into

All rights reserved by www.ijsrd.com

the design details. To stimulate the errors, exhaustive test vectors need to be authored. The black-box approach focuses on the functional requirements of the design. It attempts to find the following types of errors:

- Initialization and termination errors
- Interface errors
- Performance errors
- Incorrect or missing functions

The black-box approach provides poor observability and controllability, making the debugging task very difficult. It can ensure that the design functions as expected for the given input stimuli, but it does not ensure that the input stimuli fully exercise the design code.

2) White-Box Verification Approach [7]

This approach provides good observability and controllability for verification. It is also called as structural verification. As shown in Figure, the design data and structure are visible for verification. The stimulus for corner cases can be easily generated, enabling the source of errors to be detected and identified. This approach is widely used for verification in design houses.

3) Gray-Box Verification Approach [7]

In this approach, some of the details of the DUT are known, but not all of the relevant ones to the function that is being verified. This may be because of contractual restrictions, or because the user does not want to verify at greater level of detail. This approach is a mix between white-box and black-box verification.

VI. Conclusion

SOC verification might seem very similar to ASIC verification at first glance, but it is actually special in many aspects. The main focus of SOC verification needs to be on the integration of the many blocks it is composed of. As the use of IP is prevalent in SOC designs, there is need for well defined ways for the IP developer to communicate the integration rules in an executable way, and to help the integrator verify that the IP was incorporated correctly. The complexity introduced by the many hardware blocks and by the software running on the processor points out the need to change some of the traditional verification schemes, and trade them in for more automated verification approaches. Similar conclusions can be seen in [1, 12, 13].

References

[1] "An Integrated Methodology for SoC Design, Verification, and Application Development". SOC verification. N.p., n.d. Web. 20 Apr. 2014. <http://www.amir.com/docs/gspx04_1190.pdf>.

[2] "ERR: ETD etd-0830101-210728." ERR: ETD etd-0830101-210728. N.p., n.d. Web. 20 Apr. 2014. <http://etd.lib.nsysu.edu.tw/ETD-db/ETD-search/getfile?URN=etd-0830101-210728&filename=etd-0830101-210728.pdf>.

[3] "Functional Verification Automation for IP, Bridging the Gap Between IP Developers and IP Integrators." Veristy. N.p., n.d. Web. 1 May 2014. <http://www.verisity.com/html/technical_papers.html>.

[4] "How to verify SoCs." EDN. N.p., n.d. Web. 29 Apr. 2014. <http://www.edn.com/design/test-and-measurement/4394687/How-to-verify-SoCs>.

[5] "How to verify SoCs 2." EDN. N.p., n.d. Web. 15 May 2014. <http://www.edn.com/design/test-and-measurement/4394687/2/How-to-verify-SoCs>.

[6] "Practical Approaches Of SoC Verification." Center for Embedded Computer system. N.p., n.d. Web. 4 Apr. 2014. <http://cecs.uci.edu/ics259/05-08-03/verisitySOCverify.pdf>.

[7] Rashinkar, Prakash, and Peter Paterson. "chapter 2 SoC level Verification,Chapter 3 Block Level Verification." System-on-a-Chip Verification Methodology and Techniques. Boston, MA: Springer US, 2002. 66-70,88-89,117-120. Print.

[8] "SOC description." wisegeek. N.p., n.d. Web. 25 Apr. 2014. <http://www.wisegeek.com/what-is-a-system-on-a-chip-SoC.htm>.

[9] "Spec-Based Verification." verisity. N.p., n.d. Web. 2 May 2014. <http://www.verisity.com/html/technical_papers.html>.

[10] "System-on-a-chip." Wikipedia. Wikimedia Foundation, 5 Apr. 2014. Web. 15 Apr. 2014. <http://en.wikipedia.org/wiki/System-on-a-chip>.

[11] "The Great Debate: SOC vs. SIP | EE Times." EETimes. N.p., n.d. Web. 30 Apr. 2014. <http://www.eetimes.com/document.asp?doc_id=1153043>.

[12] "The forgotten SoC verification team | EE Times." *EETimes*. N.p., n.d. Web. 26 Apr. 2014. <http://www.eetimes.com/document.asp?doc_id=1279817>.

[13] D. Geist, G. Biran, T. Arons, M. Slavkin, Y. Nustov, M.Farkas, K. Holtz, A. Long, D. King, S. Barret, "A Methodology For the Verification of a 'System on Chip'," DAC, 1999.

[14] C. Hanoch, "High Level Verification Automation: A New Methodology For Functional Verification of Systems/ ASICs",DesignCon,

All rights reserved by *www.ijsrd.com*