# From Spec to Verification Closure: a case study of applying UVM-MS for first pass success to a complex Mixed-Signal SoC design

Neyaz Khan*
Maxim Integrated Products
14460 Maxim Drive
Dallas, TX, 75244
neyaz.khan@maxim-ic.com

Yaron Kashai
Cadence Design Systems
2655 Seely Avenue
San Jose, CA, 94087
yaron@cadence.com

## ABSTRACT

Why Metrics Driven Verification for Analog?

In their 2006 paper titled “Verification of Complex Analog Integrated Circuits” Kundert et al [1] write: *Functional complexity in analog, mixed-signal, and RF (A/RF) designs is increasing dramatically. Today’s simple A/ RF functional block such as an RF receiver or power management unit can have hundreds to thousands of control bits. A/RF designs implement many modes of operation for different standards, power saving modes, and calibration. Increasingly, catastrophic failures in chips are due to functional bugs, and not due to missed performance specifications. Functionally verifying A/ RF designs is a daunting task requiring a rigorous and systematic verification methodology. As occurred in digital design, analog verification is becoming a critical task that is distinct from design.*

Today, 5 years later, the challenges of analog verification have increased even further. Analog IP development is expensive, driving the reuse of IP in several SoCs. Each new integration brings about a new verification challenge because of configuration and connectivity changes. The verification of an analog block in the context of an SoC is tremendously difficult and time consuming. This highlights the need for thoroughly verified analog IP, as well as strong verification capabilities of the analog module in the SoC verification context.

Common methods of SoC level analog verification include *black-box verification* of the analog portion and the *capture-and-reply approach*. The black-box approach involves integrating a highly abstracted model of the analog functionality, often just an interface with a loop-back for the digital signals. This prevents simulation speed degradation, but provides very little in terms of verification - it is often impossible to tell if the analog portion functions correctly, or whether it is exercised to a desired degree. The capture- and-replay approach extracts the boundary of the analog portion from the SoC simulation at some specific times (after initial configuration for example). The waveform is then converted and replayed as input to an AMS simulation of the analog circuit. This method is better at exposing functional errors, but it is limited by the static nature of the analog-digital boundary captured. Key processes such as calibration require reactive feedback between the digital and analog portions, which is impossible to achieve using this method. Providing the exact timing and complex handshakes that may be required by the analog portion makes the capture process complicated and error prone. Furthermore, the process is manual, lacking automated checking, severally limiting the number of cases that can be tested.

The deficiencies described increase the risk involved in designing AMS SoCs. Analog circuits may not be hooked up correctly, may not function correctly and may not be driven as expected, while current verification methods may fail to detect the problems. Analog designs are often classified as “small A big D” or “small D big A” depending on the size of the analog content. Common wisdom suggests focusing on the “big” portion as the primary verification goal.

In contrast, it is claimed that the critical aspect for verification is the level of interaction between the digital and analog portions. If the interaction is significant and complex, as it tends to be in modern circuits, the verification methodology must address the design as a whole, applying to both analog and digital portions with the same level of automation and rigor.

Digital verification engineering emerged in the last 20 years as an indispensable part of chip design. As complexities grow and productivity pressures rise, the expansion of verification engineering into the analog space in the short term is inevitable. The extended methodology is named **UVM-MS**. Methodology extensions include verification planning for analog blocks, analog signal generation, checking and assertion techniques for analog properties and analyzing analog functional coverage [7]. The methodology features abstract, high level modeling of analog circuits using *real number modeling* (RNM). Automation and management aspects include batch execution and regression environments, as well as progress tracking with respect to the verification plan.

This paper will provide comprehensive details of how UVM-MS is applied to a real life example – in this case a Noise Cancelling Receiver (NCR) design as part of a MS-SoC.

**Highlights:**

- Creation of an executable verification plan (vPlan) for analog DUT
- Real Number Model (wreal) for the NCR
- UVM-MS based verification components that contains:
    - Digital MS based Analog signal generation using a wire UVC
    - Analog monitors that measure the envelope of a signal – with built in coverage
    - Driving and monitoring configurations controlled by analog sequences with programmable resolution
    - Functional Coverage collection on analog parameters in design
    - Mixed Signal assertion based checks that span between digital & analog
- Closing the loop – backannotating analog coverage & checks onto the vPlan for verification closure.

---

* Formerly of Cadence Design Systems

## Keywords

Mix-signal Functional Design Verification, Metric Driven Verification (MDV), Universal Verification Methodology (UVM). Analog Modeling, UVM-MS, MS-SoC

## 1. INTRODUCTION

The UVM methodology has succeeded in tackling the hardest verification challenges in digital design. It is a metric-driven approach using coverage directed random stimulus generation, supporting multiple verification languages. UVM promotes module-to-chip reuse and project-to-project reuse as means of maximizing development efficiency. It is the methodology of choice to be extended for supporting analog verification. The extended methodology is named **Universal Verification Methodology–Mixed-Signal (UVM-MS)** [7]. Methodology extensions include verification planning for analog blocks, analog signal generation, checking and assertion techniques for analog properties and analyzing analog functional coverage. The methodology features abstract, high-level modeling of analog circuits using real number modeling (RNM). Automation and management aspects include batch execution and regression environments, as well as progress tracking with respect to the verification plan.

Figure 1 depicts a high-level view of the methodology, spanning both IP and SoC-level verification. At the IP level, a verification plan is created, the analog circuit is modeled, and a test environment is put together. For best verification performance, the analog circuit should be modeled as an abstract RNM, though the methodology can also apply to designs modeled as AMS and/or SPICE. The flow of information is represented by the solid, angular arrows. The test plan, models and verification artifacts are reused at the SoC level, as indicated by the dashed, curved arrows.

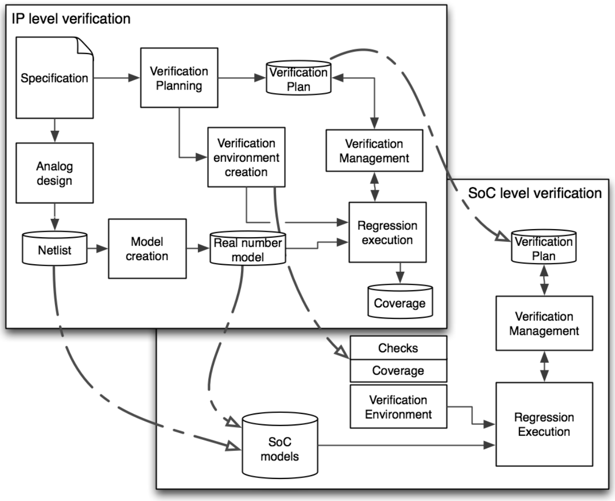


*Figure 1: Applying UVM-MS at the IP and SoC Level*

Several teams are expected to collaborate in verifying mixed signal designs. The analog architect and a team of analog designers are in charge of finalizing the specification, creating the design, and verifying its electrical properties. IP-level verification engineers will implement a test environment and carry out metric-driven verification of the IP. The system-level integrator will oversee the connection of the IP in the SoC. SoC verification engineers are in charge of integrating the IP-level test environment into the SoC level and execution of the SoC-level testing. This separation of roles and responsibilities highlights the different expertise required, in terms of domain knowledge, tools, languages and methods.

The application of UVM-MS starts at the analog IP or module level to ensure correctness and robustness. This is done in parallel to the development of the analog circuit, and augments the verification done by the analog designer using SPICE-level analyses. The analog designers' work is mostly interactive and manual, and augmenting it with a metric-driven regression environment greatly improves quality and reduces risk, especially in view of rapid spec and design changes. Methodology and design tools evolution will enable the introduction of UVM-MS concepts into the analog designers' work processes, leveraging the interactive work as way of authoring checks and coverage.

The IP verification team with the help of the analog design team puts together a verification plan. The plan has to outline the properties to be verified, the testing scenarios and coverage metrics that would ensure functional correctness. Subsequently, a UVM-MS verification environment is authored by the verification team. The verification environment is retarget-able to either the design netlist or an abstracted AMS or Real Number Model (**RNM**). If abstract models are used, a batch mechanism to maintain model validity with respect to the netlist is included. The creation of a metric-driven verification environment at the IP level is done **in addition** to the traditional analog design flow, and is **not intended to replace** any portion of it. This additional investment is required to meet quality and risk profiles given current design complexity, in particular:

- Verifying functionality and performance under all possible digital configurations, or a statistically meaningful portion thereof, if the number of combinations is too large.
- Verifying dynamic control scenarios, like calibration for example, where digital controls are tied into a converging feedback loop.
- Verifying that control transitions, such as power mode and test mode switching, do not disrupt analog functionality. Here too, the sheer number of possible combinations typically requires randomly generated scenarios.

The large number of tests required to address the concerns listed above necessitates a metric-driven approach, featuring random generation of stimulus. Directed testing cannot address these needs effectively. This IP-level process is depicted in Figure 2.

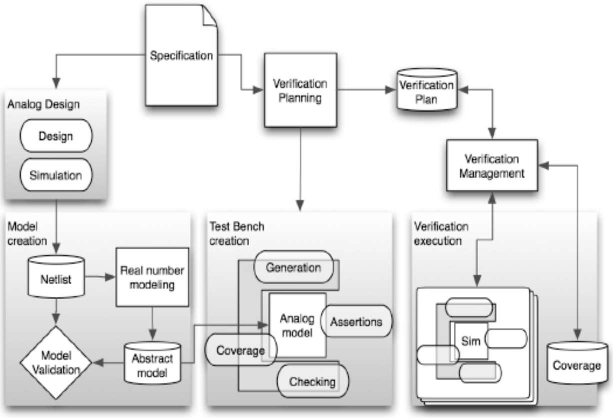


*Figure 2: Verification Flow at the IP Level*

At the SoC level, the UVM-MS test plan is pulled into the SoC verification plan as a chapter of the verification plan. The verified abstract model of the IP is integrated in the SoC verification environment, enabling meaningful verification at reasonable simulation speeds. Components of the IP-level verification environment are reused, for the most part, minimizing the investment for creating the SoC verification environment and boosting confidence in its correctness. The reuse of verification artifacts during integration is depicted in Figure 3

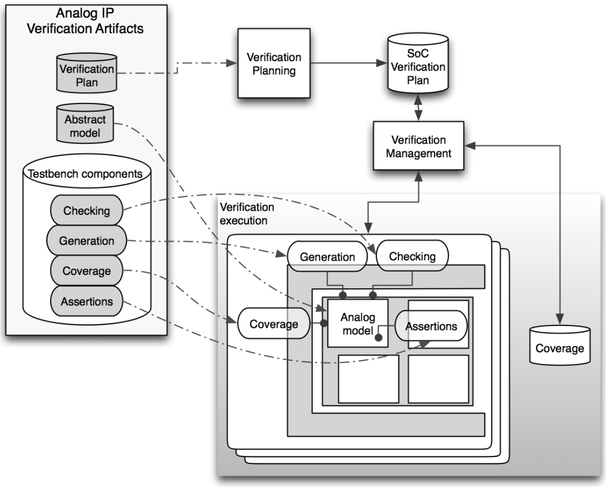


*Figure 3: Verification at the SoC Level Reusing IP Verification Elements*

The resulting SoC verification environment is highly capable, performing meaningful and thorough verification of analog functions and digital/analog interaction. This is achieved in spite of the knowledge gap that exists between the analog IP designer and the SoC integrator, thanks to the knowledge captured in the IP-level verification artifacts.

# 2. Applying MDV to Analog

## 2.1 Guiding Principles of MDV for Analog

Metric Driven Verification is a broadly used concept for verifying large digital designs. Modern designs have huge state spaces hence it is impossible to simulate all their possible conditions, or even a small fraction of those. MDV is guided by the functional specification, rather than the 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 over-all balanced coverage.

Some popular verification methodologies are based on the MDV concept (OVM [4], and more recently UVM [6]). These methodologies teach a specific style that is well supported by tools and libraries. Experience with those 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 mix-signal designs that exhibit similar growth.

## 2.2 Adapting MDV to Analog Design

Conceptually, applying MDV to analog is straight forward [8]: 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 DC bias among other possibilities.

**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. In addition, simulations must be high performance enough to allow for a large number of simulations to be executed. This brings about the following needs.

**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) enables the large volume of simulations required.

**Constrained random stimulus** – input stimulus need 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 vain **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 gain, power consumption, frequency response 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 mix-signal verification methodology to be acceptable, all of the challenges above must be addressed in a cost-effective manner using the newly developed methodology UVM-MS *Universal Verification Methodology – Mixed-Signal* [7]. The rest of the paper is based on applying UVM-MS to the verification of a real-life design: a Noise Cancelling Receiver (NCR).

## 3.3 Planning for Analog Verification

Metric-driven verification relies on a verification plan to be used as a basis. The plan lists all the features that need to be verified, what to check for and how to measure coverage. A typical plan describes test scenarios that would exercise each feature and important feature combinations.

Verification planning was done using the Cadence Enterprise Planner (EP) tool. Using EP, it is possible to maintain a relation between an annotated spec, a section in the plan and implementation code in the testbench. Using EP flow it was possible to work with partial specs, which is very useful in a world of ever changing requirements and revisions of code. EP provided an automated mechanism for maintaining the three representations in sync.

### 3.3.1 Including Analog Properties

Verifying analog features often require measuring continuous values, such as voltage or current at a certain node, continuous (real) values can be sampled, but in order for them to make sense as coverage they need to be quantized into bins. For example, a supply voltage may be classified as nominal, low, high, or off—creating a four element coverage vector. More complex continuous properties, such as gain and signal-to-noise ratio can be computed based on several direct measurements, for example the signal amplitude at various locations in the data path. Such computed quantities need to be similarly quantized when captured as coverage.

Deciding what quantities to measure, either directly or indirectly, and how to quantize them needs to be part of the verification plan. When considering analog circuits, the plan should include all properties of concern, including those that are not directly measured by a functional simulation (or transient analysis). For example, the circuit's frequency response may be a property that requires verification. Rough estimates for some such properties can be computed based on a large number of functional simulations. Others may require SPICE-level analysis that could be automated as part of a regression system.

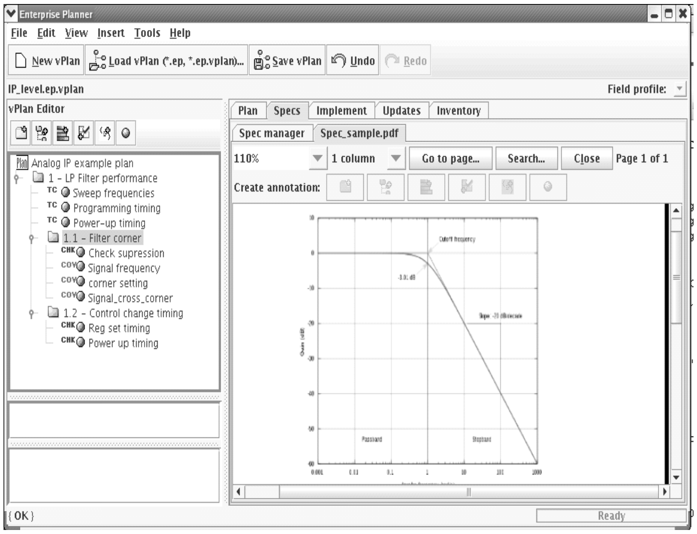
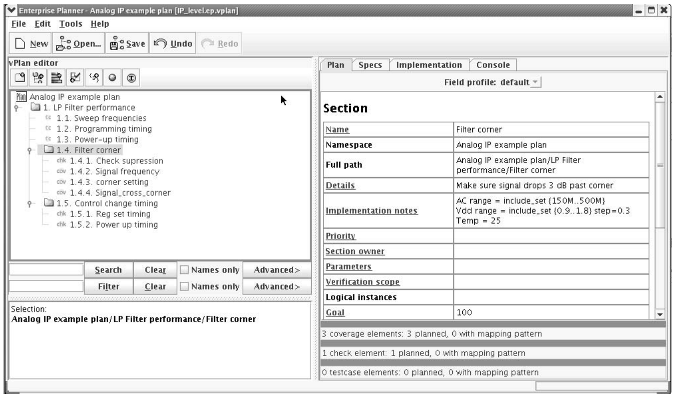


*Figure 4: Enterprise Planner Used to Capture Analog Property*

Note above: Analog property captured: attenuation of signal past corner frequency. A programmable filters coverage space is the cross of the programming values and the frequency range.

Special care should be taken to capture interactions between analog functions and digital control. This includes control registers set during configuration and calibration, various operation modes, switching power modes, and so on. In *Figure 4* above, the low-pass filter is programmable. Verification obligations include verifying the frequency response (best done using AC analysis, but can also be checked using transient analysis with a number of frequencies). Another aspect is the delay between the control register setting and the filter response. This is harder to check, as one must not assume the response time is independent of the control value–that may be implementation dependent. Hence a test scenario including random control settings at random intervals, along with frequency variations is required. Collecting coverage of the parameters and checking the delay time meets the spec ensures correct functionality.

The contributors to an analog test plan should include the architect and analog design team. Information about corners and sweep settings (values included and excluded from sweeps) should be captured and documented in the plan even though they are not directly applicable in a digital setting. Capturing such information in the plan ensures that projects reusing the IP for integration and revisions have access to this information.


Enterprise Planner - Analog IP example plan [IP_level.ep.vplan]
File Edit Tools Help
New
Open...
Save
Undo
Redo
vPlan editor
Plan Specs Implementation Console
Field profile: default
Analog IP example plan
1. LP Filter performance
1.1. Sweep frequencies
1.2. Programming timing
1.3. Power-up timing
1.4. Filter corner
chk 1.4.1. Check supression
cov 1.4.2. Signal frequency
cov 1.4.3. corner setting
cov 1.4.4. Signal_cross_corner
1.5. Control change timing
chk 1.5.1. Reg set timing
chk 1.5.2. Power up timing
Search
Clear
☐ Names only
Advanced>
Filter
Clear
☐ Names only
Advanced>

Selection:
**Analog IP example plan/LP Filter performance/Filter corner**

**Section**

| | |
|---|---|
| Name | Filter corner |
| **Namespace** | Analog IP example plan |
| **Full path** | Analog IP example plan/LP Filter performance/Filter corner |
| Details | Make sure signal drops 3 dB past corner |
| Implementation notes | AC range = include_set {150M..500M}<br>Vdd range = include_set {0.9..1.8} step=0.3<br>Temp = 25 |
| Priority | |
| Section owner | |
| Parameters | |
| Verification scope | |
| **Logical instances** | |
| Goal | 100 |

3 coverage elements: 3 planned, 0 with mapping pattern

1 check element: 1 planned, 0 with mapping pattern

0 testcase elements: 0 planned, 0 with mapping pattern

*Figure 5: vPlan Implementation including Analog*

### 3.3.2 Verification Plan Structured for Reuse

The verification plan is used to guide the verification effort at the IP and SoC levels. It needs to be as complete as possible and has to be revised each time the spec, requirements, or implementation change. To facilitate that, the plan should be structured so that the mapping to specification sections is natural. The plan should be detailed enough such that each plan item (check, coverage, or scenario) maps directly to a single piece of code in the test environment. This often requires a revision once the testbench is implemented. An easy way to achieve that is reading into EP a coverage file and mapping each coverage point, check and test case to a section in the plan.

Different integration targets will probably use the IP in different ways, often disabling features or locking them into a particular mode. Key values that are likely to change between projects could be defined as parameters, using the parameterized plan capability. A well structured plan will help identify which coverage points, checkers, and scenarios can be excluded and how to account for that when computing the overall coverage. EP provides convenient features for excluding sections from the overall coverage score as well as mapping logical scopes to physical scopes.

# 4.0 Constructing a UVM-MS Verification Environment

Creating a verification environment is a programming and modeling task carried out by a verification engineer, with reference to the verification plan. The environment needs to be capable of driving the circuit in all modes, conditions, and scenarios specified in the plan, check all the properties specified, and measure the requested coverage. The investment in a sophisticated verification environment is mitigated in two ways: leveraging pre-exiting universal verification components (UVCs), and re-using components of the verification environment during SoC integration and possibly project-to-project.

For a very detailed discussion on this topic, refer to [7]. A brief description of the main components and careabouts for creating an UVM-MS based verification environment for the NCR example is discussed in section 5.

# 5. Application OF UVM-MS to Real-Life Design:

## Noise Cancelling Receiver (NCR) - This is a made-up specification for the purpose of illustration only

## 5.1 Block Architecture

The Noise Cancelling Receiver (NCR) is part of a challenge-response system illustrated in Figure 6. The system receives transmissions through a directional antenna. The received signal is amplified and demodulated, and is expected to carry a sequence of codes. The codes are matched up with a code-table inside the decoder, and if authenticated, the responder is invoked to transmit a response.

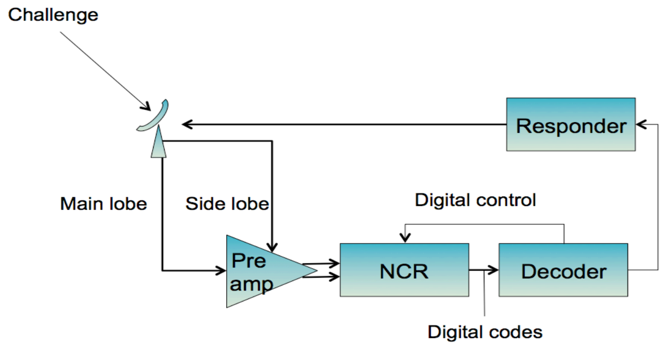


*Figure 6: Receiver System*

The system ignores out-of-lobe transmissions and off-axis jamming noise. Side lobe signal is passed on and subtracted from the main lobe for that purpose. The system features a test mode in which side lobe input is ignored.

The main NCR components are depicted in Figure 7:

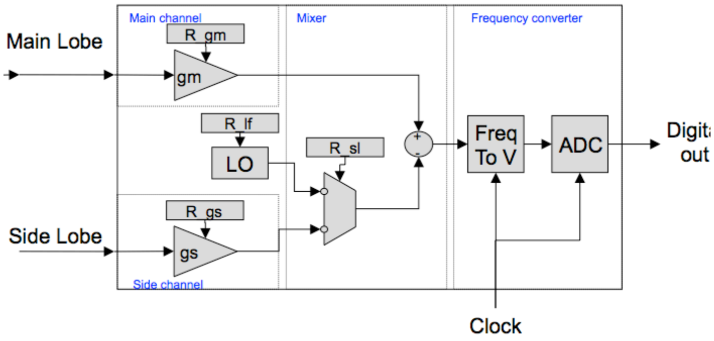


*Figure 7: NCR block architecture*

The NCR block includes the following components:
**Main channel section** – a digitally controlled amplifier for the main channel
**Side channel section** – a digitally controlled amplifier for the side channel
**Mixer section** – mixes the main channel with either the side channel or a local oscillator
**Frequency converter section** – transforms the modulation frequency into digital symbols at the block output

## 5.2 Operation

The NCR block's input signal ranges between 0.5-1.5 GHz. At the block input, the signal is about 2Vp-p. Each input channel contains a digitally controlled amplifier with a gain of 0.6-2.85 (-4.5dB to 9dB). Gain accuracy needs to be within 5% of the programmed gain.

### 5.2.1 Modulation detection

The signal may be amplitude modulated by frequencies in the 300-400KHz range that represent digital code symbols. The modulation amplitude is about 0.2Vp-p. Table 1 lists the codes used:

| Code Name | Frequency (KHz) | NCR Output (6 bits) |
|---|---|---|
| SYM_TOP | < 270.0 | [0x3E-0x3F] |
| SYM_A | 300.0 | [0x33-0x35] |
| SYM_B | 320.0 | [0x30-0x32] |
| SYM_C | 340.0 | [0x2D-0x2F] |
| SYM_D | 360.0 | [0x2B-0x2C] |
| SYM_E | 380.0 | [0x29-0x2A] |
| SYM_F | 400.0 | [0x26-0x28] |

*Table 1: Symbol encoding*

Code symbols are separated by SYM_TOP. For example to decode the sequence {A, B, A} the transmission should carry the following sequence: {SYM_TOP, SYM_A, SYM_TOP, SYM_B, SYM_TOP, SYM_A}. The NCR block must correctly identify code symbols within a maximum of 6 uSec from the time the modulation is present at the block input. Code modulation can be of any duration, but no shorter than 9 uSec. A SYM_TOP must be detected within 4 uSec, and it may be as short as 2.5 uSec.

Symbols may be missed (failed to be detected) at most 1% of the time. Symbols must not be falsely detected (indicated at the output when not present).

### 5.2.2 Normal and Test mode operation

In *Normal* mode, the mixer selector passes on the side channel signal to be mixed with the main channel. In *Test* mode, the selector passes on the local oscillator (LO) signal and the side channel is ignored.

The LO is controlled by the R_lf register. The mode select is done by the R_sl register.

When the LO is in use, it receives digital phase feedback from the mixer circuit, allowing the LO to scan and lock to the main channel. During *Normal* operation, the LO should not be running.

### 5.2.3 Control Registers

The following registers are controlling the NCR operation:

| Register | Bit-size | RTL name | Initial Value | |
|---|---|---|---|---|
| R_gm | 3 | reg_gmain | 0b000 | |
| R_gs | 3 | reg_gside | 0b000 | |
| R_lf | 4 | reg_losc | 0b1000 | |
| R_sl | 1 | reg_side_sel | 0b0 | |

*Table 2: Control registers*

### 5.2.3.1 Gain control registers

The two input channels are symmetrical. Each channel gain is register controlled. The gain values are specified in Table 3 below.

| R_gm, R_gs (3 bits) | Gain | Gain (dB) |
|---|---|---|
| 0b000 | 0.0 | -∞ |
| 0b001 | 0.65 | -3.75 |
| 0b010 | 0.92 | -0.73 |
| 0b011 | 1.12 | 0.98 |
| 0b100 | 1.95 | 5.8 |
| 0b101 | 2.25 | 7.0 |
| 0b110 | 2.50 | 8.0 |
| 0b111 | 2.85 | 9.0 |

*Table 3: Gain control registers*

### 5.4.3.2 LO Control register

The LO control register R_lf controls the output frequency of the LO. The register value is taken as a 4 bit signed integer. The LO frequency is computed as follows:

$$f_{LO} = \frac{10^7\left(100 + \frac{R_{lf}}{0b1000}\right)}{2}$$

### 5.2.4.3 Mixer select

The mixer select register R_sl is set to *Normal* (0b0) by default. When set to 0b1, *Test* mode is active.

## 5.3 Creation of an Executable Verification Plan (vPlan)

Metric driven verification relies on a verification plan to be used as a basis. The plan lists all the features that need to be verified, **what** to check for and **how** to measure coverage. A typical plan describes test scenarios that would exercise each feature and important feature combinations as described in Section 3.3.1

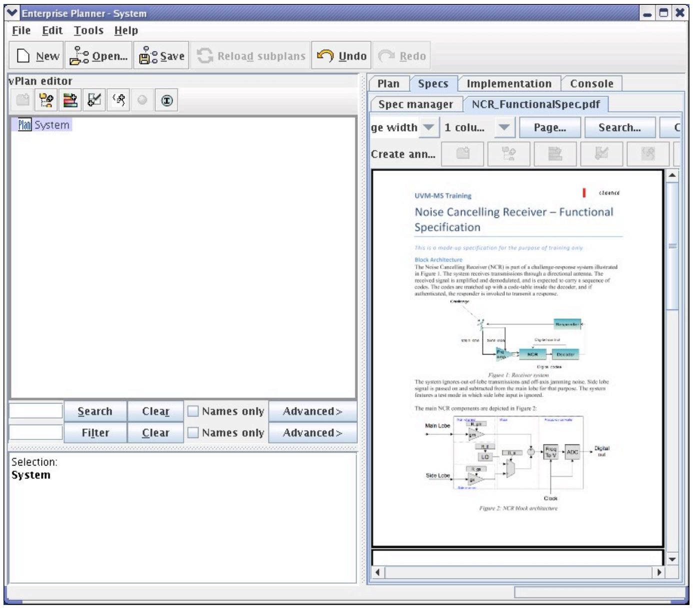
### 5.3.1 Capturing Analog properties in vPlan

Verifying analog features often require measuring continuous values, such as voltage or current at a certain node. Continuous (real) values can be sampled, but in order for them to make sense as coverage they need to be quantized into bins.

When considering analog circuits, the plan should include all properties of concern, including those that are not directly measured by a functional simulation (or transient analysis). Special care should be taken to capture interactions between analog functions and digital control.

Let's explore the NCR spec to identify features of interest and create a vPlan for the NCR. EPlanner tool was used to create the executable verification plan.

The first step is to import the spec of the NCR device (in pdf format) into the plan creation tool – Figure 8. After a planning session with the key stakeholders in the project (designers, verification engineers, architects, s/w developers), one is ready to start adding sections and subsections to the vPlan – see examples in Figure 9.



UVM-MS Training

cadence

Noise Cancelling Receiver – Functional Specification

*This is a made-up specification for the purpose of training only*

Block Architecture

The Noise Cancelling Receiver (NCR) is part of a challenge-response system illustrated in Figure 1. The system receives transmissions through a directional antenna. The received signal is amplified and demodulated, and is expected to carry a sequence of codes. The codes are matched up with a code-table inside the decoder, and if authenticated, the responder is invoked to transmit a response.

*Figure 1: Receiver system*

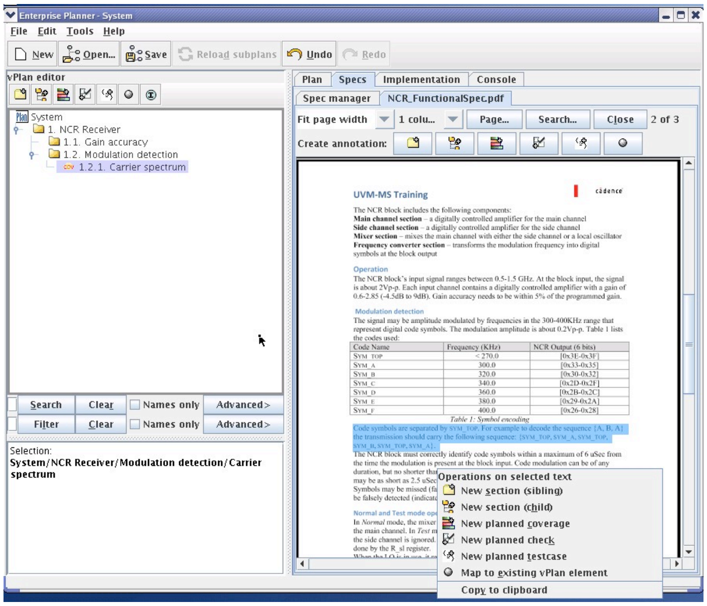
The system ignores out-of-lobe transmissions and off-axis jamming noise. Side lobe signal is passed on and subtracted from the main lobe for that purpose. The system features a test mode in which side lobe input is ignored.

The main NCR components are depicted in Figure 2:

*Figure 2: NCR block architecture*

*Figure 8: Importing spec into tool for vPlan creation*

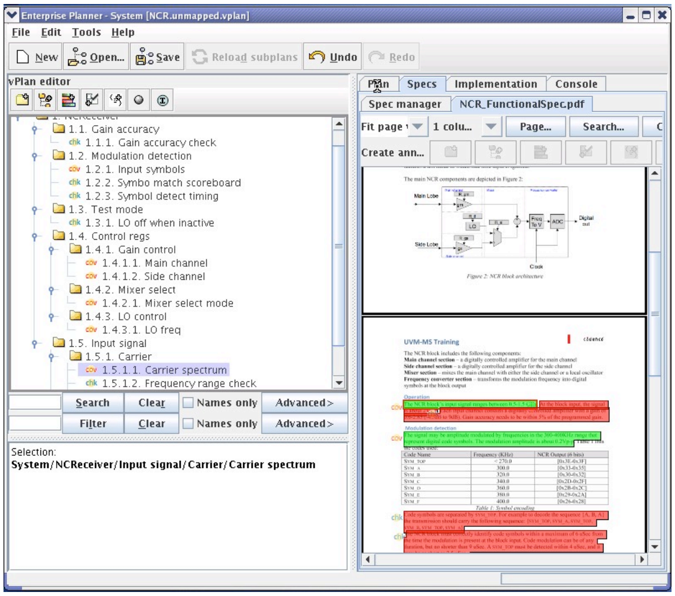
Planned sections for coverage, checkers, assertions and specific tests etc. are added to the vPlan.


Enterprise Planner - System
File Edit Tools Help
New Open... Save Reload subplans Undo Redo
vPlan editor
System
1. NCR Receiver
1.1. Gain accuracy
1.2. Modulation detection
1.2.1. Carrier spectrum
Search Clear Names only Advanced>
Filter Clear Names only Advanced>
Plan Specs Implementation Console
Spec manager NCR_FunctionalSpec.pdf
Fit page width 1 colu... Page... Search... Close 2 of 3
Create annotation:
UVM-MS Training
cādence
Operations on selected text
New section (sibling)
New section (child)
New planned coverage
New planned check
New planned testcase
Map to existing vPlan element
Copy to clipboard

Selection:
System/NCR Receiver/Modulation detection/Carrier spectrum

Figure 9: Adding Planned Checks and Coverage to vPlan

At the end of the planning session, a complete vPlan is available with verification features of interest extracted and **planned coverage**, checkers, and tests well understood and documented in an executable spec – Figure 10.


Enterprise Planner - System [NCR.unmapped.vplan]
File Edit Tools Help
New Open... Save Reload subplans Undo Redo
vPlan editor
1.1. Gain accuracy
1.1.1. Gain accuracy check
1.2. Modulation detection
1.2.1. Input symbols
1.2.2. Symbo match scoreboard
1.2.3. Symbol detect timing
1.3. Test mode
1.3.1. LO off when inactive
1.4. Control regs
1.4.1. Gain control
1.4.1.1. Main channel
1.4.1.2. Side channel
1.4.2. Mixer select
1.4.2.1. Mixer select mode
1.4.3. LO control
1.4.3.1. LO freq
1.5. Input signal
1.5.1. Carrier
1.5.1.1. Carrier spectrum
1.5.1.2. Frequency range check
Search Clear Names only Advanced>
Filter Clear Names only Advanced>
Plan Specs Implementation Console
Spec manager NCR_FunctionalSpec.pdf
Fit page 1 colu... Page... Search...
Create ann...
UVM-MS Training
cādence

Selection:
System/NCReceiver/Input signal/Carrier/Carrier spectrum

Figure 10: Annotated vPlan with features extracted from specs

## 5.4 Analog Model Creation and Validation

Analog models are often developed at a concrete netlist level, using Spectre (or some other variant of Spice) to simulate. These models are highly accurate, but they are very slow to simulate. This is a major drawback when considering the large number of combinations that have to be covered to verify functionality.

Behavioral analog models, like Verilog-AMS offer a much faster runtime, while compromising accuracy to some small degree. Much analog analysis can still be performed on AMS models, as they are simulated using an analog solver.

Even better performance can be achieved using Real Number Models, for example Verilog models using *reals* and *wreals* (we call these *wreal* models here). Such models are even more abstract, as the user chooses explicitly which electrical phenomenons to model. Such a modeling style can be executed on an event driven (digital) simulator, which is orders of magnitude faster than using an analog solver. Such models tend to be less accurate, and they can only be used for functional simulation (transient analysis).

In many cases a wreal model is the best choice for implementing UVM-MS. Model accuracy is typically not an issue for the kind of checking done during functional simulations and the speed advantage is very significant. Creating such a model may require a considerable investment, but that investment will be leveraged for IP verification and possibly multiple SoC integrations (where speed is really critical). Furthermore, such models can be reused with small modifications from one generation of a design to the next, thus promoting reuse.

The UVM-MS methodology applies to all modeling styles, enabling easy switching of models within the same test environment. This is especially important for debug purposes, when an error detected during a fast model simulation needs to be re-played on a more accurate model.

Each time more than one representation of a system exists, the question of congruence between models is raised. In the case of analog design, the netlist model is likely to evolve over time, as the design work progresses. The more abstract models (AMS, wreal) need to catch up. Because the bulk of verification is using the abstract models, the question of congruence becomes critical. Model validation deals with verifying that an abstract model matches the concrete model.

Because the models to be compared are not at the same level of abstraction, one has to define the exact nature of comparison to be done. The following criteria are commonly used to determine matching:

- Waveforms of select signals to be compared. Some tolerance in terms of timing and value may be defined to allow some flexibility in modeling.
- Expressions or equations that are evaluated continuously or at the end of the run.
- Test bench checkers that should not trigger for either model execution.
- Interface ports that need to match in name, type and direction.

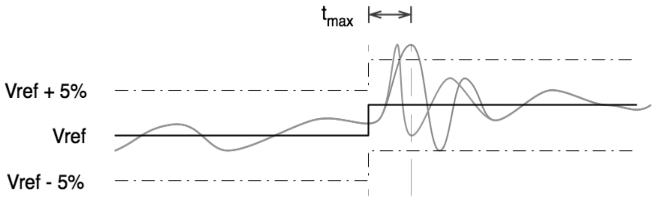
Models keep changing throughout the design and verification cycle, and hence it is important to validate the models repeatedly, as part of a regression process. To enable that, the validation needs to become an automated, push button task. This can be done by applying the same stimulus to the two models executing in their respective test benches and comparing the outputs.

## 5.5 Signal Generator and Monitor

Signals are generated by a programmable signal generator and monitor, which is used to drive and measure analog signals in a controlled manner, also known as *dms_wire* UVC. Typical functions include:

- Driving a signal, such as a sine wave at a specified frequency, amplitude bias (DC level), and phase.
- Measuring the envelope of a signal, which is assumed to be periodic, within a time window. The time window is defined by a triggering event or a sequence item.
- Driving and monitoring configurations, which are controlled by dedicated sequence items and supporting easy integration into multi-channel test sequences. Controls can also be set by way of constraints for pre-run configurations.
- The monitor features built-in coverage, customizable by extension, as well as a method port interface for custom coverage and checking.
- Time resolution is explicitly controlled (the number of samples per cycle).
- Optionally, a continuous time signal source can be used for driving netlist models accurately. It is possible to combine several signal generators to achieve complex input signals. Modulation and noise injection are common cases.

Other monitors are needed, for example for coverage checking of a-periodic analog signals often requires threshold crossing detection. Simpler cases use hard constants, for instance checking that a signal voltage is always below 0.9 Volts. A more complex case requires comparison between a signal and a reference. Because of model and circuit inaccuracies, the checking needs to allow some tolerance, both in terms of the measured quantity and the response time. Consider for example the requirement: "*the regulator will maintain the output voltage at 95% +/-5% of the reference voltage, with a response time of less than tmax*".



Checking the relation between reference and signals, one signal is within voltage and time tolerances. The other signal takes too long to settle after the step and shoots over the high margin.

The *dms_threshold* block implements a programmable detector that is robust in face of short transients (spikes, glitches). It features:

- Transient suppression mechanism eliminating crossings (high-low-high and low-high-low) shorter than a set suppression time interval.
- Programmable high and low threshold and suppression time interval, set by way of constraints for pre-run configurations.
- The (nominal) threshold value is connected to a reference voltage signal.
- Event port outputs indicating definite threshold crossing (high-to-low, low-to-high and their union). Unfiltered high-crossed and low-crossed events can also be interfaced. This is for custom checking and coverage collection.
- Automatic coverage of crossings (low-to-high, high-to-low, both).

## 5.6 Analog Configuration Interface

Analog circuits often feature banks of control registers used to trim, program, or calibrate the circuit. These registers should be divided into distinct control interfaces according to their function, keeping in mind that any particular integration of the IP may treat such interfaces differently. For example, in one case an interface may be tied to a digital block driving it, while in a simpler instantiation that interface may be tied to constant values.

Creating a simple *dms_reg* UVC can provide the following functions:

- A macro construct for defining the interface, its control clock, and the set of registers included.
- Each register is specified with its size in bits and its reset value
- A sequence item interface provides easy integration of interface setup commands (reset, write) into multi-channel test sequences.
- Automatic coverage definition and collection upon value change.
- Value change notification through method ports, facilitating custom checking and coverage collection.

## 5.7 Checking analog functionality

The UVM-MS approach to checking is possibly its biggest departure from current practices in analog design. When using metric driven verification there is simply no escape from automatic checking, while manual inspection of waveforms is still a much used method for checking analog correctness. The challenge of automating analog checking is **not to replace manual inspection** completely, but rather to provide a reasonably good detection of errors for the multitude of batch runs. Deciding which functions and features to check for is a major decision during the verification planning phase.

### 5.7.1 Triggering a check on control changes

Carefully timing a check in a specific test may be good enough for a feature test, but having the check trigger automatically is much more powerful. Automatic activation of tests ensures the checked condition is monitored continuously, even when that feature is not targeted. This is especially important when the checker is integrated in a bigger environment where tight control over input timing may not be possible. In order to determine the activation timing of a check, one has to consider which controls effect the checked condition. These controls are likely to be register settings and external interfaces. Each time any of these controls change, the check should be triggered. The following timing diagram illustrates this concept.

Let us now look at some potential checks that can be implemented in the NCR Verification Environment (VE).

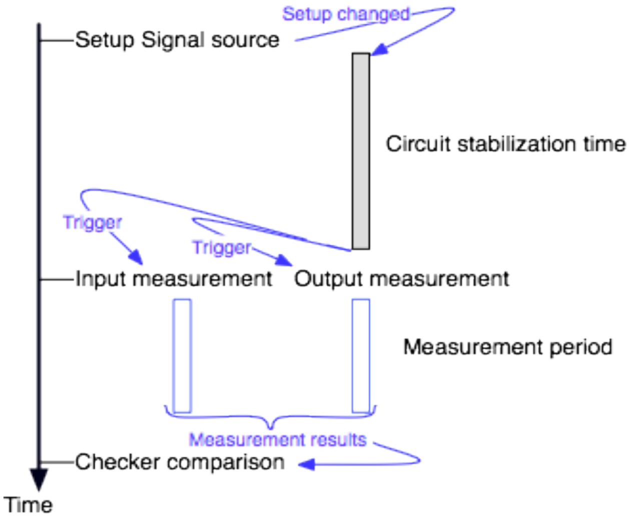


*Figure 11: Event triggers for automated checks in verification env.*

#### 5.7.1.1 Checking - Stage 1: Gain Check

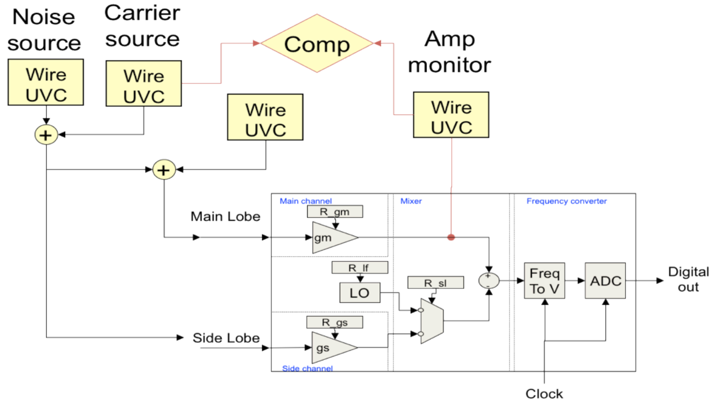


*Figure 12: NCR Stage 1 Gain Check*

Timing the check requires figuring out a triggering event to start the measurement on both signals. In case there is an expected delay between input and output, the measurements need to be delayed accordingly. The measurement happens over a period of time, at the end of which each monitor provides its output, typically as a call to a method port. The final checker code is triggered by the returned measurements.

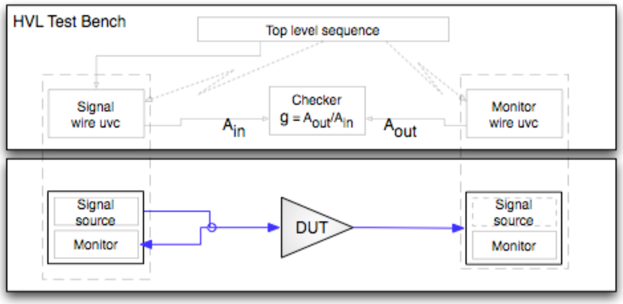


*Figure 13: Triggering event to start measurement*

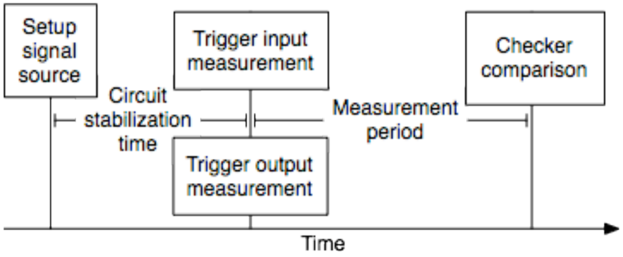

Setup signal source
Trigger input measurement
Checker comparison
Circuit stabilization time
Measurement period
Trigger output measurement
Time

*Figure 14: Setup for automated tests & checks*

Looking at the NCR Spec, the gain-control register sets the gain of the pre-Amp. From Table 3, the two input channels are symmetrical. Each channel gain is register controlled. Using this checkers can be created and added to the verification env (VE).

### 5.7.2 Assertions based Checks

Assertions are a complementary approach to checking. Assertions tend to be local checks that are embedded or associated with the design IP. Assertions are seamlessly integrated into the coverage and reporting scheme of the test bench. Assertions are most often used to check **input conditions** and local **invariants.**

#### 5.7.2.1 Checking input conditions

A common problem during SoC integration is hooking up the IPs incorrectly. This may be as crude as switching the polarity of a signal, or a subtle mismatch in voltage levels. However, once such an error occurs it is very hard to detect and debug in the context of the integrated system. Input checking is all about detecting such problems. They are typically implemented as assertions so that they are included in the integrated system.

Assertion languages like PSL and SVA are synchronous, meaning they are associated with a clock event. Still, to check a condition holds at the end of reset, the falling edge of reset can be used as the clock event. The AMS extension to PSL supports checking electrical properties of signals, like the voltage level of a node. Hence verifying the input voltage on a node directly after reset is trivially simple, as shown below.

```
input_v_check: assert always
  ((V(sig_in)>0.5m) &&(V(sig_in)<10m)
  ) @(negedge reset);
```

A more complicated case arises when the property to be checked is dependent on the configuration. Assume for example, an input is allowed to be 0 if a particular feature is disabled, but should meet the specified voltage range otherwise. This can be expressed using an implication (an overlapping suffix implication in PSL). This topic is covered in much more detail in [7].

```
input_cond_check: assert always
  (feature_enabled |->((V(sig_in)>0.5m) && (V(sig_in)<10m))
  ) @(negedge reset);
```

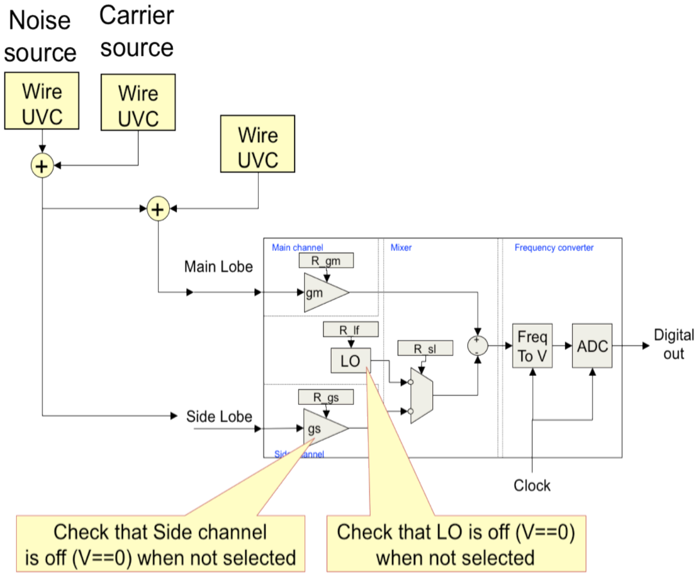
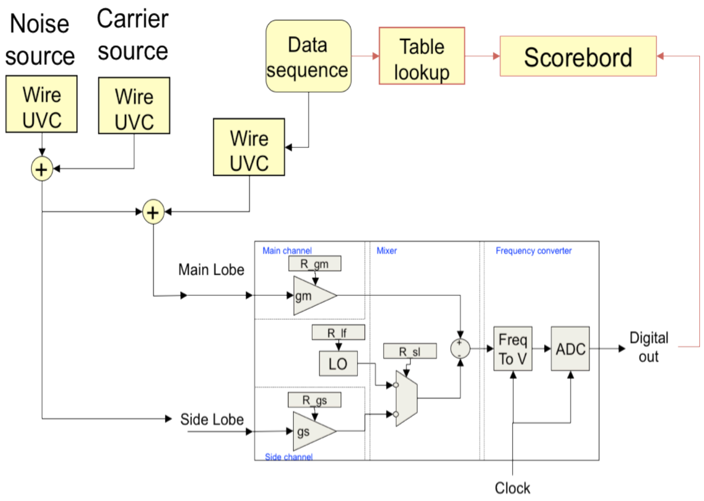


*Figure 15: NCR Stage 2 Gain Check*

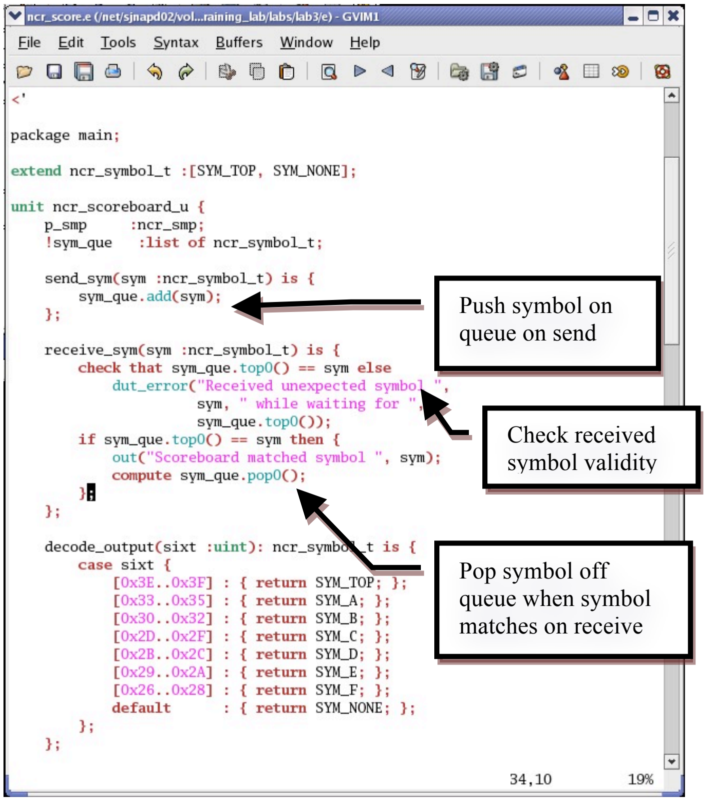
An example of assertions used for checks above in Figure 15.

```
Check1_Stage2: assert (always
  ({reg_side_sel == 1} |=> {V(side_out) == 0.0}
  ) @(posedge ctrl_write_clk) );
```

### 5.7.2.2 Checking - Stage 3: end-to-end Checks using Scoreboards



End-to-end checkers are implemented by using UVM components described in Section #4. The code below shows examples of how this is achieved.

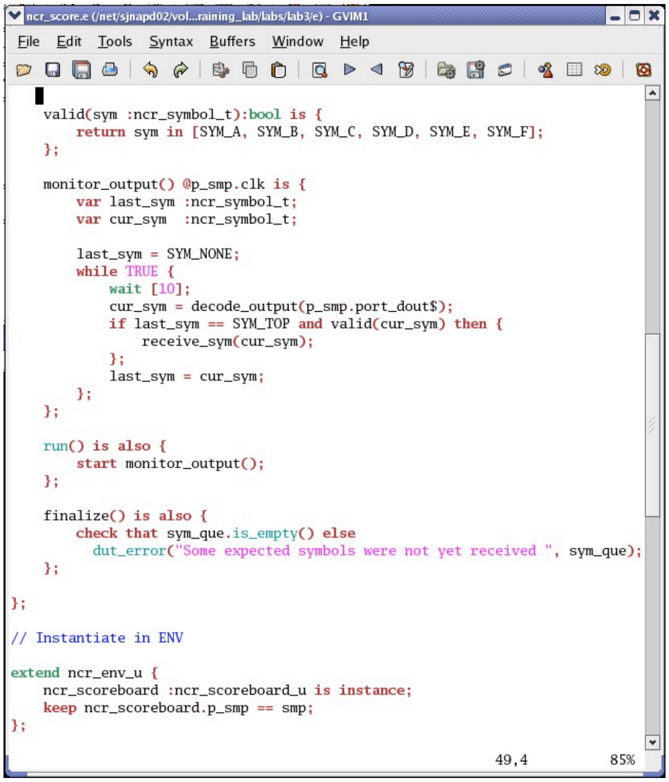

ncr_score.e (/net/sjnapd02/vol...raining_lab/labs/lab3/e) - GVIM1
File Edit Tools Syntax Buffers Window Help
34,10 19%

```
<'

package main;

extend ncr_symbol_t :[SYM_TOP, SYM_NONE];

unit ncr_scoreboard_u {
    p_smp      :ncr_smp;
    !sym_que    :list of ncr_symbol_t;

    send_sym(sym :ncr_symbol_t) is {
        sym_que.add(sym);
    };

    receive_sym(sym :ncr_symbol_t) is {
        check that sym_que.top0() == sym else
            dut_error("Received unexpected symbol ",
                      sym, " while waiting for ",
                      sym_que.top0());
        if sym_que.top0() == sym then {
            out("Scoreboard matched symbol ", sym);
            compute sym_que.pop0();
        };
    };

    decode_output(sixt :uint): ncr_symbol_t is {
        case sixt {
            [0x3E..0x3F] : { return SYM_TOP; };
            [0x33..0x35] : { return SYM_A; };
            [0x30..0x32] : { return SYM_B; };
            [0x2D..0x2F] : { return SYM_C; };
            [0x2B..0x2C] : { return SYM_D; };
            [0x29..0x2A] : { return SYM_E; };
            [0x26..0x28] : { return SYM_F; };
            default      : { return SYM_NONE; };
        };
    };
```

Push symbol on queue on send

Check received symbol validity

Pop symbol off queue when symbol matches on receive


ncr_score.e (/net/sjnapd02/vol...raining_lab/labs/lab3/e) - GVIM1
File Edit Tools Syntax Buffers Window Help
49,4 85%

```
    valid(sym :ncr_symbol_t):bool is {
        return sym in [SYM_A, SYM_B, SYM_C, SYM_D, SYM_E, SYM_F];
    };

    monitor_output() @p_smp.clk is {
        var last_sym :ncr_symbol_t;
        var cur_sym  :ncr_symbol_t;

        last_sym = SYM_NONE;
        while TRUE {
            wait [10];
            cur_sym = decode_output(p_smp.port_dout$);
            if last_sym == SYM_TOP and valid(cur_sym) then {
                receive_sym(cur_sym);
            };
            last_sym = cur_sym;
        };
    };

    run() is also {
        start monitor_output();
    };

    finalize() is also {
        check that sym_que.is_empty() else
          dut_error("Some expected symbols were not yet received ", sym_que);
    };

};

// Instantiate in ENV

extend ncr_env_u {
    ncr_scoreboard :ncr_scoreboard_u is instance;
    keep ncr_scoreboard.p_smp == smp;
};
```

*Figure 16: Scoreboard Implementation*

## 5.8 Coverage

### 5.8.1 Analog Coverage

In the context of Analog, Coverage collection involves capturing data at specified points in time. For analog coverage, deciding on **what** data to collect, **how** to categorize (continuous real value) data into bins and deciding on **when** to sample are the tasks at hand.

### 5.8.2 Direct and computed coverage collection

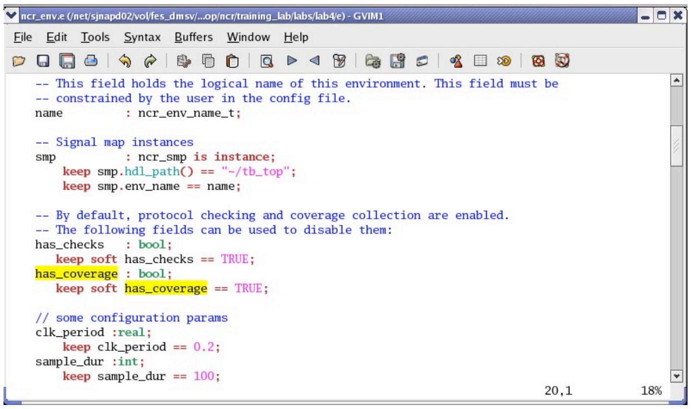
Covered values can be voltage or current measured directly off circuit nodes. Other values may be computed by the test bench before sampling. Direct measurements are the simplest to implement - the covered node is sampled directly by the high-level testbench.

Values that cannot be measured directly may be computed at the HDL testbench level for one of the following reasons:

- The quantity measured for coverage is not directly available. For instance, to compute the power of a node, one could create a real wire holding the multiplication of voltage and current. The result wire is then sampled for coverage.
- The source of the measured quantity differs depending on the DUT model being used. A value of interest may be available as an internal node in one model, a different node in another model, and a derived value in a third. In this case, replicating the value at a node in the HDL testbench allows the high-level testbench to deal with the different models uniformly.
- Selective coverage. Coverage collected from within the analog model may be unavailable when running with a netlist model. Such coverage should be defined under a *when* condition, so that it can be turned on and off as needed. This is discussed in more detail in [7].

Coverage collection for rapidly changing signals require an indirect measurement of the desired properties. The *dms_wire* UVC provides capabilities to monitor such signals and compute their amplitude, frequency, phase and DC bias. These values cannot be sampled directly.

For a more complete description, please refer to [7].


ncr_env.e (/net/sjnapd02/vol/fes_dmsv/...op/ncr/training_lab/labs/lab4/e) - GVIM1
File Edit Tools Syntax Buffers Window Help

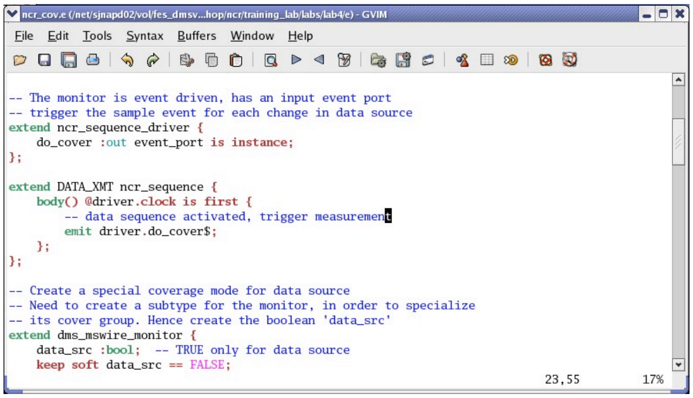
```
    -- This field holds the logical name of this environment. This field must be
    -- constrained by the user in the config file.
    name         : ncr_env_name_t;

    -- Signal map instances
    smp          : ncr_smp is instance;
        keep smp.hdl_path() == "~/tb_top";
        keep smp.env_name == name;

    -- By default, protocol checking and coverage collection are enabled.
    -- The following fields can be used to disable them:
    has_checks   : bool;
       keep soft has_checks == TRUE;
    has_coverage : bool;
       keep soft has_coverage == TRUE;

    // some configuration params
    clk_period :real;
        keep clk_period == 0.2;
    sample_dur :int;
        keep sample_dur == 100;
                                                                20,1          18%
```

Figure 17: Enabling Analog Coverage

## 5.8.3 Methods for Timing the Collection of Coverage

Sampled signals may be highly dynamic, like a high frequency data signal, or slow changing, like a reference voltage. Slow changing data can be sampled using simple triggers. In contrast, naive sampling of dynamic signals will generate a broad spectrum of values depending on the exact sampling time. Consider the attempt to measure the amplitude of a sine wave —it cannot be done accurately using a simple trigger.

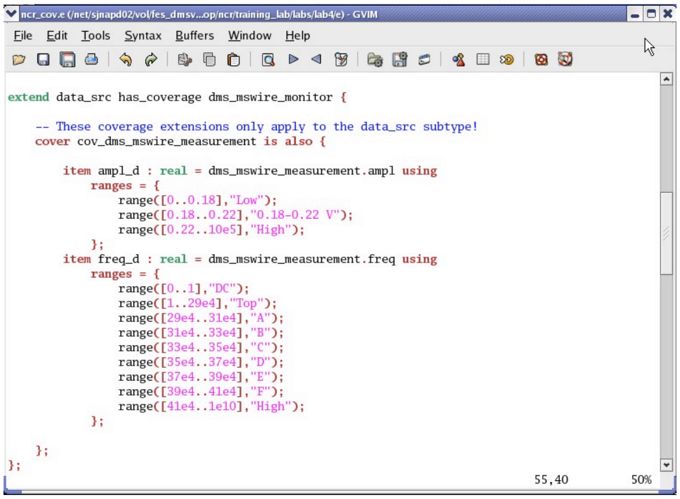
Trigger timing should take into account the event that causes a perturbation to the analog input or control, as well as the time it takes for the data path to respond and stabilize. Consider, for example, a digitally controlled voltage source. The change event is a write operation to the control register. Coverage should be sampled after the appropriate propagation delay, or response time, which is typically part of the circuit specification. The coverage event should be triggered by the write operation, delayed by the nominal response time. Figure 18, shows how coverage collection is triggered.


ncr_cov.e (/net/sjnapd02/vol/fes_dmsv...hop/ncr/training_lab/labs/lab4/e) - GVIM
File Edit Tools Syntax Buffers Window Help

```
-- The monitor is event driven, has an input event port
-- trigger the sample event for each change in data source
extend ncr_sequence_driver {
    do_cover :out event_port is instance;
};

extend DATA_XMT ncr_sequence {
    body() @driver.clock is first {
        -- data sequence activated, trigger measurement
        emit driver.do_cover$;
    };
};

-- Create a special coverage mode for data source
-- Need to create a subtype for the monitor, in order to specialize
-- its cover group. Hence create the boolean 'data_src'
extend dms_mswire_monitor {
    data_src :bool;  -- TRUE only for data source
    keep soft data_src == FALSE;
                                                                23,55          17%
```

Figure 18: Triggering Coverage collection

Coverage collection for rapidly changing signals requires an indirect measurement of the desired properties. The *dms_wire* UVC provides capabilities to monitor such signals and compute their amplitude, frequency, phase, and DC bias. These values cannot be sampled directly as mentioned above. Figure 19 shows the coverage model.

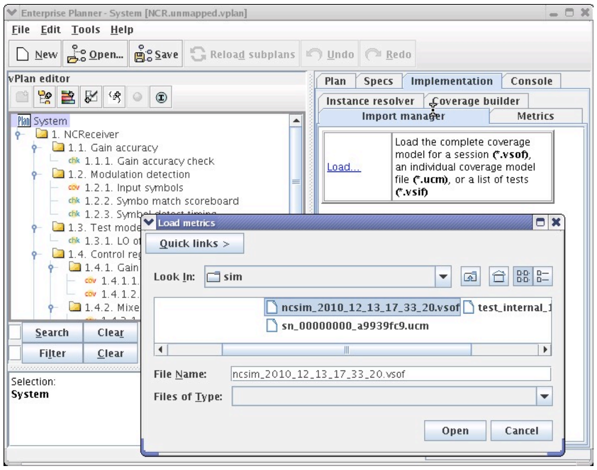

ncr_cov.e (/net/sjnapd02/vol/fes_dmsv...op/ncr/training_lab/labs/lab4/e) - GVIM
File Edit Tools Syntax Buffers Window Help

```
extend data_src has_coverage dms_mswire_monitor {

    -- These coverage extensions only apply to the data_src subtype!
    cover cov_dms_mswire_measurement is also {

        item ampl_d : real = dms_mswire_measurement.ampl using
            ranges = {
                range([0..0.18],"Low");
                range([0.18..0.22],"0.18-0.22 V");
                range([0.22..10e5],"High");
            };
        item freq_d : real = dms_mswire_measurement.freq using
            ranges = {
                range([0..1],"DC");
                range([1..29e4],"Top");
                range([29e4..31e4],"A");
                range([31e4..33e4],"B");
                range([33e4..35e4],"C");
                range([35e4..37e4],"D");
                range([37e4..39e4],"E");
                range([39e4..41e4],"F");
                range([41e4..1e10],"High");
            };

    };
};
                                                                55,40          50%
```

Figure 19: Coverage Model for Analog

Figure 10 shows the planned coverage in the vPlan. The next step is to load in collected coverage as specified by the various sources of coverage in the VE – cover-groups, assertions etc. Figure 19 below shows raw coverage data being loaded into the vPlan.

The link between spec – to planned coverage – to collected coverage is managed and achieved by mapping collected coverage to planned coverage as discussed in the next section.



System
1. NCReceiver
1.1. Gain accuracy
1.1.1. Gain accuracy check
1.2. Modulation detection
1.2.1. Input symbols
1.2.2. Symbo match scoreboard
1.2.3. Symb
1.3. Test mode
1.3.1. LO o
1.4. Control re
1.4.1. Gain
1.4.1.1.
1.4.1.2.
1.4.2. Mixe

Load the complete coverage model for a session **(*.vsof)**, an individual coverage model file **(*.ucm)**, or a list of tests **(*.vsif)**

Selection: **System**

Look In: sim

ncsim_2010_12_13_17_33_20.vsof
test_internal_1
sn_00000000_a9939fc9.ucm

File Name: ncsim_2010_12_13_17_33_20.vsof

Files of Type:

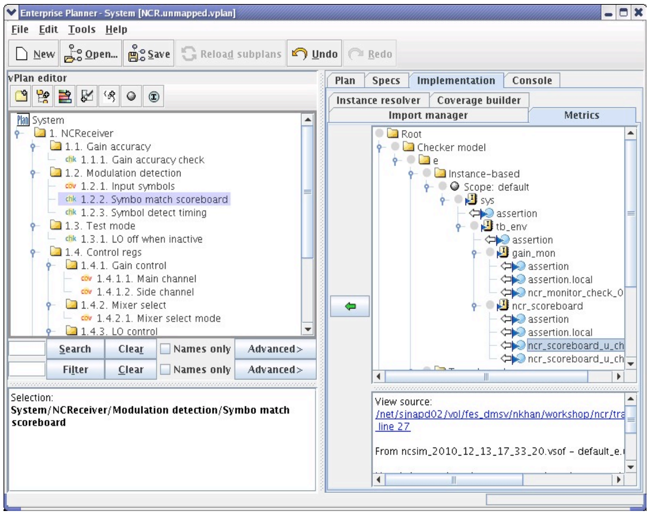
Figure 20: Loading vPlan and raw coverage from simulation run

## 5.8.3 Mapping Raw coverage data into vPlan:

Collected coverage data is mapped onto planned coverage in vPlan

- collected metrics (on right side) to planned metrics (on left side)
- implemented checks and assertions (on right) to planned checks (on the left side)

This is the process by which the collected metrics is backannotated onto the planned metrics.

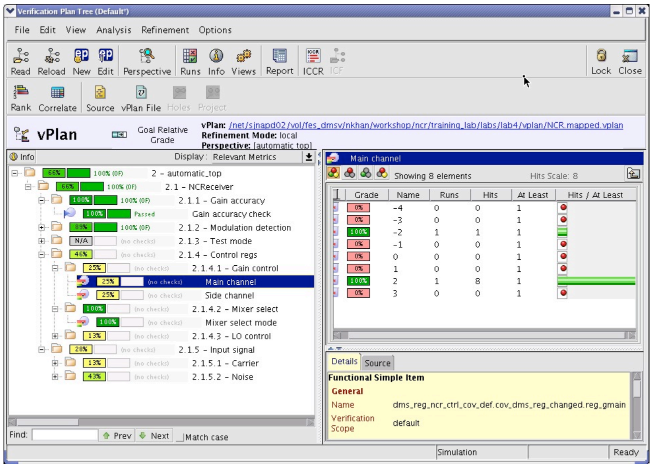

Enterprise Planner - System [NCR.unmapped.vplan]
File Edit Tools Help
New Open... Save Reload subplans Undo Redo
vPlan editor
Plan Specs Implementation Console
Instance resolver Coverage builder
Import manager Metrics
System
1. NCReceiver
1.1. Gain accuracy
chk 1.1.1. Gain accuracy check
1.2. Modulation detection
cov 1.2.1. Input symbols
chk 1.2.2. Symbo match scoreboard
chk 1.2.3. Symbol detect timing
1.3. Test mode
chk 1.3.1. LO off when inactive
1.4. Control regs
1.4.1. Gain control
cov 1.4.1.1. Main channel
cov 1.4.1.2. Side channel
1.4.2. Mixer select
cov 1.4.2.1. Mixer select mode
1.4.3. LO control
Search Clear Names only Advanced>
Filter Clear Names only Advanced>
Root
Checker model
e
Instance-based
Scope: default
sys
assertion
tb_env
assertion
gain_mon
assertion
assertion.local
ncr_monitor_check_0
ncr_scoreboard
assertion
assertion.local
ncr_scoreboard_u_ch
ncr_scoreboard_u_ch

Selection:
**System/NCReceiver/Modulation detection/Symbo match scoreboard**

View source:
/net/sjnapd02/vol/fes_dmsv/nkhan/workshop/ncr/tra
line 27

From ncsim_2010_12_13_17_33_20.vsof - default_e.

*Figure 21: Mapping Collected coverage to Planned coverage in vPlan*


Verification Plan Tree (Default*)
File Edit View Analysis Refinement Options
Read Reload New Edit Perspective Runs Info Views Report ICCR ICF Lock Close
Rank Correlate Source vPlan File Holes Project
vPlan
Goal Relative Grade
Info
Display: Relevant Metrics
Main channel
Showing 8 elements
Hits Scale: 8
Details Source
Find:
Prev Next Match case
Simulation
Ready

**vPlan:** /net/sjnapd02/vol/fes_dmsv/nkhan/workshop/ncr/training_lab/labs/lab4/vplan/NCR.mapped.vplan
**Refinement Mode:** local
**Perspective:** [automatic top]

| | | | |
|---|---|---|---|
| 66% | 100% (0F) | 2 – automatic_top |
| 66% | 100% (0F) | 2.1 – NCReceiver |
| 100% | 100% (0F) | 2.1.1 – Gain accuracy |
| 100% | Passed | Gain accuracy check |
| 89% | 100% (0F) | 2.1.2 – Modulation detection |
| N/A | (no checks) | 2.1.3 – Test mode |
| 46% | (no checks) | 2.1.4 – Control regs |
| 25% | (no checks) | 2.1.4.1 – Gain control |
| 25% | (no checks) | Main channel |
| 25% | (no checks) | Side channel |
| 100% | (no checks) | 2.1.4.2 – Mixer select |
| 100% | (no checks) | Mixer select mode |
| 13% | (no checks) | 2.1.4.3 – LO control |
| 28% | (no checks) | 2.1.5 – Input signal |
| 13% | (no checks) | 2.1.5.1 – Carrier |
| 43% | (no checks) | 2.1.5.2 – Noise |

| Grade | Name | Runs | Hits | At Least | Hits / At Least |
|---|---|---|---|---|---|
| 0% | -4 | 0 | 0 | 1 | ● |
| 0% | -3 | 0 | 0 | 1 | ● |
| 100% | -2 | 1 | 1 | 1 | 🟩 |
| 0% | -1 | 0 | 0 | 1 | ● |
| 0% | 0 | 0 | 0 | 1 | ● |
| 0% | 1 | 0 | 0 | 1 | ● |
| 100% | 2 | 1 | 8 | 1 | 🟩 |
| 0% | 3 | 0 | 0 | 1 | ● |

**Functional Simple Item**
**General**
Name dms_reg_ncr_ctrl_cov_def.cov_dms_reg_changed.reg_gmain
Verification Scope default

*Figure 22: Coverage data from regression mapped back onto vPlan*

We now have a very efficient and complete way to apply UVM based Metrics and Plan driven verification to Mixed Signal designs.

## 6. CONCLUSIONS

The UVM-MS methodology presented here is the first mix-signal verification methodology we are aware of that scales from analog IP blocks to mix-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 as enumerated above 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 costs associated and expertise needed. Based on this experience we would 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 mix-signal design verification. While the authors have made a conscious effort to provide as much detail as possible within the constraints of the size of this paper, there are several other topics that need to be discussed. For those interested in more details, our experiences have been fully captured in a new book [7].

## 7. REFERENCES

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

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

[3] 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*

[4] www.ovmworld.org

[5] www.SystemVerilog.org

[6] www.uvmworld.org

[7] Bishnupriya B., John D., Gary H., Nick H., Yaron K., Neyaz K., Zeev K., Efrat S., 2012, Advanced Verification Topics, *Cadence Design Systems, ISBN 978-1-105-11375-8*

[8] Neyaz Khan, Yaron Kashai, Hao Fang, 2011, DVCon Metrics Driven Verification of Mixed Signal Designs