FUNCTIONAL COVERAGE ANALYSIS OF OVM BASED VERIFICATION OF H.264 CAVLD SLICE HEADER DECODER

Akhilesh Kumar and Chandan Kumar

Department of E&C Engineering, NIT Jamshedpur, Jharkhand, India

ABSTRACT

Commercial chip design verification is a complex activity involving many abstraction levels (such as architectural, register transfer, gate, switch, circuit, fabrication), many different aspects of design (such as timing, speed, functional, power, reliability and manufacturability) and many different design styles (such as ASIC, full custom, semi-custom, memory, cores, and asynchronous). In this paper, functional coverage analysis of verification of RTL (Register Transfer Level) design of H.264 CAVLD (context-based adaptive variable length decoding) slice header decoder using SystemVerilog implementation of OVM (open verification methodology) is presented. The methodology used for verification is OVM which has gathered very positive press coverage, including awards from magazines and industry organizations. There is no doubt that the OVM is one of the biggest stories in recent EDA (electronic design automation) history. The SystemVerilog language is at the heart of the OVM which inherited features from Verilog HDL, VHDL, C, C++ and adopted by IEEE as hardware description and verification language in 2005. The verification environment developed in OVM provides multiple levels of reuse, both within projects and between projects. Emphasis is put onto the actual usage of the verification components and functional coverage. The whole verification is done using SystemVerilog hardware description and verification language. We are using QuestaSim 6.6b for simulation.

KEYWORDS: *Functional coverage analysis, RTL (Register Transfer Level) design, CAVLD (context-based adaptive variable length decoding), slice header decoder, OVM (open verification methodology), SystemVerilog, EDA (electronic design automation).*

I. INTRODUCTION

Verification is the process which proceeds parallel as design creation process. The goal of verification is not only finding the bugs but of proving or disproving the correctness of a system with respect to strict specifications regarding the system [2].

Design verification is an essential step in the development of any product. Today, designs can no longer be sufficiently verified by ad-hoc testing and monitoring methodologies. More and more designs incorporate complex functionalities, employ reusable design components, and fully utilize the multi-million gate counts offered by chip vendors. To test these complex systems, too much time is spent constructing tests as design deficiencies are discovered, requiring test benches to be rewritten or modified, as the previous test bench code did not address the newly discovered complexity. This process of working through the bugs causes defects in the test benches themselves. Such difficulties occur because there is no effective way of specifying what is to be exercised and verified against the intended functionality [11]. Verification of RTL design using SystemVerilog implementation of OVM dramatically improves the efficiency of verifying correct behavior, detecting bugs and fixing bugs throughout the design process. It raises the level of verification from RTL and signal level to a level where users can develop tests and debug their designs closer to design specifications. It encompasses and facilitates abstractions such as transactions and properties. Consequently, design functions are exercised efficiently (with minimum required time) and monitored effectively by detecting hard-to-

find bugs [7]. This technique addresses current needs of reducing manpower and time and the anticipated complications of designing and verifying complex systems in the future.

1.1. Importance of Verification

- When a designer verifies her/his own design - then she/he is verifying her/his own interpretation of the design - not the specification.
- Verification consumes 50% to 70% of effort of the design cycle.
- Twice more Verification engineers than RTL designer.
- Finding bug in customer's environment can cost hundreds of millions.

1.2. Cost of the Bugs

Bugs found early in design have little cost. Finding a bug at chip/system level has moderate cost. A bug at system/chip level requires more debug time and isolation time. It could require new algorithm, which could affect schedule and cause board rework. Finding a bug in System Test (test floor) requires re-spin of a chip. Finding a bug after customer delivery cost millions.

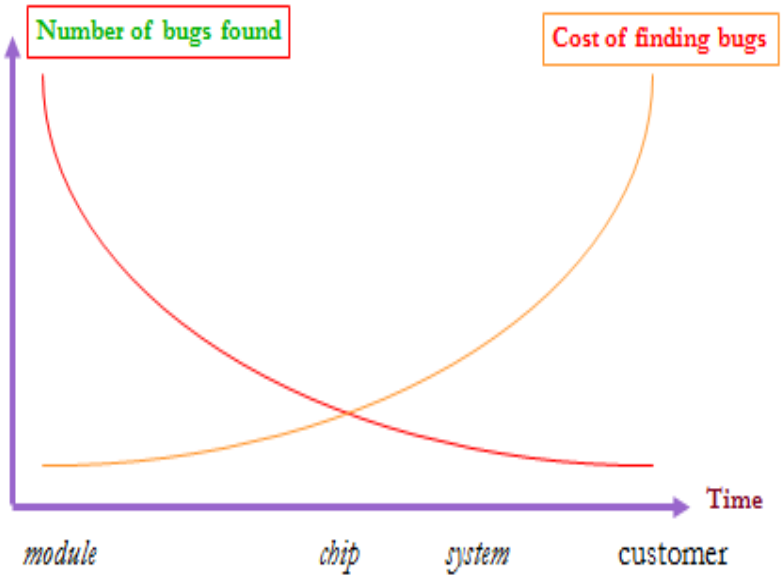

Figure 1. Cost of bugs over time.

II. SLICE HEADER

2.1. THE H.264 SYNTAX

H.264 provides a clearly defined format or syntax for representing compressed video and related information [1]. Fig. 2 shows an overview of the structure of the H.264 syntax. At the top level, an H.264 sequence consists of a series of 'packets' or Network Adaptation Layer Units, NAL Units or NALUs. These can include parameter sets containing key parameters that are used by the decoder to correctly decode the video data and slices, which are coded video frames or parts of video frames.

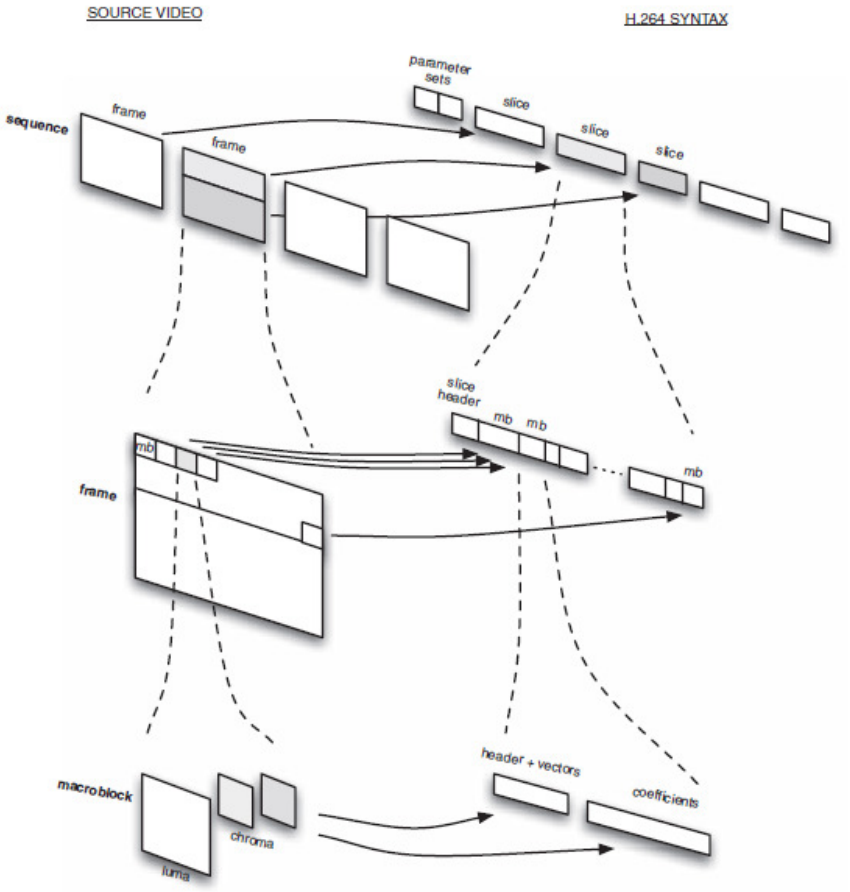

Figure 2. H.264 Syntax [1]

2.2. SLICE

A slice represents all or part of a coded video frame and consists of a number of coded macro blocks, each containing compressed data corresponding to a 16 × 16 block of displayed pixels in a video frame.

2.3. SLICE HEADER

Supplemental data placed at the beginning of slice is Slice Header.

III. SLICE HEADER DECODER

An H.264 video decoder carries out the complementary processes of decoding, inverse transform and reconstruction to produce a decoded video sequence [1].
Slice header decoder is a part of H.264 video decoder. Slice header decoder module takes the input bit stream from Bit stream parser module.

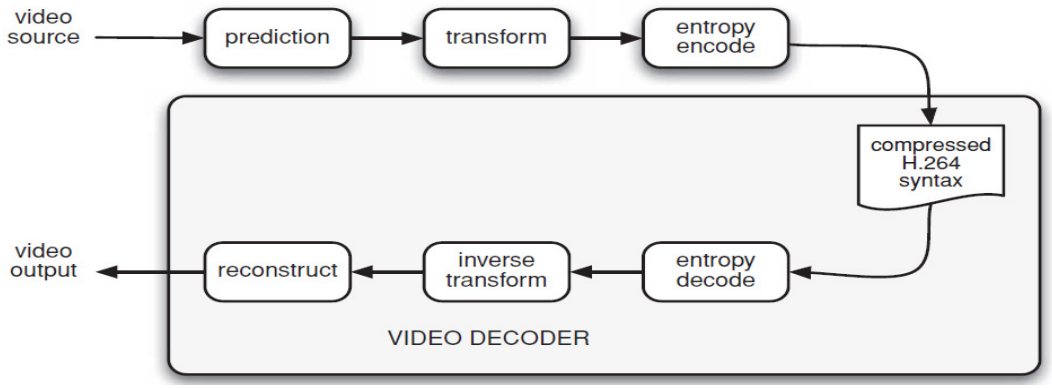

Figure 3. H.264 video coding and decoding process [1]

The slice header decoder module parses the slice header-RBSP (raw byte sequence payload) bit stream to generate first MB in slice, Slice type etc. The module sends the decoded syntax element to the controller.

IV. CAVLD

Context-adaptive variable-length decoding (CAVLD) is a form of entropy decoding used in H.264/MPEG-4 AVC video decoding. It is an inherently lossless decompression technique, like almost all entropy-decoders.

V. SYSTEM VERILOG

SystemVerilog is a combined Hardware Description Language (HDL) and Hardware Verification Language (HVL) based on extensions to Verilog HDL. SystemVerilog becomes an official IEEE standard in 2005. SystemVerilog is the extension of the IEEE Verilog 2001. It has features inherited from Verilog HDL, VHDL, C, C++. One of the most important features of SystemVerilog is that it's an **object oriented language** [4]. SystemVerilog is rapidly getting accepted as the next generation HDL for System Design, Verification and Synthesis. As a single unified design and verification language, SystemVerilog has garnered tremendous industry interest, and support [9].

VI. OVM (OPEN VERIFICATION METHODOLOGY)

The Open Verification Methodology (OVM) is a documented methodology with a supporting building-block library for the verification of semiconductor chip designs [8].
The OVM was announced in 2007 by Cadence Design Systems and Mentor Graphics as a joint effort to provide a common methodology for SystemVerilog verification. After several months of extensive validation by early users and partners, the OVM is now available to everyone. The term "everyone" means just that everyone, even EDA competitors, can go to the OVM World site and download the library, documentation, and usage examples for the methodology [7].
OVM provides the best framework to achieve coverage-driven verification (CDV). CDV combines automatic test generation, self-checking testbenches, and coverage metrics to significantly reduce the time spent verifying a design [2]. The purpose of CDV is to:

- Eliminate the effort and time spent creating hundreds of tests.
- Ensure thorough verification using up-front goal setting.
- Receive early error notifications and deploy run-time checking and error analysis to simplify debugging.

VII. OVM TESTBENCH

A testbench is a virtual environment used to verify the correctness of a design. The OVM testbench is composed of reusable verification environments called OVM verification components (OVCs). An

OVC is an encapsulated, ready-to-use, configurable verification environment for an interface protocol, a design submodule, or a full system. Each OVC follows a consistent architecture and consists of a complete set of elements for stimulating, checking, and collecting coverage information for a specific protocol or design.

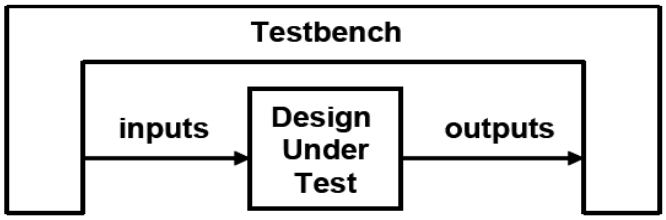

Fig 4. Testbench [2].

VIII. DEVELOPMENT OF OVM VERIFICATION COMPONENTS

SystemVerilog OVM Class Library:

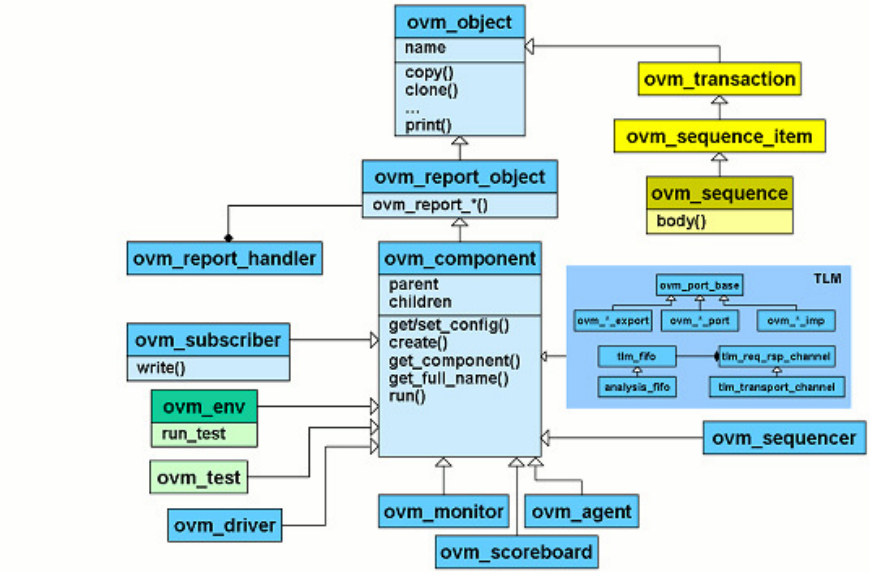

Figure 5. OVM Class Library [3]

The SystemVerilog OVM Class Library provides all the building blocks to quickly develop well-constructed, reusable, verification components and test environments [3]. The library consists of base classes, utilities, and macros. Different verification components are developed by deriving the base classes, utilities, and macros.

The OVM class library allows users in the creation of sequential constrained-random stimulus which helps collect and analyze the functional coverage and the information obtained, and include assertions as members of those configurable test-bench environments.

The OVM Verification Components (OVCs) written in SystemVerilog code is structured as follows [4]:

— Interface to the design-under-test
— Design-under-test (or DUT)
— Verification environment (or testbench)
  — Transaction (Data Item)
  — Sequencer (stimulus generator)
  — Driver
  — Top-level of verification environment

— Instantiation of sequencer
— Instantiation of driver
— Response checking
— Monitor
— Scoreboard
— Top-level module
— Instantiation of interface
— Instantiation of design-under-test
— Test, which instantiates the verification environment
— Process to run the test

Interface: Interface is nothing but bundle of wires which is used for communication between DUT(Design Under Test) and verification environment(testbench). The clock can be part of the interface or a separate port [2].

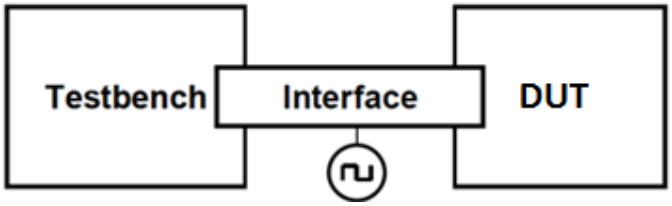

Figure 6. Interface [2]

Here, all the Slice Header Decoder signals are mentioned along with their correct data types. A modport is defined showing connections with respect to the verification environment.
Design Under Test (DUT): DUT completely describes the working model of Slice Header Decoder written in Hardware Description Language which has to be tested and verified.
Transaction (Data Item): Data items represent the input to the DUT. The sequencer which creates the random transactions are then retrieved by the driver and hence used to stimulate the pins of the DUT. Since we use a sequencer, the transaction class has to be derived from the ovm_sequence_item class, which is a subclass of ovm_transaction. By intelligently randomizing data item fields using SystemVerilog constraints, one can create a large number of meaningful tests and maximize coverage.
Sequencer: A sequencer is an advanced stimulus generator that controls the items that are provided to the driver for execution. By default, a sequencer behaves similarly to a simple stimulus generator and returns a random data item upon request from the driver. It allows to add constraints to the data item class in order to control the distribution of randomized values.
Driver: The DUT's inputs are driven by the driver that runs single commands such as bus read or write. A typical driver repeatedly receives a data item and drives it to the DUT by sampling and driving the DUT signals.
Monitor: The DUT's output drives the monitor that takes signal transitions and groups them together into commands. A monitor is a passive entity that samples DUT signals but does not drive them. Monitors collect coverage information and perform checking.
Agent: Agent encapsulates a driver, sequencer, and monitor. Agents can emulate and verify DUT devices. OVCs can contain more than one agent. Some agents (for example, master or transmit agents) initiate transactions to the DUT, while other agents (slave or receive agents) react to transaction requests.
Scoreboard: It is a very crucial element of a self-checking environment. Typically, a scoreboard verifies whether there has been proper operation of your design at a functional level.
Environment: The environment (env) is the top-level component of the OVC. The environment class (ovm_env) is architected to provide a flexible, reusable, and extendable verification component. The main function of the environment class is to model behaviour by generating constrained-random traffic, monitoring DUT responses, checking the validity of the protocol activity, and collecting coverage.

Test: The test configures the verification environment to apply a specific stimulus to the DUT. It creates an instance of the environment to invoke the environment.
Top-level module: A single top-level module connects the DUT with the verification environment through the interface instance. Global clock pulses are created here. run_test is use to run the verification process. global_stop_request is used to stop the verification process after a specified period of time or number of iterations or after a threshold value of coverage.

IX. FUNCTIONAL COVERAGE ANALYSIS

9.1. Coverage

As designs become more complex, the only effective way to verify them thoroughly is with constrained-random testing (CRT). This approach avoids the tedium of writing individual directed tests, one for each feature in the design. If the testbench is taking a random walk through the space of all design states, one can gauge the progress using coverage.
Coverage is a generic term for measuring progress to complete design verification. The coverage tools gather information during a simulation and then post-process it to produce a coverage report. One can use this report to look for coverage holes and then modify existing tests or create new ones to fill the holes. This iterative process continues until desired coverage level.

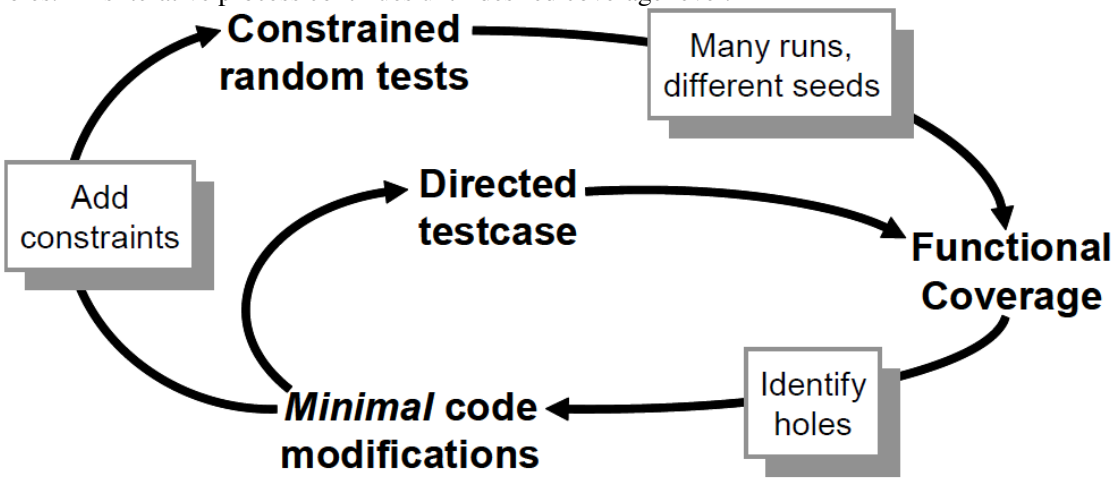

Figure 7. Coverage convergence [2]

9.2. Functional Coverage

Functional coverage is a measure of which design features have been exercised by the tests. Functional coverage is tied to the design intent and is sometimes called “specification coverage”. One can run the same random testbench over and over, simply by changing the random seed, to generate new stimulus. Each individual simulation generates a database of functional coverage information. By merging all this information together, overall progress can be measured using functional coverage. Functional coverage information is only valid for a successful simulation. When a simulation fails because there is a design bug, the coverage information must be discarded. The coverage data measures how many items in the verification plan are complete, and this plan is based on the design specification. If the design does not match the specification, the coverage data is useless. Reaching for 100% functional coverage forces to think more about what anyone want to observe and how one can direct the design into those states.

9.3. Cover Points

A cover point records the observed values of a single variable or expression.

9.4. BINS

Bins are the basic units of measurement for functional coverage. When one specify a variable or expression in a cover point, SystemVerilog creates a number of "bins" to record how many times each value has been seen. If variable is of 3-bits, maximum number of bins created by SystemVerilog is eight.

9.5. Cover Group

Multiple cover points that are sampled at the same time (such as when a transaction completes) are place together in a cover group.

X. SIMULATION RESULT OF OVM BASED VERIFICATION OF H.264 CAVLD SLICE HEADER DECODER

We use the QuestaSim 6.6b for simulation. Sequencer produces the sequences of data (transitions) which is send to the DUT through the driver which converts the transactions into pin level activity. The monitor keep track with the exercising of the DUT and its response and gives a record of coverage of the DUT for the test performed. Figure 8 showing the simulation result of coverage with cover groups. Total numbers of cover groups in the verification of Slice Header Decoder are thirty. Inside a cover group, a number of cover points are present and inside a cover point, a number of bins are present. We are considering a cover group CV_CAVLD_SH_09.

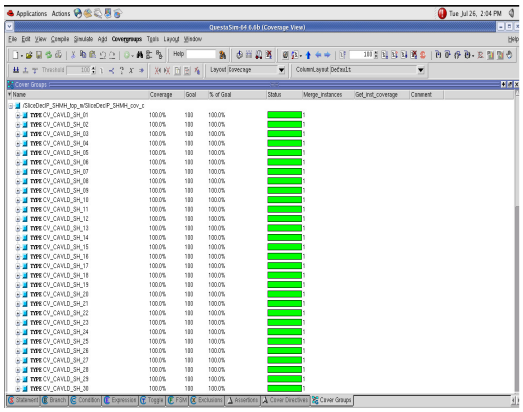

Figure 8. Simulation result of coverage

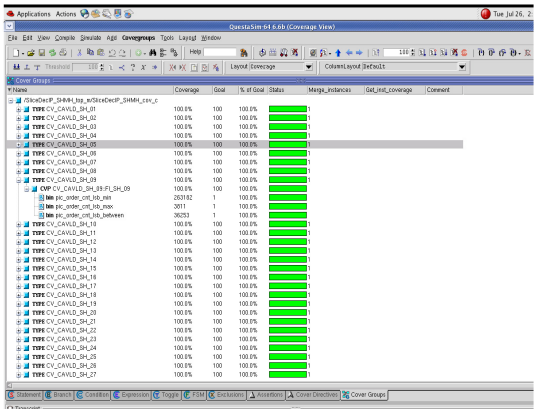

Figure 9. Simulation result of coverage with coverpoints and bins

Figure 9 shows the cover point (FI_SH_09) and bins inside the cover group CV_CAVLD_SH_09. The whole coverage report is very large and is not possible to include in this paper. We are including the coverage report related to cover group CV_CAVLD_SH_09.

Coverage reropt:

```
COVERGROUP COVERAGE:
--------------------------------------------------------------------------------
Covergroup                                              Metric      Goal/ Status
                                                                   At Least
--------------------------------------------------------------------------------
TYPE /CV_CAVLD_SH_09
                                                        100.0%         100 Covered
    Coverpoint CV_CAVLD_SH_09::FI_SH_09                 100.0%         100 Covered
        covered/total bins:                                  3           3
        missing/total bins:                                  0           3
        bin pic_order_cnt_lsb_min                       263182           1 Covered
        bin pic_order_cnt_lsb_max                         3811           1 Covered
        bin pic_order_cnt_lsb_between                    36253           1 Covered
```

The number (Metric) present in front of bins represents the number of hits of a particular bin appears during simulation.

XI. CONCLUSION

We present the verification of H.264 CAVLD Slice Header Decoder using SystemVerilog implementation of OVM. We analyze the functional coverage with cover groups, cover points, and bins. We achieve the 100 percent functional coverage for Slice Header Decoder module. Since coverage is 100%, hence the RTL design meets the desired specifications of slice header decoder.

REFERENCES

[1] 'THE H.264 ADVANCED VIDEO COMPRESSION STANDARD', Second Edition by Iain E. Richardson.
[2] 'SystemVerilog for Verification: A Guide to Learning the Testbench Language Features' by Chris Spear s.l. : Springer, 2006.
[3] OVM User Guide, Version 2.1.1, March 2010.
[4] http://www.doulos.com/knowhow.
[5] http://www.ovmworld.org.
[6] H.264: International Telecommunication Union, Recommendation ITU-TH.264: Advanced Video Coding for Generic Audiovisual Services, ITU-T, 2003.
[7] 'Open Verification Methodology: Fulfilling the Promise of SystemVerilog' by Thomas L. Anderson, Product Marketing Director Cadence Design Systems, Inc.
[8] O.Cadenas and E.Todorovich, 'Experiences applying OVM 2.0 to an 8B/10B RTL design', IEEE 5th Southern Conference on Programmable Logic, 2009, pp. 1 - 8.
[9] P.D. Mulani, 'SoC Level Verification Using SystemVerilog', IEEE 2nd International Conference on Emerging Trends in Engineering and Technology (ICETET), 2009, pp. 378 - 380.
[10] G. Gennari, D. Bagni, A.Borneo and L. Pezzoni, 'Slice header reconstruction for H.264/AVC robust decoders', IEEE 7th Workshop on Multimedia Signal Processing (2005), pp. 1 - 4.
[11] C. Pixley et al., 'Commercial design verification: methodology and tools', IEEE International Conference on Test Proceedings, 1996, pp. 839 - 848.

Authors

Akhilesh Kumar received B.Tech degree from Bhagalpur university, Bihar, India in 1986 and M.Tech degree from Ranchi, Bihar, India in 1993. He has been working in teaching and research profession since 1989. He is now working as H.O.D. in Department of Electronics and Communication Engineering at N.I.T. Jamshedpur, Jharkhand, India. His interested field of research digital circuit design.

Chandan Kumar received B. E. Degree from Visveswarya Technological University, Belgaum, Karnataka, India in 2009. Currently pursuing M. Tech project work under guidance of Prof. Akhilesh Kumar in the Department of Electronics & Communication Engg, N. I. T., Jamshedpur. Interest of field is ASIC Design & Verification, and Image Processing.