Global Scientific JOURNALS

GSJ: Volume 5, Issue 11, November 2017, Online: ISSN 2320-9186
www.globalscientificjournal.com

SELECTIVE KDE AND TEMPORAL ANALYSIS FOR VERIFICATION OF DIGITAL CHIPS

Bhushan S Vasisht,Mr Narayan T Deshpande

B.E, Dept. Of E&C, BMSCE, Bangalore

Abstract:

The demand for new and powerful electronic devices is increasing every day. These powerful devices require complicated designs. The design engineers do their best to provide with good designs but still many bugs may remain hidden in the design. These bugs may cause huge loss if they are found after the device is sent to the market. Therefore it is necessary for the designs to be verified thoroughly.

The verification process is also a complicated process.Today around 70% of the product cycle is used up for verification. Thus a method to hasten the verification while satisfying coverage is required. Due to complex designs having complex behavioral pattern, the verification engineer needs to write many test scenarios which is practically not possible. Thus they randomize the process at some level.

When a process is randomized, generally it still follows a set pattern as decided by the seed of the pseudo-random algorithm. Even for a completely random pattern, it is difficult to hit wide range of test-values. It will repeat some of the values before all values are hit.. This will result in both lesser coverage and increased time for verification.

Machine Learning techniques like Kernel Density Estimation(KDE) and Temporal Analysis are used to reduce the amount of time needed for verification.Temporal Analysis include Assertions,Properties and Sequences which are inbuilt in SystemVerilog. Kernel density estimation is used to divide the range of bit fields into many kernels and verify the resulting kernels in the descending order of the number of values in each kernels.Left over kernels can be verified by directed testing.Thus the amount of time for verification is reduced.

1.0 Introduction

In recent years, most design starts have grown so large that it is not feasible to use functional vectors for manufacturing tests, even if they provide high fault coverage, because it takes many vectors to test the functional corners of the design that the cost of time is huge. Hence verification has become a major concern in the industry.

Verification has evolved into a complex project that often spans internal and external teams, but the discontinuity associated with multiple, incompatible methodologies among those teams has limited productivity.

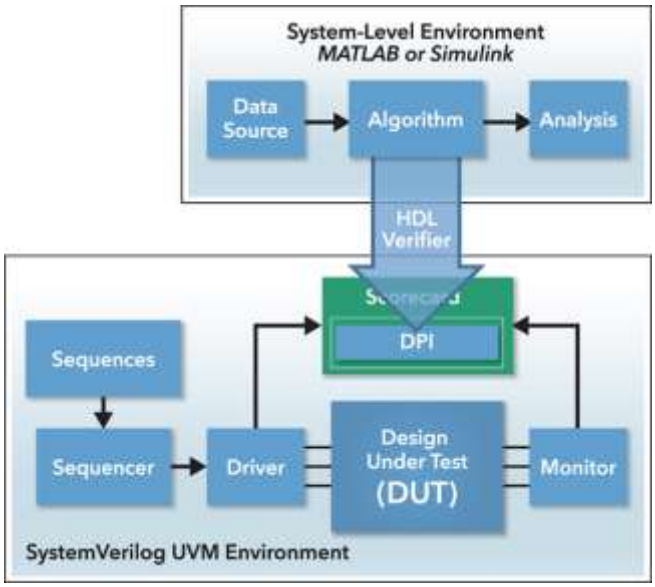
The Universal Verification Methodology (UVM) addresses verification complexity and inter-operability within companies and throughout the electronics industry for both novice and advanced teams while also providing consistency.Machine learning is the term used for processes that provide computers with the ability to learn without being explicitly programmed. Machine learning focuses on the development of computer programs that can change when exposed to new data.

1.1 Problem definition

Almost 70% of the design cycle of chips goes in verification.Verification is the most important step in the design cycle and takes huge amounts of manpower and time to be completed.Verification also involves complex processes both on the DUT side as well as utilization of software for coverage monitoring.This increases the time to market by a huge margin.

Verification involves the verification engineer to closely monitor the interactions on the DUT side and manipulate the signals being driven from the test environment to ensure all corners cases are driven to the chip and corresponding proper outputs are seen.This is a very tedious process and something which is unavoidable because the cost of each bug propagated to the customer is a huge loss to the manufacturer and the cost for repairing the bug and fabricate the chip again is also a non productive use of manpower and time.

1.2 Problem Solution/Block Diagram



ML-UVM System Architecture

1.3 Random Number Generator

Nothing in this world is purely random.Some extent of determinism can be seen in all random processes.As for coming to random number generators, they are algorithms that take a pre-loaded large number and multiply it with the current value of the random variable and then take the modulus of a similarly pre-loaded large number called the seed value.The randomness of the algorithm is constrained by the seed value. For verification,we use constrained random approach,i.e the range of values which are to generated are constrained,and the seed value is calculated by the algorithm for that range.

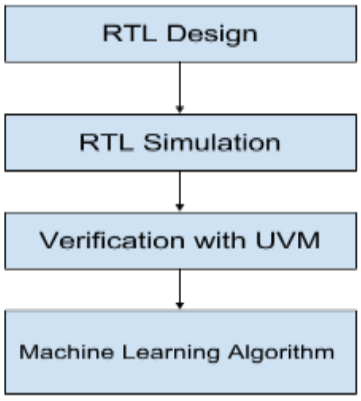
$$X_{n+1} = [a * X_n] \pmod m$$

where: $a = 16{,}807$, and $m = 2^{31} - 1$

Pseudo Random Number Generation Equation

1.4 Verification Flow

- RTL is designed with HDL languages like Verilog or VHDL or SystemVerilog.
- The simulation of the RTL can be done with simple driver testbench.
- Verification is done with UVM environment for reusable sequence generation.
- Machine Learning Algorithms are used to get optimized sequence generation for faster verification.



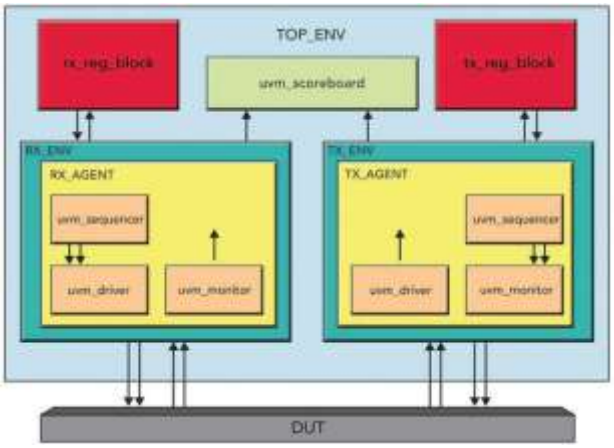
Verfication Flow Diagram

2.0 Implementation

2.1 UVM(Universal Verification Methodology)

Verification has evolved into a complex project that often spans internal and external teams, but the discontinuity associated with multiple, incompatible methodologies among those teams has limited the productivity. The Universal Verification Methodology (UVM) 1.2 Class Reference addresses verification complexity and inter-operability within companies and throughout the electronics industry for both novice and advanced teams while also providing consistency.

The UVM application programming interface (API) defines a standard for the creation, integration, and extension of UVM Verification Components (UVCs) and verification environments that scale from block to system. The UVM 1.2 Class Reference is independent of any specific design processes and is complete for the construction of verification environments.



UVM Test Example

2.2 DPI(Direct Programming Interface)

SystemVerilog DPI (Direct Programming Interface) is an interface which can be used to interface SystemVerilog with foreign languages. These Foreign languages can be C, C++, System C or others as well. DPIs consist of two layers: A SystemVerilog Layer and a Foreign language layer. Both the layers are isolated from each other.

The programming languages actually used as the foreign language are transparent and is indepedent of the SystemVerilog side of this interface. Neither the SystemVerilog compiler nor the foreign language compiler is required to analyze the source code that is interfaced. Different programming languages can be used and supported with the same intact SystemVerilog layer.

2.3 MLA(Machine Learning Algorithms)

Machine learning is the term used for processes that provide computers with the ability to learn without being explicitly programmed. Machine learning focuses on the development of computer programs that can change when exposed to new data.

There are three types of learning in MLA.They are:

- ***Supervised Learning***: Supervised learning is the machine learning task of inferring a function from labeled training data. The training data consist of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal).
    - A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances.
- ***Unsupervised Learning***: Unsupervised learning is a type of machine learning algorithm used to draw inferences from data sets consisting of input data without labeled responses.
    - The most common unsupervised learning method is cluster analysis, which is used for exploratory data analysis to find hidden patterns or grouping in data. The clusters are modeled using a measure of similarity which is defined upon metrics such as Euclidean or probabilistic distance.
- ***Reinforced Learning***: Reinforcement Learning or Reinforced learning is an area of machine learning inspired by behaviorist psychology, concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. The problem, due to its generality, is studied in many other disciplines, such as game theory, control theory, operations research, information theory, simulation-based optimization, multi-agent systems, swarm intelligence, statistics, and genetic

algorithms. In the operations research and control literature, the field where reinforcement learning methods are studied is called approximate dynamic programming. The problem has been studied in the theory of optimal control, though most studies are concerned with the existence of optimal solutions and their characterization, and not with the learning or approximation aspects. In economics and game theory, reinforcement learning may be used to explain how equilibrium may arise under bounded rationality.

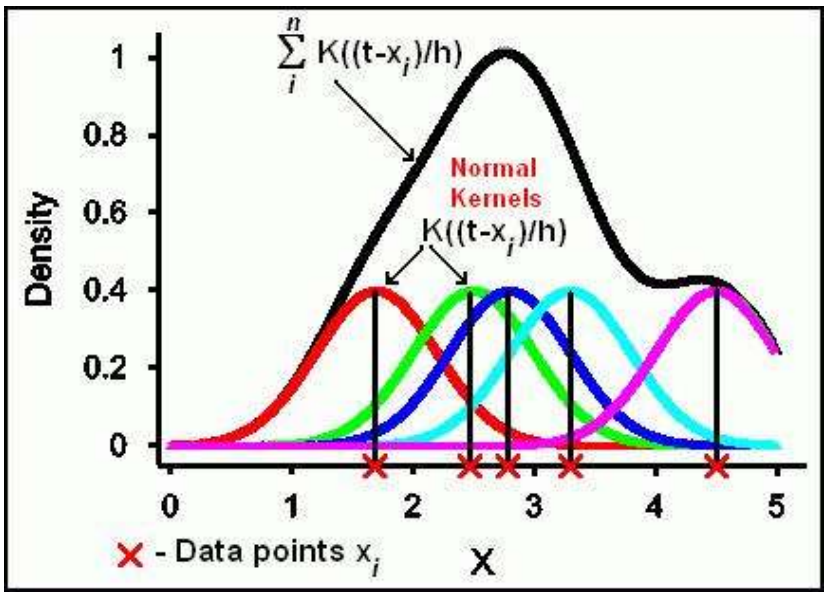
2.31 Selective KDE(Kernel Density Estimation)

In statistics, kernel density estimation (KDE) is a non-parametric way to estimate the probability density function of a random variable. Kernel density estimation is a fundamental data smoothing problem where inferences about the population are made, based on a finite data sample.

In standard KDE,a range is divided into many parts called kernels and data points are populated into the range.The inter kernel Gaussian curves are calculated and the overall curve for the range is obtained by curve smoothing techniques.Here we are using a slightly modified KDE.Here we take the range and divide it into many kernels and populate the data.After populating,we identify and select the kernel with highest number of data points and start constrained random verification for that kernel.After completion,the next highest kernel is verified and so on.The kernels with very less data points can be verified in directed testing.

This method just so happens to be suitable for DUT's which have gaps in the functional range of values,i.e some values are undefined in the range of functioning of the DUT.In such cases,the kernels will have zero or in extreme cases small number of data points.These can be ignored in constrained random verification and left for directed testing.

Examples for such DUT are memory fields,special registers, protocol testers, etc.



KDE Representation

2.32 Temporal Analysis

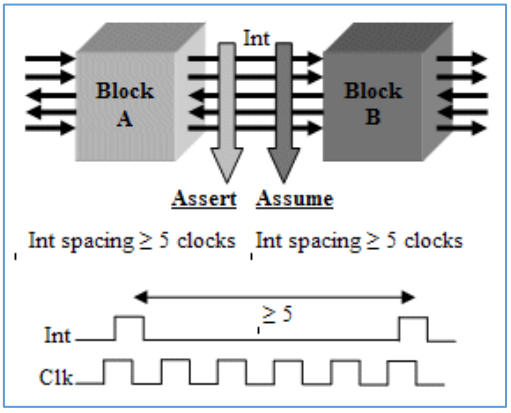
Temporal statistical analysis enables you to examine and model the behavior of a variable in a data set over time (e.g., to determine whether and how concentrations are changing over time. The behavior of a variable in a data set over time can be modeled as a function of previous data points of the same series, with or without extraneous, random influences.

Temporal Analysis of a motherboard chip

There are built in temporal analysis components in SystemVerilog which help us to model the characteristics and changes in the concentration of a variable in a data set.These components are named Assertion, Sequence and Property.This method is useful with DUTs which have data variables which can vary oover large ranges and behavior of the DUT changes with the data value.Examples are Counters,Shift Registers,etc.

Assertion:

Assertions are primarily used to validate the behaviour of a design. ("Is it working correctly?") They may also be used to provide functional coverage information for a design ("How good is the test?"). Assertions can be checked dynamically by simulation, or statically by a separate property checker tool – i.e. a formal verification tool that proves whether or not a design meets its specification. Such tools may require certain assumptions about the design's behaviour to be specified.



Assertion Example

Property:

Property is used to check whether the design is producing these kind of sequential behavior in the way it is supposed to generate or not.

```
1  property p_with_var_delay;
2    int v_cnt;
3    (start_sig, v_cnt = var_del + 1'b1)  |->  (
4    (v_cnt > 0, v_cnt = v_cnt - 1)[*1:$]
5    ##1 (end_sig && v_cnt >= 0)) ;
6  endproperty : p_with_var_delay
```

Property Example

Sequences:

A sequence is a list of boolean expressions in a linear order of increasing time. The sequence is true over time if the boolean expressions are true at the specific clock ticks. The expressions used in sequences are interpreted in the same way as the condition of a procedural if statement.

```
sequence s1
  a ##[3:5] b;
endsequence

sequence s2
  1 ##[3:7] (c ##3 d);
endsequence

property p1
  s1 |-> s2;
endproperty

assert a1
  assert p1 else ... ;
endassert
```

Sequence Example

3.0 Conclusion

RTL Used for Testing : AMBA-APB Bridge Converter

- Without the use of MLA, after 10000 clock cycles, only 71% of verification was over.
- With the addition of the MLA, after 10000 clock cycles, 85% of verification was over. By optimizing the MLA we can expect a much better result than constrained random test bench.

4.0 References

1) UVM Class Reference Manual 1.2.pdf

2) https://verificationacademy.com/verification-horizons/june-2014-volume-10-issue-2/UVM-Testbench-Structure-and-Coverage-Improvement-in-a-Mixed-Signal-Verification-Environment

3) http://infocenter.arm.com/help/topic/com.arm.doc.ddi0367b/Jgbiaici.html

4) https://en.wikipedia.org/wiki/Supervised_learning

5) https://www.mathworks.com/discovery/unsupervised-learning.html

6) https://www.bdti.com/InsideDSP/2012/09/05/MathWorks

7) https://www.mathworks.com/examples/matlab-computer-vision/mw/vision_product-VisionRecovertformCodeGenerationExample-introduction-to-code-generation-with-feature-matching-and-registration

8) Dr. Damodharan V. S., Mr. Rengarajan.V, “Innovative Methods of Testing and Verification”

9) https://www.mathworks.com/examples/matlab-coder/mw/coder_product-coderdemo_hello_world-hello-world

10) System Verilog for Verification by Chris Spear