Metric Driven Verification of Mixed-Signal Designs

Neyaz Khan
Cadence Design Systems
6400 International Pkwy, Suite 1500
Plano, TX, 75093
+1-972-618-8193
nkhan@cadence.com

Yaron Kashai
Cadence Design Systems
2655 Seely Avenue
San Jose, CA, 94087
+1-408-914-6335
yaron@cadence.com

Hao Fang
LSI Corporation
1230 Northland Drive
Mendota Heights, MN, 55120
+1-651-675-3140
Hao.Fang@lsi.com

ABSTRACT
Functional verification has long been a major concern in digital design. Over the years, the huge investment in verification spurred the development of tools and methodologies for systematic and cost-effective functional verification. In the last several years, a similar need is building in the analog design space. In this paper, we present an approach leveraging *Metric Driven Verification* (MDV) to address functional verification of analog and mixed-signal designs. The proposed verification methodology scales from a single IP block to a full system-on-a-chip (SoC), and is compatible with current digital-only verification methodologies.

Categories and Subject Descriptors
D.3.4 **[Mixed-Signal Verification]:** *Verification techniques, Mixed-signal design, coverage-driven verification***.**

General Terms
Verification, Measurement.

Keywords
Mixed-signal Functional Design Verification, Metric Driven Verification (MDV), Universal Verification Methodology (UVM). Analog Modeling.

1. INTRODUCTION
Analog design verification has been getting more attention in the last several years [6]. This is due, in part, to the fact that today's analog designs are richer in functionality, with many digital controls and configurations. Another key change is the close integration between the digital and analog blocks, for instance a calibration loop can involve digital control along with analog circuits, and sometimes software as well. A third consideration is that most SoCs today have significant analog IP, hence functional verification at the full chip level requires addressing the analog circuits along with the digital ones. Current practice draws a firm partition between analog and digital design. Analog IP blocks are designed and verified by the analog team, most often by running interactive simulations and sometimes relying on visual inspection of results. The chip integrator may receive some highly abstracted digital models, often nothing more than a boundary representing the analog IP. Hence simulations at the SoC level are ineffective at exposing functionality and integration errors of the analog IP. This state of affairs led to a growing number of failures and re-spins that are blamed on the analog-digital integration.

The industry is responding to this challenge by improving the integration between analog and digital simulation environments and extending assertion based verification to apply to analog modules as well [9]. Advanced modeling techniques are introduced to speed up the simulation of analog circuits [4].

A main challenge is the large number of configuration and programming options available for most analog IP blocks. The large number of configuration and programming combinations compounds the verification challenge for such blocks. To address this problem we chose to leverage metric driven verification (MDV) [1], adapting it for use with analog circuits both at the block level and the SoC level. MDV offers a systematic and repeatable approach to verification, starting with a verification plan that is quantitative and measurable. A test bench is then constructed in which automatic stimulus generation and checking are performed. Input stimulus and design behavior are monitored, and coverage data is collected. Finally, coverage results are reviewed against the goals stated in the verification plan. This tight feedback ensures rapid convergence of the verification effort.

We have implemented the proposed methodology on a leading-edge design from LSI Corporation. This proof-of-concept project served to validate the approach in terms of technical feasibility, overall cost, and acceptance by the analog design and design verification teams.

The rest of this paper is organized as follows: Section 2 revisits the main principles behind MDV and their adaptation to serve analog verification. Section 3 discusses the methodology in detail. Section 4 describes our experience verifying an advanced design using the proposed methodology. Section 5 summarizes and offers some conclusions.

2. MDV REVISITED
2.1 Guiding Principles of MDV
Metric Driven Verification is a broadly used concept for verifying large digital designs. Modern designs have state spaces so huge that only a tiny fraction of all possible combinations can be simulated. MDV helps achieve good functional coverage in these limiting circumstances. MDV is guided by a functional specification, rather than design implementation. The functional specification is parsed down to a hierarchy of features in a *verification plan*, where each feature can be shown to meet the specification by some measurement. These measurements are called *functional coverage*. The resulting functional coverage space is many orders of magnitude smaller than the design state space – making it a practical metric. A test bench is created to exercise the design, check its functionality

and measure coverage. Layers of automation are added to run large volumes of simulations with random perturbations. The collected coverage is aggregated and compared with the verification plan. Areas lacking in coverage are targeted to get an overall balanced coverage.

Some popular verification methodologies are based on the MDV concept (OVM [2,3,11], and more recently UVM [8,13]). These methodologies teach a specific style that is well supported by tools and libraries. Experience with these has demonstrated the effectiveness of the metric-driven approach for some of the most complex digital chips produced. The effectiveness of MDV in tackling the state space growth problem motivated us to explore a possible adaptation to analog and mixed-signal designs that exhibit similar growth.

2.2 Adapting MDV to Analog Design

Conceptually, applying MDV is straight forward: one should enumerate the functional features of the design, associate a measurement with each feature and simulate the design to collect sufficient coverage, indicating all functions are implemented correctly. Unfortunately there are a number of practical obstacles when analog designs are concerned. Some of the most prominent ones are discussed below.

Analog verification planning and coverage collection. Analog features are expressed in a terminology that is richer and broader than typical digital features. This implies that capturing analog features in verification planning tools requires some extensions. Furthermore, the measurement of analog functional coverage is more involved. Rather than measuring a logic value, analog properties may require the measurement of amplitude, gain, frequency, phase or similar values that are more difficult to measure.

Batch execution. A fundamental assumption for MDV is the ability to run a large number of simulations in an automated manner. The volume of simulations requires that stimulus is automatically generated and the test bench is self-checking. The approach is inherently incompatible with interactive simulation and manual inspection of results – which still dominates the analog verification practice.

High performance models. Accurate analog models, such as a Spice netlist, are very slow to simulate when compared to digital event-driven simulations. For the sake of functional verification, the accuracy of the model needs to be traded for higher performance. The use of more abstract models such as AMS or real number models (RNM) is required to support large number of simulations.

Constrained random stimulus. Input stimulus needs to be generated such that simulations explore different behaviors and cover the functional space. For analog designs, this means randomized control over both digital and analog inputs to the design. The problem of driving analog input stimulus that is effectively controlled by constraints and sequences is a major requirement.

Self-checking. Determining that an analog design works as planned is more involved and somewhat fuzzy when compared to digital design. Nevertheless, automatic checking must be implemented. Checking can be in the form of embedded assertions as well as more elaborate structures such as scoreboards.

We recognize that functional verification in the suggested vein does little to offset the verification work done by the analog team. The analog team will continue to be concerned with performance metrics specific to the design at hand. Looking at power consumption, frequency responses, transient responses and similar features while running Spice-level simulations is at the core of the analog design work. Performing MDV-style functional verification represents extra investment.

For a mixed-signal verification methodology to be acceptable, all of the challenges above must be addressed in a cost-effective manner. The additional investment is justified by the need to compensate for the growing risk introduced by adding digital control functionality. The work done at the IP block level is reusable at the SoC level, offering thorough mixed-signal verification that is mostly missing today.

3. THE UVM-MS METHODOLOGY

3.1 Main Concerns and Design Decisions

We have named the new methodology *Universal Verification Methodology – Mixed-Signal*, or UVM-MS. We have adopted the digital UVM methodology as a basis because of its broad presence and tool support. Specific additions make the methodology applicable to mixed-signal designs, including driving analog inputs and measuring analog metrics.

A primary design decision was to accommodate all modeling styles for the design under verification (DUV). Spice netlist models with high accuracy are readily available, but their simulation speed is low. AMS models and real number models offer progressively higher performance, but are an extra cost to create and they require verification with respect to the design they model. We felt that a methodology that restricted the use of particular model styles would be too limiting. Thankfully, the simulation environment and test bench tools at our disposal can support all the modeling styles above, as well as any mixture of those.

Another key decision is maintaining the abstraction level provided by modern *hardware verification languages* (HVLs). This is required to enable a smooth integration into SoC level verification environments. Top layers of an HVL test bench are closer to a software program than a hardware design. The generation and synchronization of inputs is performed by procedures called *sequences* [10]. To fit in, analog inputs would have to be controlled in a similar way.

The UVM-MS methodology needs to bridge the gap between abstract HVL constructs and the detailed driving and sampling of analog signals. We chose to address this by stacking HVL control code on top of a Verilog-AMS layer that implements the low level operations. This division takes advantage of the different semantic constructs available for each language. It also leads to more optimal runtime performance.

In the process of implementing the methodology, it became clear that a library of simple components could help a great deal in constructing new verification environments. Such components provide commonly used interfaces between analog signals and test bench functions, saving time and effort.

3.2 Tool Flow

The overall tool flow is depicted in Figure 1 below.

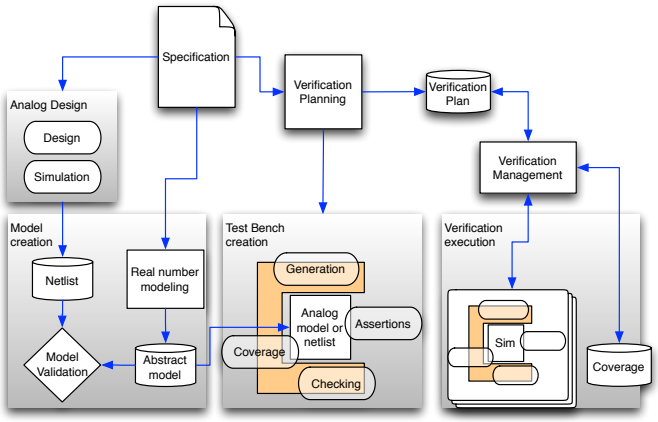

Figure 1: UVM-MS tool flow for analog IP verification

Both the analog design process and the verification process start with a specification. We recognize that a formal spec is not always available or may be in flux through much of the design process; hence the term is used here in a broader sense to include any documentation that drives the design process. The spec is used as an input to the verification planning process, where experts like the project lead spend time to enumerate the hierarchy of features that need verification. During planning, features are associated with the spec on one hand and with metrics that verify them on the other hand. A verification planning tool captures this information and creates a database called *vPlan*.

A verification environment is created based on the plan. The environment's architecture is somewhat specific to the DUV, and general guidelines are offered by UVM-MS, including the choice of library components that can be used as building blocks.

It probably makes sense to run the first few simulations using the Spice netlist created by the analog design process. These simulations serve to clean the test bench, but they are likely to be too slow for massive regression runs. It is therefore recommended that an appropriate abstract model be created – ideally an AMS or real number model that is high performance. If an abstract model is created, it needs to be validated against the specification and kept in sync with the original Spice netlist.

When both the test environment and the DUV model are in place, regression runs can commence. Accumulated coverage is mapped back to the plan, so at any time, the verification engineers can assess progress and tune the runs as needed. The process of directing simulations, spawning them and tracking their execution can be done manually, but a verification management tool is invaluable for organizing and automating these tasks.

As verification progresses, the number of errors uncovered drops and the accumulated coverage reaches an acceptable level, indicating the block is ready to be handed over to the integration team. A main advantage of the UVM-MS methodology is the ability to hand off both *verification intent*, in the form of the vPlan database, and *verification implementation* that includes units of the test environment and the abstract model of the DUV. Handing off verification aspects along with the design is a major help in planning and executing verification at the SoC level. It enables a deep and meaningful verification of the integration. Taking into account that many analog IP blocks are integrated into multiple SoCs over a longer period of time, the ability to archive and reuse verification artifacts along with the design is invaluable.

3.3 Verification Environment Architecture

A generic architecture of a verification environment is provided in Figure 2. In this simplified scheme, the DUV has a single analog input and output, and a digital control input. The analog input is driven by a signal generator block. The digital control block is connected to a digital driver. A specialized monitor samples the analog output. Both digital and analog sources are controlled by sequence drivers, which in turn are controlled by the test sequence. The test sequence can specify control operations as well as inject analog waveforms, synchronizing operations as needed.

The monitor component is sampling the analog output for the purpose of checking and coverage collection. Additional coverage and checking can be implemented by sampling DUV internal nodes and conditions. Such checking can use assertions as well as end-to-end checking schemes similar to scoreboards.

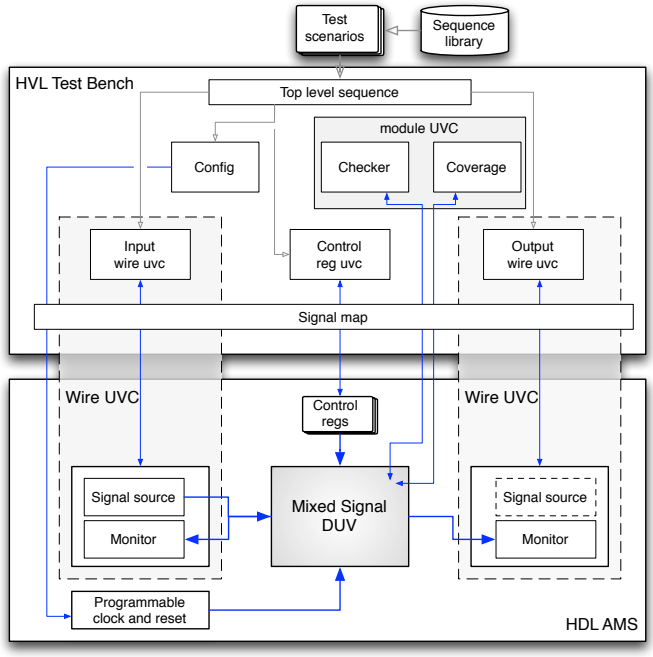

Figure 2: A generic UVM-MS architecture

A typical checking scheme for an analog DUV is illustrated below. The gain checker in Figure 3 samples both the input and output signals, providing the amplitude for each. The checker computes the measured gain and compares it with an expected value. The timing of the check is controlled by the top-level sequence that may also adjust control variables influencing the gain.

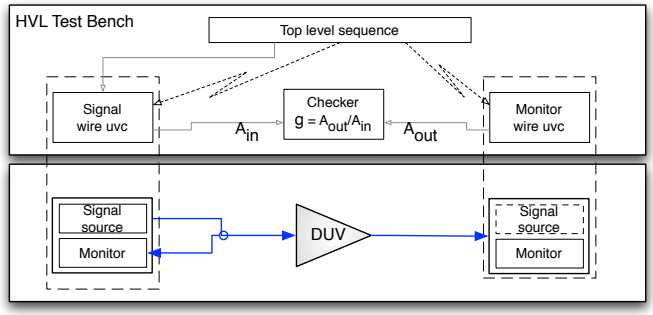

Figure 3: A typical end-to-end checker, verifying gain

3.4 Utilizing Verification Components

The architecture depicted in Figure 2 identifies some components that are commonly needed to drive and monitor an analog DUV. In

order to ease the methodology deployment, such components are collected in a library that is associated with the methodology. We have adopted the term *Universal Verification Component* (UVC) for these library components, following the UVM terminology, though we acknowledge that these components are much simpler and lower level than digital UVCs. Still, the similarity in role and in architecture justifies the name.

A prime example of a library component is the *wire* UVC, that function as a signal source and a signal monitor for high frequency analog sine wave signals. Figure 4 below illustrates the wire UVC architecture.

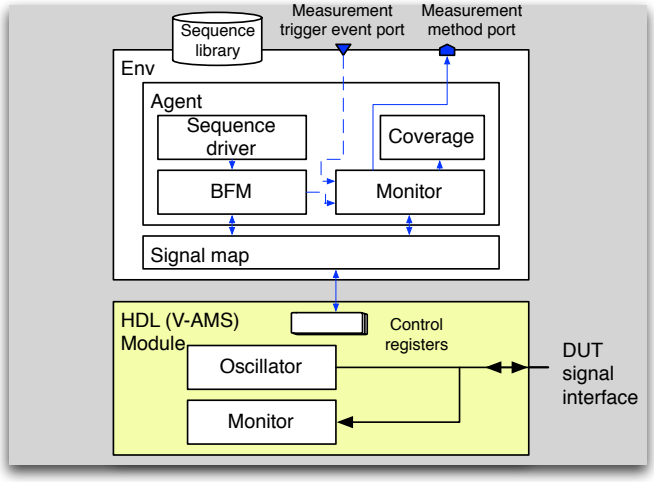

Figure 4: Wire UVC architecture

The wire UVC is implemented partly in HVL (***e*** [7] currently, though a SystemVerilog [3,12] version exists as well). The lower-level driver and monitor functions are implemented in Verilog-AMS. This allows for an efficient and flexible implementation that connects easily to electrical ports as well as real value ports.

The wire UVC controls the frequency, amplitude, phase and DC bias of the generated signal. Higher-level control is provided through a sequence library. The same signal properties are measured by the monitor and passed on to facilitate checking and coverage collection. Figure 5 below depicts the controlled signal properties and a sample sequence item.

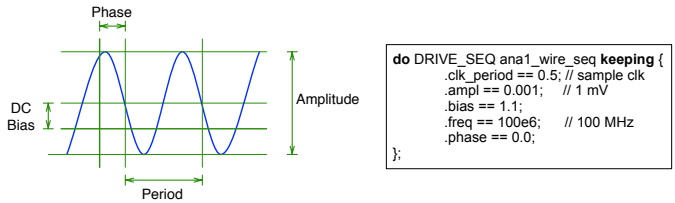

Figure 5: Sequence item controlling signal properties

Another common interface component is a *register* UVC that connects to digital control inputs. The register UVC is a much-simplified version of a register package UVC used extensively in digital verification. Other UVCs in the library include convertors between analog and digital values (ADC and DAC), as well as various ramp generators, threshold checkers and the like.

3.5 Leveraging Abstract Models

Developing abstract models for analog circuits is outside the scope of this paper. A good introduction is provided in [5]. Such abstract models can be detailed and accurate enough for functional verification, while offering significant speed increases. Real number models in particular, can execute in an event-driven simulation framework, which is inherently faster than analog solvers that are at the heart of analog simulation.

It should be noted that interfaces of abstract models can vary between real number ports and Verilog AMS ports of either *electrical* or *logic* disciplines. Since the test environment needs to support either modeling style, the test bench interfaces may need to adapt. The simulation environment we had access to automated this task by inserting *connect modules* that perform the necessary adaptation.

4. APPLYING UVM-MS

4.1 Design and Test Bench Architecture

For a proof-of-concept, we have targeted a current design from LSI Corporation. For a DUV, we have picked a major analog block featuring digitally controlled variable-gain-control functions along with filtering and fault detection. The DUV included advanced power management requiring careful sequencing of power-up and power-down operations. A specification was available for the design, as well as an informal test plan captured as an Excel spreadsheet. We have used both the spec and the informal plan to design and capture a formal verification plan, captured in a vPlan database. A tool called *Enterprise Planner* was used for this purpose.

The development of the test environment was divided into two phases. Phase 1 concentrated on driving and measuring the main control and analog interfaces, verifying gain functionality and power control. Phase 2 extended the environment with additional checking and implemented a more realistic signal source.

The overall test bench architecture is very similar to Figure 2 above. Several instances of the wire UVC were used to construct a signal source and an output monitor for the analog interfaces. The register UVC was used to control the numerous digital inputs to the DUV. An end-to-end gain checker similar to the one depicted in Figure 3 was implemented. The checker was triggered on any change in the setting of the control registers. Additional checkers and monitors were implemented to verify the fault detection and power control functionality.

The verification environment featured multiple sources that had to be coordinated and controlled. This was achieved by creating a top-level sequence controlling major interfaces. Sequences in UVM are hierarchical procedures that control the activation of test functions such as driving inputs and checking outputs. Sequences can synchronize to clocks, wait for events, fork sub-sequences and so on. A library of low-level sequences defines the basic control operations available. Test scenarios are implemented by combining such operations. Each sequence can have parameters that are determined when the sequence is called. Such parameters can be constrained and determining their value requires a *random generation* step, essentially solving a constraint satisfaction problem (CSP). Thanks to this randomness, the same sequence hierarchy can yield many different test scenarios that share a structure and explore a broad space around it.

Figure 6 below shows the structure of a simple test sequence. The sequence starts by activating the input signal source with specific frequency and amplitude parameters. Next, the power-up sub-sequence is applied to power up the DUV, an operation that involves multiple steps. Three different gain configurations are applied sequentially, each automatically triggering a gain check for the configured condition. Finally, the DUV is powered down by

applying the relevant sub-sequence, resulting in the DUV shutting down.

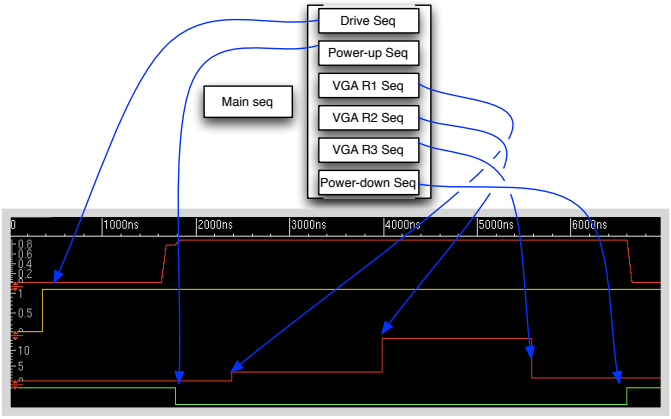

Figure 6: A simple test sequence

The test in Figure 6 is limited because the gain measurements are all performed for a single input frequency. A more sophisticated test is depicted in Figure 7. That test calls sub-sequences to setup the input frequency and gain controls to random values, while considering constraints that ensure only legal values are picked. Such tests achieve significant coverage growth when run repeatedly.

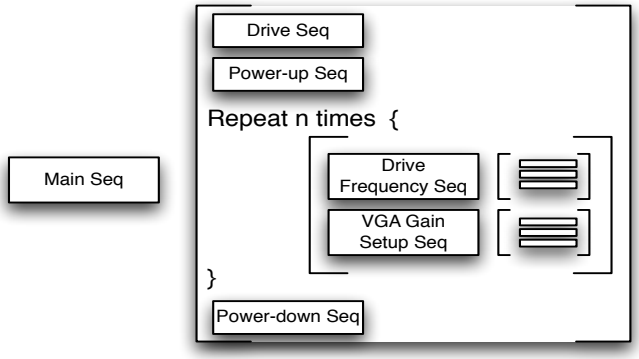

Figure 7: A compound test sequence

A model of the design was initially available as a Spice netlist, which was used in the first phase of the work. The netlist model was sufficiently fast to run simple tests, taking tens of minutes to execute. It was clear, however, that running massive regressions as well as system-level tests with the netlist model is impractical. To address these needs, a Verilog real number model using extended *wreal* support [10] was created. This model was integrated in phase 2 of the project.

4.2 Experimental Results

The proof-of-concept project was executed by the authors in about 3 months of calendar time. Verification planning took about half the time, during which team members familiarized themselves with the design and interacted with the analog design team to understand the requirements. Phase 1 of the implementation took about three weeks and phase 2 was completed in two weeks. An additional two weeks were spent creating and validating the real number model of the DUV. That work was done by an expert analog modeler.

The real number model of the DUV provided a 24X speedup over the Spice netlist simulation, while its level of inaccuracy was measured to be 0.3% (gain inaccuracy measured at 100 MHz). The model accuracy was far better than required, given that tolerances on checkers were in the range of 5%. The significant speedup allowed for very rapid regression runs. A whole suite of regressions would execute at the time it took for a single test using the netlist model. Even more importantly, running the real number model in a chip-level simulation became a practical option.

An automatic regression environment was created, using a verification management tool. That environment provided two levels of control over executing tests: At the test level, it allowed the application of constraints that determine the generation and sequencing of inputs for each and every run. At the regression session level, model parameters such as supply voltage, temperature and such could be controlled. Due to technical limitations, the simulator parameters needed to be compiled in with the build and elaboration process of the simulation. This restriction led us to a flow that involved two steps of generation: a first step happened during the *pre-session* phase, generating simulation parameters at random from a constrained set of values. The parameters generated are included in the build process. The resulting simulation model was run many times with different test configurations that were generated at random during each *execute* step. Figure 8 illustrates this setup.

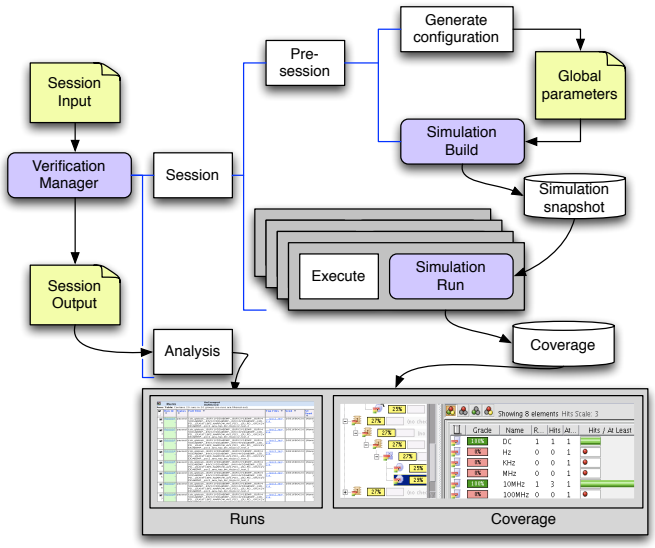

Figure 8: Verification management implementing two-step generation process

Each regression run resulted in a list of simulations with pass/fail indication, as well as aggregate coverage accumulated in a data file. Coverage information could be analyzed directly, but it was much preferred to map coverage back to the verification plan. The result of that process is illustrated in Figure 9. The verification plan is represented on the left pane, with coverage grade and color indication for each feature in the hierarchy. Detailed coverage for the selected feature is provided in the right hand side pane.

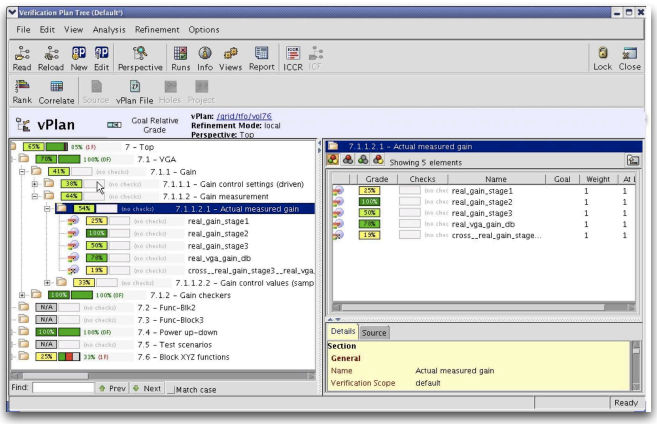

Figure 9: Analyzing coverage mapped to verification plan

We held in-depth reviews of the project at the end of each phase. The reviews involved the analog design team as well as verification experts not directly related to the design. The reviews were met with intense interest. The analog team had to be educated about some of the terms and concepts; yet after a quick introduction, team members were actively involved in analyzing results and proposing new checks and coverage metrics. The discussions during the planning phase and during the reviews exposed several ambiguities in the specification, especially as we considered the checks and metrics needed to verify a particular feature. These discussions provided valuable insight to all involved and stressed the rigorous nature of verification promoted by the UVM-MS methodology.

While the analog design team faced a significant hurdle tackling the tools and methods used by the methodology, the verification engineers involved in the review were quick to pick up the topic. Well versed in digital verification, they only required a short walk through of the verification environment to figure out how the architecture worked. They felt quite confident in their ability to maintain and extend the environment.

Verification exposed several suspect cases that were further investigated by the analog design team. Furthermore, both the DUV model and the test environment are available to be leveraged for SoC level integration.

5. CONCLUSIONS

The UVM-MS methodology presented here is the first mixed-signal verification methodology we are aware of that scales from analog IP blocks to mixed-signal SoCs. The methodology does introduce an extra cost at the IP verification level. This cost is justified as mitigation to the additional risk incurred due to the introduction of digital controls, sophisticated power management and broad configurability found in current analog circuits. The cost and effort implementing UVM-MS is minimized thanks to a library of small verification components. The investment at the IP level can be leveraged many times over, as the IP block is integrated into different SoCs.

Implementing the methodology on a real design provided some evidence of how effective MDV can be, even when applied to analog design. The positive reaction of the analog design team was an indication that this method can become a welcome addition to the design flow. The experiment also provided some insight to the associated costs and expertise needed. Based on this experience, we recommend that the verification team include some verification specialists along with the analog designers. The verification engineers should focus on architecting and implementing the test environment, while the analog engineers can provide the DUV model and lead the analysis and debug. The whole team should contribute to the planning phase.

We conclude that an MDV approach such as the one proposed in this paper is likely to become mainstream in mixed-signal design verification.

6. REFERENCES

[1] Carter, H and Hemmady, S, 2007, Metric Driven Design Verification, *Springer, ISBN: 978-0-387-38151-0*

[2] Glasser, Mark, 2009, Open Verification Methodology Cookbook, *Springer, ISBN: 978-1-4419-0967-1*

[3] Iman, S., 2008, Step-by-Step Functional Verification with SystemVerilog and OVM, *Hansen Brown Publishing, ISBN: 978-0-9816562-1-2*

[4] Intrinsix, 2010, A real solution for mixed signal SoC verification, *EE Times, http://www.eetimes.com/design/eda-design/4018835/A-real-solution-for-mixed-signal-SoC-verification*

[5] Joeres, S. Groh, H.-W. Heinen, S., 2007, Event driven analog modeling of RF frontends, *Behavioral Modeling and Simulation Workshop, 2007. BMAS 2007. IEEE International, ISBN: 978-1-4244-1567-0*

[6] Kundert, Ken and Chang, Henry, 2006. Verification of Complex Analog Circuits. *Proceeding of IEEE 2006 Custom Integrated Circuit Conf. (CICC)*

[7] Palnitkar, Samir, 2003, Design Verification with e, *Prentice Hall, ISBN: 978-0131413092*

[8] Rosenberg, S. and Meade, K., 2010, A Practical Guide to Adopting the Universal Verification Methodology (UVM), *Cadence Design Systems, ISBN 978-0-578-05995-6*

[9] www.accellera.org/activities/verilog-ams/

[10] Solutions for Mixed-Signal SoC Verification http://www.cadence.com/rl/Resources/white_papers/ms_soc_verification_wp.pdf

[11] www.ovmworld.org

[12] www.SystemVerilog.org

[13] www.uvmworld.org