Verification of Complex Analog and RF IC Designs

Henry Chang
Ken Kundert

Version 1a, February 2007

Meeting performance specifications in the design of Analog and RF (A/RF) blocks and integrated circuits (IC) continues to require a high degree of skill, creativity, and expertise. However, today's A/RF designers are increasingly faced with a new challenge. Functional complexity in terms modes of operation, extensive digital calibration, and architectural algorithms is now overwhelming traditional A/RF design methodologies. Functionally verifying A/RF designs is a daunting task requiring a rigorous methodology. As occurred in digital design, analog verification is becoming a separate and critical task. This paper describes the verification issues faced by the A/RF designer and presents a verification methodology to address these challenges. It presents a systematic approach to A/RF verification, the concept of an analog verification engineer, how to establish the methodology, and concludes with an example.

The is paper was first published in the February 2007 issue of "The Proceedings of the IEEE". It was last updated on November 24, 2009. Contact the authors via e-mail at consulting@designers-guide.com.

1 Introduction

During the past decade, analog, mixed-signal, and radio frequency (A/RF) design has undergone dramatic changes primarily driven by the competitive pressures of the consumer marketplace. Consumer systems such as digital camera, mobile phones, wireless/wired internet appliances, video set-top boxes and the like are perceived to be rapidly growing global markets. Many integrated circuits (IC) providers have entered these markets hoping to reap substantial rewards. This has fueled tremendous competition. With the advent of the fabless semiconductor company, chip companies can no longer differentiate on the fabrication process or costs. The barrier to entry for a new design company is low. Differentiation can now only occur in the form of better design and marketing. Under this backdrop, design failure is equated with company closures or layoffs.

In these consumer systems, A/RF plays a critical role providing the interface to the consumer and enabling high-bandwidth communication channels within the system and between systems. The challenges facing the A/RF designer are daunting. These include achieving the desired functionality with the required performance, and providing this at the lowest cost and power possible, and finishing before the competition.

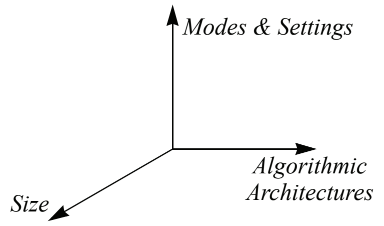
Two fundamental changes are driving failure in A/RF designs. The first is an explosion in functional complexity. The second is that the design team has changed. Both result in existing design methodologies becoming overwhelmed.

1.1 Complexity Explosion

Figure 1 illustrates the multiple dimensions in which verification complexity is exploding in A/RF design. Of course, the size of designs has grown relentlessly over time, and this has been accompanied to a shift from bipolar to CMOS and to dramatically lower supply voltages. Both of these transitions has increased the use of switching circuits because they are particularly well suited to low voltage CMOS. Switching circuits come in two forms, switched-capacitor circuits (ex. SC filters, SC data converters, etc.) and inverter/logic gate-like circuits (ring oscillators, phase detectors, etc.). By their very nature, switching circuits are algorithmic, meaning that they generally require a clock signal and they create their output over many cycles of the clock. In addition, the desire for better circuit performance has increased the use of calibration, error cancellation, and adaptive circuits, all of which are algorithmic in nature.

Over time the use of algorithmic circuits has increased dramatically, and at the same time the complexity of the algorithms and the number of cycles needed to produce the output has also increased [1,2,3,4,5,6]. Consider analog to digital converters (ADCs). The earliest integrated ADCs were flash converters that produced an output in one cycle. Then came dual slope, successive approximation, pipelined and $\Delta\Sigma$ converters. Today there is heavy used of MASH $\Delta\Sigma$ con-

FIGURE 1 *Dimensions of complexity in A/RF verification.*



verters, which implement very sophisticated algorithms and take hundreds or perhaps thousands of cycles to produce an output. The combination of large circuit size and the need to simulate a large number of cycles because the algorithmic nature of modern circuits is resulting in transistor-level simulations that require days or perhaps weeks to complete, even when using timing simulators (also known as fast or reduced-accuracy circuit simulators).

In addition to these two factors, analog circuits are increasingly parameterized and implement a large number of modes of operation. The parameterization comes in the form of digital busses that control various settings that would change during normal circuit operation, such as the gain of a stereo amplifier, or that perform trimming to compensate for variations in the processing of the silicon. The large number of modes results from the need for flexibility in large designs, which generally need to operate in a wide variety of different settings. For example, a mobile phone must support multiple communication standards, must operate properly over a wide range of operating conditions, and must conserve power in all situations. Large designs also tend to be expensive to develop and so must fit into a variety of applications to provide a large enough market to justify the development costs. With all of the various settings and modes, large analog circuits often have thousands of distinct operating configurations, each of which must be verified. And if each mode takes a week to simulate because of the large size and algorithmic nature of the circuit, then complete verification of the design using traditional transistor-level simulation becomes completely impractical.

1.2 Complex Design Team

Short design cycles and more featured functional units result in A/RF designs requiring a team of design engineers with different skills and specific roles. Also, A/RF designs are typically integrated into a larger system-on-a-chip (SoC) [9]. At the SoC level, the design team consists of the chip design lead, chip implementors and digital verification engineers. For the A/RF design, there is the A/RF design lead, the block designers, the layout engineers, and the system/digital-signal processing architects. Surrounding the design is computer-aided design (CAD) support, production and test engineers, application/product engineers, business owners, and the IC customer.

Table 1 shows the core team, and the primary concerns of these team members. While designers must focus on meeting functional and performance objectives, communication within the design team is critical to the success of the IC. In digital design, communication is based on the rigorous hand off of "sign-off" quality models that are used to communicate design intent. A/RF design employs a much looser communication mechanism usually based on a design specification document. The key hand-off points in the A/RF design are between the A/RF design lead and A/RF block designer; and between the A/RF design lead and the chip lead where the A/RF world crosses into the digital world [10]. As functional complexity grows both hand-off points are becoming more critical.

TABLE 1 *Primary Concerns of the Design Engineers*

Role	*Primary Concerns*
Chip Lead	Does the chip implement the desired function? Are the blocks connected together correctly?
A/RF design lead	Will the unit work in all modes? Is the model being delivered to the chip lead accurate?
A/RF block designer	Will the A/RF block meet performance requirements under all process conditions?

1.3 Impact of Changes

Design and design team complexity has altered the nature of A/RF design. Functional failures in A/RF are now taking precedent over failures stemming from not meeting performance specifications. The shear number of control lines that need to be checked in various combinations is overwhelming. However, performance specifications such as signal-to-noise ratio, noise figure, and phase noise continue to get more and more difficult to achieve. It has been the tradition of A/RF designers to focus on the performance challenges and they continue to do so, with it consuming most of their design effort. Little time is available, nor is it their mind set, to now check all of the specifications while changing the hundreds of control signals. Where verifying performance specifications is the traditional problem of simulating an ever increasing number of transistors in a simulator, A/RF functional verification grows in difficulty with the number of modes that need to be supported.

Functional failures can come in the form of inoperable modes. A PLL could be delivered with only a fraction of its modes working without the designer or customer being aware until after manufacturing. Worse, there are now many examples of failed ICs due to "simple" errors such as inverted signals, swapped most significant bit (MSB)/ least significant bit (LSB), and incorrect sequencing of power up.

Functional failures tend to result in failed chips that are show stoppers. The end customer cannot begin bringing up the firmware that is typically run on the SoCs. A performance failure is not as fatal because the system development can usually continue while the IC is re-spun for performance improvement. The end result of functional failures is many design iterations (re-spins). These are usually at great expense in terms of non-recurring engineering costs (NRE) and missed market windows.

1.4 Claims

As happened in digital design in the late 1990's, complexity is now overwhelming design teams and causing errors in the design of A/RF ICs, functional units, and basic building blocks. In digital design, verification separated from design and today dominates the majority of the engineering effort. We make the following claims:

- A systematic functional verification methodology is required today for A/RF designs [14,15].
- A new type of engineer may be required — the A/RF verification engineer.
- The proposed methodology can be used today to derive benefit.
- The methodology provides most value to the engineering manager, A/RF design leads, and system integrators — the engineers who are most concerned with having a working chip, accurate specifications, functionality, and verified models such as a high-level description language (HDL) model to facilitate integration in the SoC.
- It provides a launching point for A/RF re-use.

1.5 Goal of Paper

This paper describes a systematic A/RF functional verification methodology and provides a starting point for the A/RF design team and verification engineer. Existing approaches will be described. A new nomenclature for A/RF verification based heavily on the terminology used in digital verification [11,12] is employed. We will also put into this context answers to many open questions in the analog CAD such as the role of the system design tools, analog modeling languages, and how to accelerate simulation while ensuring accuracy. We provide an example and make some concluding remarks. Finally, a glossary is provided at the end of this paper.

2 Existing Approaches

Many approaches and combinations of approaches are used today to address the complexity and teamwork challenges. We segment them into design approaches and CAD approaches.

2.1 Design Approaches

Understanding that verification is a daunting challenge with no solution, designers apply design techniques to reduce the need for verification. There are several approaches:

- Separate analog and digital — the need for verification in the A/RF team is reduced because the digital functionality has been moved out.
- Everything programmable — make all functions controllable, so that if there are errors in the design these errors can be corrected in firmware.
- Everything tunable — make all resistors, capacitors, current sources, and voltages sources tunable to boost performance and to tune out process imperfections.

Although these are powerful techniques, substantial penalties are paid for employing these approaches. In separating analog from digital, penalties include a loss of control of part of the design, limits on architectural choices, increases in the input/output (I/O) lines between analog and digital portions, and added test challenges. In the programmable and tunable approaches where errors can be fixed after manufacturing, complexity is added to allow for the programmability, which if not verified can be sources of error themselves. Further, without a verification methodology, late changes in the functionality of a design causes endless incongruities between the design, models, register tables, and documentation.

Another approach that is more of an unconscious choice is to design defensively. Knowing the limitations of the design tools, simulators, and other constraints, designers limit their design choices to avoid potential problems. In particular, without a strong verification methodology, complex architectures may be avoided.

2.2 CAD Approaches

2.2.1 Speed Up Simulation

Several approaches exist. These include mixed-mode simulation (mixing transistor and RTL); mixed-level simulation (mixing model and transistors); exploiting the characteristics of the circuit, such as timing simulators and RF simulators; and better simulation algorithms. These approaches all require additional effort and engineering trade-off decisions on the part of the A/RF designer — usually accuracy vs. speed. Better simulation algorithms may hold promise, but development has been slow in this field relative to what is required.

A combination of simulators is required today. There is no one-size-fits-all solution. Without a verification methodology to provide context, use of these tools help, but lack consistency from designer to designer and there is uncertainty as to the level of verification that is actually provided.

2.2.2 Correct-by-Construction

With the promise of digital synthesis, much research and several startups have attempted to provide synthesis for analog. None have succeeded to date. Fundamentally, no one has invented a transformation mechanism that does not require verification as part of the transformation. In digital, the use of Boolean algebra can be proven to be correct-by-construction. All of the analog techniques to date require the use of simulation, models, or equations to verify correctness. Only limited success has been made at the operational amplifier level on hundreds of transistors and where only performance specifications have been considered.

Interestingly, digital synthesis made a leap frog advance in addressing the digital verification challenge, but is now again caught by verification where increasingly designers want to describe higher level functions, but no transformation has been invented to move from those higher level functions to the register transfer level (RTL).

2.3 Summary

Designers are applying what they can to meet the new challenges. Often, designers do not see the problems as the design manager sees them. They often misjudge the effort to make the entire SoC work together. With the SoC being a single product developed by as many as 20 to 50 engineers, more systematic approaches must be taken to A/RF verification, otherwise chips will continue to fail due to lack of communication and uniformity in verification and design techniques.

3 Verification Methodology

The goal of the verification methodology is to systematically find errors in A/RF design in a reproducible manner. It also provides several substantial collateral benefits. It tends to make the design process itself more efficient and predictable [7,8]. It produces verified models and testbenches of the design that aids in reuse and integration. It can be used to accelerate performance verification, and it tends to uncover errors in the specification documentation.

3.1 Scope and Assumptions

There are many practical outcomes of this methodology. For this paper, we focus on answering the following critical questions:

- Does the design meet the intended function?
- Do the customer required models of the design match the actual design?
- Is the design being used as expected?
- Is the design documentation correct?

We assume that existing verification techniques such as static timing analysis, digital functional testing, layout vs. schematic, design rule checking, and electrical rule checking are being applied judiciously. We also assume that the design is sufficiently complex that it cannot be simulated at the transistor level for all of the tests required. If not the case, verification is much simpler and is a degenerate case of what is described.

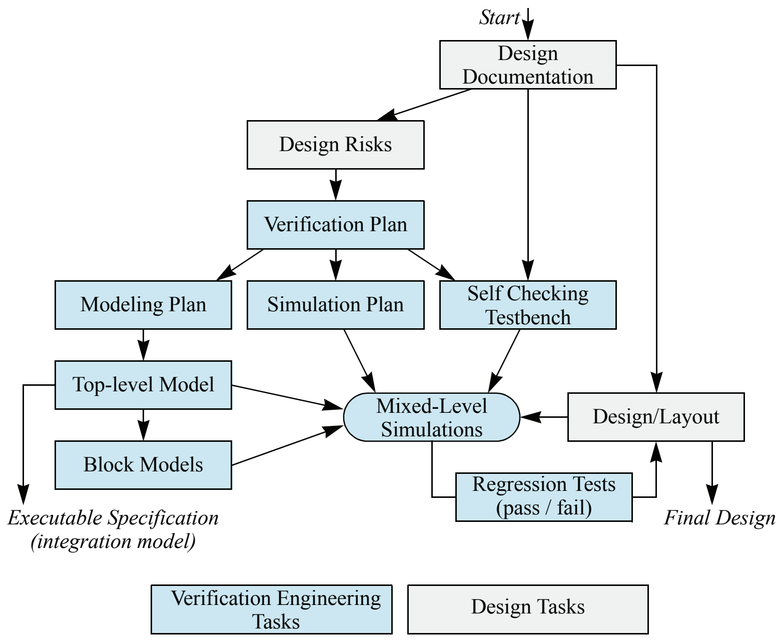
The primary focus of the methodology described is on functional verification as distinct from performance verification. However, we believe that over time that the methodology can be extended to cover performance verification.

3.2 Key Concepts

3.2.1 Verification Plan

An A/RF verification flow is illustrated in Figure 2. The starting point of an A/RF functional unit design is the design specifications in the form of *design documentation*. The outputs from the design flow are the *final design* and a *top-level model*. The final design consists of the schematics, layout, and other implementation views [10]. The top-level model is a model of the functional unit, sufficient for the chip integration team to test the functional unit's behavior and verify connectivity in the context of the SoC. It is usually written in an HDL.

FIGURE 2 *The A/RF verification flow.*



The verification flow begins with the *verification plan*, which is derived directly from the design requirements document. In addition, design risks are identified by the designers and verification strategies are developed to mitigate those risks. The verification plan includes the identified issues and presents a brief description of how verification is to be done.

3.2.2 Mixed-Level Simulation

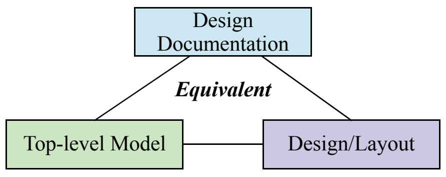
The key tool in A/RF verification is the circuit simulator. A fundamental simulation issue in today’s complex A/RF designs is that circuit simulators are too slow to simulate the entire A/RF design for even for 1 or 2 tests. Tests can take weeks to months to simulate. They are certainly too slow for the hundreds of tests needed to cover all the operating modes. At the heart of this methodology is a technique called *mixed-level simulation* [8,13,16] to work around this issue. Mixed-level simulation is a means by which HDL or AHDL models can be run with transistor level models. For example, in a particular test, the critical circuits being tested will be left at transistor level while the non-critical circuits such as bias generators or digital logic will be left at AHDL or HDL respectively. In this way, each test runs considerably faster, and many tests can be run at once with the result being that simulations complete hundreds to thousands of times sooner.

In this paper, we will use Verilog-AMS [7,17] for our AHDL and Verilog [18] for our HDL. VHDL-AMS [19,20] and VHDL [21] can also be used. Also, for HDL, new languages such as SystemVerilog [22] and SystemC [23] can also be used as long as a mixed-signal simulator exists for those languages.

3.2.3 Simulation and Modeling Plans

Because mixed-level simulation is not as straightforward as simulating the entire design at the transistor level, careful planning is required to ensure verification is done properly. The *modeling plan* and the *simulation plan* are the processes by which the verification plan is implemented. Typically, there is a *top-level model* and the *block models*. The top-level model serves as the “sign-off” quality *executable specification* delivered to the integrator of the functional unit. The modeling plan describes what AHDL and HDL models need to be written, and the simulation plan describes the testbench and all of the simulation *configurations* that need to be run. Configurations specify for each test which part of the circuit should be at transistor level. The block models are used to enable mixed-level simulation.

FIGURE 3 *Equivalent design representations.*



3.2.4 Self-Checking Testbenches and Regression Testing

Full benefit from this methodology is derived by applying an *autonomous* or *self checking testbench*. The self checking testbench drives the circuit and captures the simulated results and compares them to the expected results as specified in the design documentation. These tests are extensive, testing top-level as well as block level behavior. Mixed-level simulations are run as described in the simulation plan. The goal shown in Figure 3 is to verify that the design matches the design intent and that the design matches the top-level model. The testbench is described in detail in Section 3.4. Because the documentation, design, and top-level model are now linked, the model is, in fact, an *executable specification* that can be delivered to the design's customer and to other designers. It is a far less ambiguous means of communication than written documentation, which is often inaccurate and out of date.

The expected results can be placed inline with the tests, in separate files, or a *golden model* can be established where the golden model itself represents the expected behavior. In the latter case, the self checking testbench would apply stimulus to the design and the golden model, compare results and look for discrepancies. In this paper, we apply the former technique where expected results are inline with the tests.

The key reason for choosing the self checking approach is to enable automated *regression testing*. Tests run automatically on a nightly or continual basis. Summaries are posted or sent to designers to alert them to issues. It is impractical to expect designers to inspect detailed results such as individual waveforms after hundreds of tests are run. Without automation, the level of testing required in today's A/RF designs is not practical.

3.2.5 Analog Verification Engineers

In Figure 2, note that the design and verification tasks are separated. To execute the verification tasks, some design teams have employed the use of a new type of engineer, the *analog verification engineer*. This engineer is analogous to the digital verification engineer. The job of this engineer(s) is to focus on verification of the A/RF design at the functional unit and the block level. There are several reasons for requiring a new engineer.

- New skills are required that are usually not present in design teams — verification engineers must be proficient at modeling, developing tests, well versed in digital and analog modeling languages, simulators used in digital, mixed-mode, and analog, and scripting and batch mode operation of regression testing.
- The need to focus — developing models and tests is a substantial effort, and in most projects is too large of a job to be done part time by a design engineer. Furthermore, designers are often consumed by their design and so cannot give the required attention to the verification task.

- Degree of independence — if the engineer that is designing a circuit is also responsible for verifying it, then any misunderstanding of the requirements will pollute both the design and the tests and so will not be caught. This is much less likely to happen with a verification engineer, especially if the verification engineer keeps a certain distance from the design engineers and works primary from the specification.

The analog verification engineer does not have to be as design knowledgeable as the analog designer as they will not be designing, but they do need to understand rudimentary analog design concepts and techniques, terminology, and the use of analog design tools.

Although an analog verification engineers is an extra cost, improved efficiencies in the design process can offset this burden. For example, the addition of verification engineers can off-load modeling and functional verification tasks from the circuit designers. Instead they can focus on performance validation and on innovating new and more promising architectures. In addition, the overall efficiency of the design process can be improved as new skills are being brought to the design team, skills that designers often lack.

Disadvantages to having a separate analog verification engineer include the possibility of reducing analog design ownership and depending on the familiarity of the analog verification engineer with the design, added time may be required to learn the design. An alternative approach is to split the role of the analog verification engineer between the system verification engineer, the analog design lead, and the analog designers. In the end, whether or not to have a dedicated analog verification engineer is very project and team dependent. Throughout the rest of the paper we will refer to the analog verification engineer, but alternate staffing approaches can be used to accomplish the verification tasks.

3.2.6 Benefits

The key benefits of this methodology are:

- Designers are always simulating their blocks in the context of the overall functional unit
- Extensive regression tests are always being run to make sure that interfaces are correct
- The design documentation, the HDL model and the design is always synchronized.
- Design time can be accelerated since errors are caught earlier. In the least, since the modeling and verification is done by a separate person, the design should be in no way hindered by using the verification methodology.
- Designers can spend more time designing. They can explore new circuits and new architectures.
- More functionally complex designs can be undertaken knowing there is systematic verification in parallel.

3.3 Verification in Context of Design

Rarely is today's A/RF design a standalone IC. A/RF *functional units* are most often integrated into an SoC where most of the chip is digital. For the purposes of this paper, we define basic levels in the design.

Chip — this is the complete IC or SoC.

Functional unit — this is part of the IC that is to be shipped from one design group to be integrated by another. Practically speaking, in analog, examples are an RF receiver, a CODEC, and a Serial Deserializer (SerDes). The functional unit is typically designed by several designers.

Block — This is a basic analog block that is the job of one designer to deliver. Examples include A/D converters, digital-to-analog (D/A) converters, and PLLs.

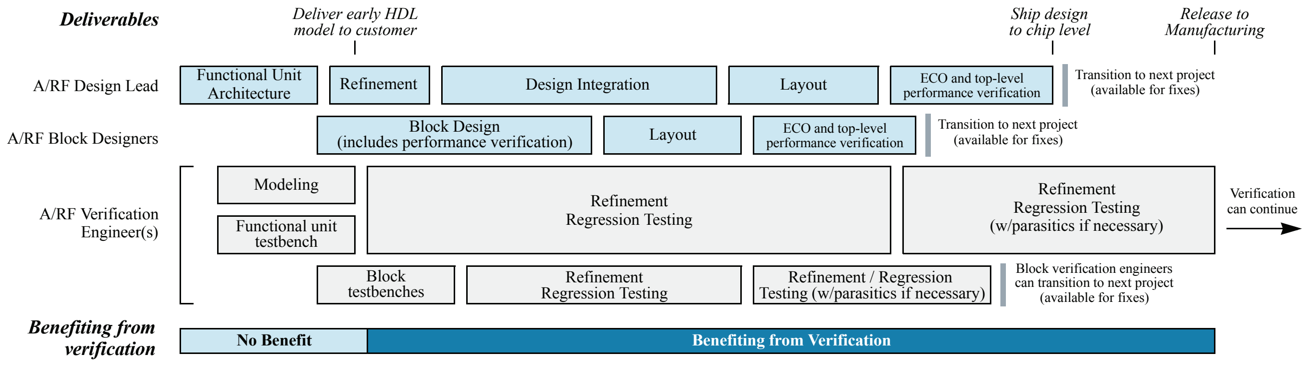
RTL or Gate — description of a digital design in terms of registers, combinational logic, low-level buses and control circuits usually implemented by finite state machines (FSMs)

Transistor — This refers to the device level — transistors, capacitors, resistors, inductors.

3.3.1 Design Time Line

The ideal design and verification project time line is shown in Figure 4. The design project begins with the creation of the specifications for the A/RF functional unit. These are generally derived by the chip system engineer. Once the chip architecture is solidified, the system engineer works with the *A/RF design lead* to formulate specifications for the analog portion of the design based on an estimate of what is possible in the A/RF functional unit. Reuse of existing components is often taken into account.

FIGURE 4 *Ideal A/RF design and verification project time line.*


Deliverables
Deliver early HDL model to customer
Ship design to chip level
Release to Manufacturing
A/RF Design Lead
Functional Unit Architecture
Refinement
Design Integration
Layout
ECO and top-level performance verification
Transition to next project (available for fixes)
A/RF Block Designers
Block Design (includes performance verification)
Layout
ECO and top-level performance verification
Transition to next project (available for fixes)
A/RF Verification Engineer(s)
Modeling
Functional unit testbench
Refinement Regression Testing
Refinement Regression Testing (w/parasitics if necessary)
Verification can continue
Block testbenches
Refinement Regression Testing
Refinement / Regression Testing (w/parasitics if necessary)
Block verification engineers can transition to next project (available for fixes)
Benefiting from verification
No Benefit
Benefiting from Verification

At the functional unit level, analog architectural questions are explored such as determining the best architecture for blocks and what are the detailed parametric design values to be used. System exploration tools such as Matlab, Simulink [24,25], Ptolomy [26], and even a generic spreadsheet are helpful in this phase because they allow designers to quickly explore their design and to transform the design requirements into design parameters [27,28]. For example, Matlab is adept at converting filter characteristics into filter coefficients. At this point in the design flow the focus is on modeling the signal flow through the system with an emphasis on estimating expected performance, so second-order effects are often included. However, details such as modeling the control flow or interface details between the blocks are rarely modeled, because it would slow down architectural exploration.

This is the time when the *analog verification lead* joins the project. He/she works with the design lead to develop models for each block in Verilog-AMS. Verilog-AMS is used rather than a language like Matlab because these models will eventually be used to perform mixed-level simulation. Here the modeling focus changes from the system design phase to the implementation phase where verification is a key goal. Only the function of the blocks is modeled with little to no effort expended to modeling second-order effects. Instead the emphasis is on modeling all of the functionality, including control flow and the interfaces. Thus, system exploration tools are the tool of the design engineer, while AHDLs are for the verification engineer. At this point, the verification engineer can also influence the design to make it more verifiable and testable. The outcome of this phase is a first cut at a partitioned pin-accurate top-level functional unit schematic with behavioral models for each of the blocks that is simulatable and implements all functional aspects of the specification. The block designers now join the project.

The block designers start with the documentation, the top-level schematic, and the models, and use them to gain an understanding of what they are expected to design. Meanwhile, the verification engineer or engineers (more would join the project at this point if it is large or complex) would begin developing block-level and then functional unit-level tests. This is where rigor is brought into design management. Two types of tests are developed. *Quick tests* are a set of basic tests that can ideally run in minutes and can be used by designers before they “check-in” their blocks to assure that they meet minimum functionality and compatibility requirements. Regression tests are more extensive, designed for overnight runs. It is important for designers to check in their designs often so that the verification engineer can run these more extensive tests. It also helps to keep the overall design synchronized. With the designers’ focus on performance, they will likely also use the models written by the verification engineer to speed up simulation by replacing noncritical circuitry with simple functional models. For higher level specifications such as bit error rate (BER) and signal-to-noise ratio (SNR), more complex verification post-processing can be included. The post processing can be custom written or the system exploration tools can once again be used.

Besides developing and running tests, the verification engineers also develop and test the functional unit Verilog model if one needs to be delivered to the customer. They would also be expected to perform verification reviews that include themselves, the block designers, the analog lead, and CAD support to assure that their tests are both comprehensive and efficient.

Once the top-level schematic and models have settled out it is possible to bring the test engineers onto the project. They would use the models to begin developing the tests that will be used on the production floor. Bringing the test engineers in early allows the tests to be developed in parallel with the design effort. This can shorten first customer shipments and allows them to contribute suggestions that would make the design more testable [29,30,31].

As the design approaches completion, the focus of the verification engineer shifts to more comprehensive functional unit-level tests. Every block should be simulated at the transistor level within the larger system in a mixed-level simulation. If there is a possibility of subtle interaction between multiple blocks, the blocks should be simulated together at the transistor level. An example of when this is necessary is when the design includes a separate bias block. That block should be run pair-wise at the transistor level with every block it feeds.

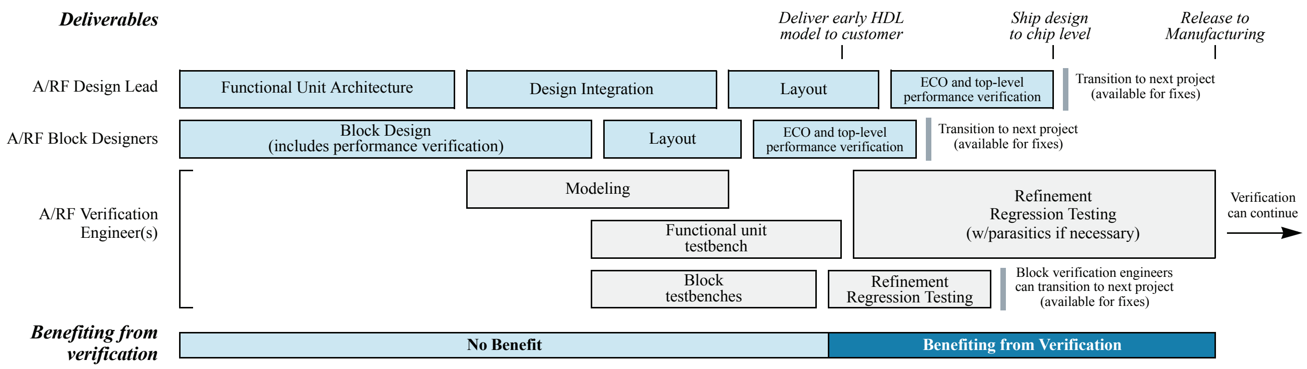
Maximum benefit is derived from the verification tasks by following the time line shown at the bottom of Figure 4. Verification starts at the beginning of the project the design benefits from automated regression testing throughout most of the project.

3.3.2 Incomplete Implementation of the Methodology

If at the start of the project, the design lead cannot (or chooses not to) define the architecture, functionality, and the interfaces in enough detail that the top-level models and testbenches can be developed, then the time line reverts to what is shown in Figure 5. There are few differences in the two project time lines. Besides not having the more solidified top-level specification, the only other difference is when the block designers start on the project. The level of effort is virtually identical. However, the difference in the amount of verification that can occur is dramatic. In the ideal case, during the entire design phase there is benefit from verification. In the late decision case, benefit is only derived during final layout. There are several disadvantages. Design is slower because designers need to take the time to continually check functionality. Also, errors that are caught by verification result in a costly engineering change order (ECO) where layout has to be changed.

For maximum benefit, the design lead must be willing to make an early decision on interfaces and block functionality. This does not mean these decisions are set in stone. As long as these interfaces do not change on a daily level, the verification methodology will work. Making early decisions is usually to the benefit of the lead designer. Too often, if these decisions are not made, the block designers end up making the interface decisions and the lead designer is constantly behind trying to keep the design synchronized while the block designers continue to make changes. The lead designer is also the one that derives most of the benefit

FIGURE 5 *A/RF design and verification project time line with partial implementation of methodology.*


Deliverables
Deliver early HDL model to customer
Ship design to chip level
Release to Manufacturing
A/RF Design Lead
Functional Unit Architecture
Design Integration
Layout
ECO and top-level performance verification
Transition to next project (available for fixes)
A/RF Block Designers
Block Design (includes performance verification)
Layout
ECO and top-level performance verification
Transition to next project (available for fixes)
A/RF Verification Engineer(s)
Modeling
Functional unit testbench
Refinement Regression Testing (w/parasitics if necessary)
Verification can continue
Block testbenches
Refinement Regression Testing
Block verification engineers can transition to next project (available for fixes)
Benefiting from verification
No Benefit
Benefiting from Verification

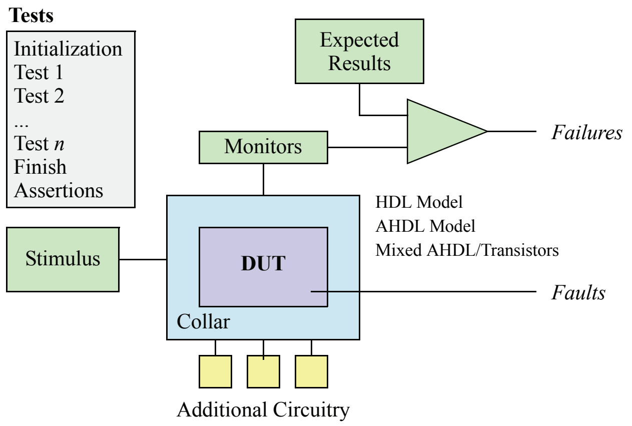
from this methodology — as the person who is ultimately responsible for ensuring that the functional unit is designed correctly.

Using the “top-down” and “bottom-up” terms [32,33,34,35], we describe this methodology as architecturally and functionally top-down with performance designed in a bottom-up style. In general, design teams already follow this approach. The difference is in the level of solidity in the functionality going down. We claim that with just a bit more definition, huge benefits can be derived with verification.

3.4 The Testbench

Apart from the *device-under-test* (DUT), the testbench (shown in Figure 6) contains all of the necessary components for verification. It is the responsibility of the lead verification engineer to assemble the testbench.

FIGURE 6 *Elements of the self-checking testbench.*



The DUT is tested by attaching it to a testbench. The testbench provides the inputs necessary to drive the device while monitoring its output. The term DUT refers to each of the multiple representations of the design — Verilog, Verilog-AMS, and Verilog-AMS mixed with transistors.

The actual testing of the DUT is performed in two different ways: *assertions* and *tests* [12]. An *assertion* is modeling code that continually observes one or more signals and raises a *fault* when it detects an error condition. For example, an assertion may monitor a bias line and an enable line to verify that the current on the bias line should be 10μA ±10% if its enable line is high and zero otherwise. Other uses include checking setup and hold times, checking that illegal codes are not generated, and even checking that the input/output characteristics of blocks are implemented properly.

3.4.1 Assertions

Assertions have the property that they are always checked, regardless of what tests are running. They can be used advantageously in several places. First, they are in the testbench where they look for unexpected behavior from the DUT. Second, they can be placed in the models or in the circuit where they check that the design is being used correctly. For example, when modeling a block, it is generally unnecessary to model the behavior of the block as a function of the bias signals, rather one models the primary function of the block and simply asserts that the bias signals are present and within expected tolerances. Also, assertions can be placed in the model or circuits that are to be delivered to an end customer. In this case, the assertions ensure that the circuit is properly interfaced to the remainder of the system and being used properly. Too often, a block or functional unit is re-used where the designer leaves the project or has forgotten the design and use details. The assertions continue to ensure that the design is being used properly.

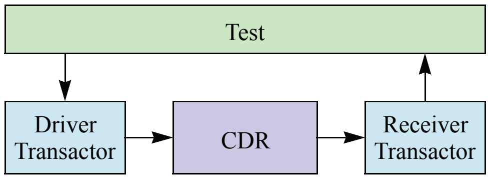
3.4.2 Tests

A *test* is a grouping of a *stimulus*, a *monitor*, and a comparison to check against the *expected response*. The test applies the stimulus to the DUT and compares the *achieved results* with the expected results to determine a pass/fail response. The stimulus is the signal or a sequence of signals applied to the DUT. A monitor is used to observe the output. The monitor could be as simple as observing a current or voltage, or could be more complicated, taking several signals and processing them. Additional code is then added to compare against the expected results. Here, a tolerance or bounds may need to be specified when comparing analog signals. A *failure* occurs if the comparison finds that the actual and expected results are different or different beyond a certain tolerance.

3.4.3 Transactions and Transactors

To enable re-use of tests within the same design or between designs, tests can be written in an abstract manner by employing a transaction-based interface to the DUT. A *transaction* is a sequence of signal transitions on a group of one or more pins that accomplishes a single task. A *transactor* is a block of code that acts as an interface between the test code and the DUT by converting between high-level test instructions and low-level transactions with the DUT.

FIGURE 7 *Using transactors to test the bit-error rate of a clock & data recovery stage.*



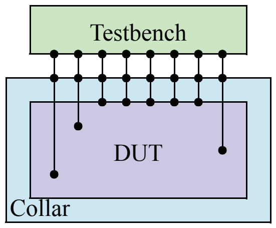
For example, a BER test for a wire-line clock & data recovery stage (CDR) might produce a long sequence of abstract symbols that are fed to the receiver and then compared to the values actually received, as shown in Figure 7. Transactors would be used to connect this conceptually simple test to the CDR. One transactor might perform a 8b/10b encoding of the multi-bit symbols produced by the test before converting them into to a serial low voltage differential signaling (LVDS) stream that is fed to the input of the CDR. Another transactor may be used to observe the serial bit stream produced by the CDR, deserialize and decode it, and then present the results back to the test for checking. In this way, the BER test is quite reusable. It could be testing the BER of anything. For example, it could be applied to the CDR as shown, to a SerDes, to an entire wireline receiver, or perhaps to a transceiver configured in loop-back mode. It could even be applied to entirely different types of system, such as data converters. For each of these, the test would remain the same, only the transactors need be changed. In addition, the BER test could be replaced with other tests. For example, to stress the CDR, the BER test, which tends to produce evenly distributed random symbols, can be replaced with a test that produces corner case symbol streams, such as those with long run lengths. This shows that the transactors are also reusable.

3.4.4 Collars

A *collar* surrounds the DUT as shown in Figure 8. The purpose of the collar is to create a wrapper so that the testbench can be applied without change to all DUT representations. In this way, there is only one testbench and DUT. The collar is considerably simpler than the testbench and changes much less frequently. As such, it is much less likely that an error be created and go undetected if there are multiple versions of the collar than if there are multiple versions of the testbench.

The collar performs several important functions. The first is to allow the testbench to observe internal signals of the DUT in a manner that is portable across all versions of the DUT. In A/RF design, there are many modes associated with small internal adjustments used for compensating for second order effects. Just connecting to the DUT's terminals does not provide adequate testing. For example, a typical adjustment is for the common-mode voltage at the output of one

FIGURE 8 *The testbench connects to the DUT through a collar, which is responsible for mapping signals internal to the DUT.*



stage going into another. If this is not observable from the DUT's terminals, it is desirable to probe into the DUT and observe the common-mode voltages directly. In both the circuit-level and Verilog-AMS versions of the device, the common-mode voltages could be observed using Verilog's hierarchical names to directly probe the outputs or inputs of each stage. However, this level of the hierarchy likely does not exist in the high-level Verilog model. Rather the common-mode voltages would likely be stored on local variables within the Verilog DUT. In this case we add a collar that acts as a wrapper and takes responsibility for mapping all of the signals of interest in the DUT to its periphery for access by the testbench.

Another important function of the collar is to make any language conversions required. If the testbench is in Verilog-AMS, then the collar must covert other representations of the DUT to be Verilog-AMS compatible. For example, Verilog does not allow for real number terminals. Often, due to pin matching requirements in the flow, analog pins in Verilog are represented by a single bit. However, for the DUT models to behave properly, Verilog models have to have analog inputs and outputs. Fortunately, Verilog can operate on real numbers. In this case, the function of the collar is to allow external blocks to pass in and pull out real number values.

Where no mapping is required, a simple pass through collar may be required to maintain hierarchical consistency between the representations. Multiple versions of the collar are developed as needed for each of the DUT representations.

Finally, in the testbench, there may be additional circuitry such as the DUT's load, drivers for the input, or filters at the input and output.

3.5 Roles of Engineers

As shown in Figure 4, each engineer has distinct roles to play. Table 2 describes the main tasks that need to be completed with a focus on the verification tasks. It describes which engineer is primarily responsible and what defines completion.

Key to this methodology is separating roles and responsibilities. The verification engineer should never modify the design. The designer can suggest additions to the testbench, but it is the verification engineer who must approve

TABLE 2 *Engineering Roles*

Task	Engineer	Tasks	Sign-off
Functional unit models	Lead verification engineer	Develop the HDL model Integrate AHDL models	Passes self checking testbench
Block level models	Verification engineer	Develop AHDL models	Passes self checking testbench
Functional unit testbench	Verification engineer and lead designer.	Verification engineer focuses on specifications & customer expectations. Lead designer focuses on block interaction issues, requirement for testing fabricated part, and prime concerns in the design.	Verification review
Block level testbenches	Verification engineer and block designers	Verification engineer focuses on specifications and block designers focus on their primary concerns	Verification review
Design/layout	Lead designers, block designers, layout engineers	Design, layout, standard verification tasks (DRC, LVS, ERC) and delivery of design	Design review Passes self checking testbench
Documentation	Design lead	Provide end user description of the functional unit and how to use it	Design review Verification review

changes. Furthermore the design engineers should update the specifications when needed, but it is the verification engineer that must assure that the design conforms to the specifications, so when updates to the specification are needed, the verification engineer waits until the specification is updated before updating the tests.

3.6 Design and Verification Re-use

Reusing testbenches represents a powerful step toward design for re-use in analog. Unlike the transistor level design which will certainly change, the testbench can be re-used or used as a starting point with little change especially in the cases of a standards based design or a design shrink for cost reduction. When augmenting a design, where backward compatibility to functions has to be kept, this provides a great way to ensure that previous functions are being implemented to previous specifications.

4 Establishing the Design Methodology

This section serves as a guide implementing this methodology. We hit upon key tasks in verification and provide recommendations.

4.1 Writing Models, Testbenches, Regression Tests

One of the seemingly most daunting tasks is writing the models. We believe this is not nearly as challenging and unbounded as perceived *if* models are written in the context of a verification methodology.

When developing the models, the primary concern should be on modeling the basic functionality of the block. Each model should accurately model the interface of the block so that the model can replace the block in simulation. This means the number and type of pins must match, and the signal levels and impedances must be compatible with the rest of the circuit. Out of bounds conditions need not be modeled, but rather should simply put the model is a fault state so the problem can be easily found. Modeling second order effects should be avoided as the additional code required slows both the development and execution of the model and are orthogonal to the primary goal of verifying the functionality of the circuit. Use the verification and modeling plan as a guide to determine the level of detail required. For example, a simple equation that relates the signals on the terminals is preferred to a more complicated model that tries to mimic the internal working of the block. Only mimic the structure if it is intended that mixed-level simulation be done or if it is the easiest way of mimicking essential behavior.

An example of a typical model used during functional verification is shown in Listing 1. Notice that this model is pin-accurate in that includes both the supply and bias pins; the supply currents are modeled; the bias line is modeled to the degree that the circuit that supplies the bias current to the block will operate properly; the bias current and supply voltage are checked to assure that they are within tolerance; and that the function of the block is modeled at a very simple idealized level.

LISTING 1 *Verilog-AMS functional model of a flash ADC.*

```
module flash( out, in, clk, bias, pwrdn, vdd );
    input in, clk, bias, pwrdn, vdd;
    output [15:0] out;
    electrical in, bias, vdd;
    integer i, level;
    reg pwrFault, biasFault;
    reg [15:0] d;

    always @(posedge clk) begin
        pwrFault = (V(vdd) > 1.9) || (V(vdd) < 1.7);
        biasFault = (I(vdd, bias) > 16u) || (I(vdd,bias) < 14u);
        level = 16*(V(in)+0.5);    // convert input to an integer
        for (i=0; i<16; i=i+1)
            d[i] = (i < level);
    end
    assign out = (pwrdn || pwrFault || biasFault) ? 16'bx : d;

    analog begin
        V(vdd,bias) <+ pwrdn ? 0 : 0.5 + 20k*I(vdd,bias);
        I(vdd) <+ pwrdn ? 1u : 500u;
    end
endmodule
```

Questions often arise as to which language(s) should be used for the models. There are two functional unit level models. The first is a complete behavioral model in Verilog. This is to be delivered to the functional unit's customer. The second is a pin-accurate netlist in Verilog-AMS. This model instantiates the lower level blocks, either the transistor representation or the behavioral model. Usually, design environments allow this Verilog-AMS netlist to be generated automatically. This functional unit netlist is the top-level for the A/RF mixed-level simulation. The A/RF blocks are modeled behaviorally in Verilog-AMS in order to allow mixed-level simulation. Languages such as Matlab would not be appropriate as it does not support co-simulation with transistor-level circuits. If blocks are too large for transistor level simulation than a further decomposition can be done where there is a block level netlist and lower level block behavioral models. The digital blocks in the A/RF design can remain at gate or RTL. Often, a Verilog netlist can be automatically generated where the gate functional description is found in the standard cell library. The DUT collars are written in Verilog-AMS.

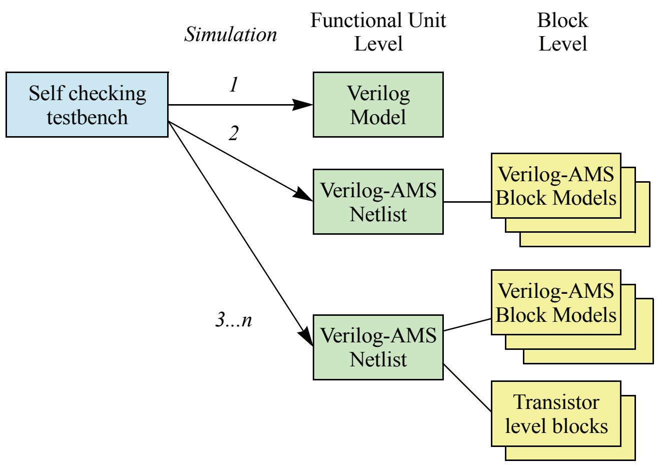
The self-checking testbench components are all written in Verilog-AMS. This gives maximum flexibility in quickly being able to change the testbench and conditions during different parts of the test.

A scripting language such as Tcl can be used for post processing simulation results. Via the programming language interface (PLI), Tcl can control a mixed-signal simulator, and pull out simulation results. Additional analysis on data such as a Fast Fourier Transform (FFT) can be easily applied or Tcl can do any format conversion necessary to send simulation results it to a system exploration tool such as Matlab. Regression testing can be controlled by Tcl or by the scripting language best supported by the design environment.

We recommend that tests and transactions be written in such a way that tests can be easily combined to create a larger test, where the tests continue to self check and add to overall coverage. In this way, a user can create a new test without understanding the detailed functionality of other blocks (e.g. a loop back test where a designer or verification engineer understands one block, but not the other). Tests can be easily aggregated to create larger system tests, but ensure that lower tests are still being conducted. It aids in building modular tests that can be re-used.

To ensure that the model has been written correctly, we rely on the self checking testbench. We apply the testbench systematically to all of the representations of the DUT as shown in Figure 9. All of the simulations should be run. The best starting point is with Simulation #2 shown in the figure. Simulation times are fast and this configuration serves as a good vehicle to debug the testbench and the Verilog-AMS models. Recall that the self checks in the testbench and the behavior of the models are derived from the design documentation. Simulations #3... *n* are applied to ensure that the specifications in the documentation match the transistor level. *n* varies depending on the number of configurations. Finally,

FIGURE 9 *Ensuring that the models are written correctly*



simulation #1 is run to make sure that the Verilog model being delivered to the customer is correct.

One of the factors in determining the number of configurations is where the design is. We define three phases of design:

Design — designer focused on establishing the topology of the circuit and then on improving a small number of circuit performance metrics

Verification — verification engineer focused on confirming that the circuit behaves as expected

ECO or last minute change — A last minute necessary change

During design, rapid turnaround (less than 15 minutes) is required, with longer turnaround (1-2 hour) tolerated on more difficult tests. During the early phases of verification, overnight is okay, and towards the end of the verification phase or during ECO where few changes are made, several day simulations or over the weekend are fine. Near the end of the design, the transistor level representation can also include the parasitics. This will slow down the simulations, but because the design is changing very slowly or not at all, this can be done. Table 3 lists our recommendations to setting up the configurations and choosing the tests.

TABLE 3 *Setting up the simulation configurations*

Phase of Design	Testing Priorities
Design	Simple quick tests with few transactions and quick setup. Focus tests on questions being asked in exploration
Verification	Prioritize tests so that those where errors are most like to occur are run first
ECO	Run complete configuration and parallelize as much as possible

Finally, there are a multitude of simulation choices in today's design environment. Table 4 summarizes when simulators are most likely be used, by whom, and for what purpose.

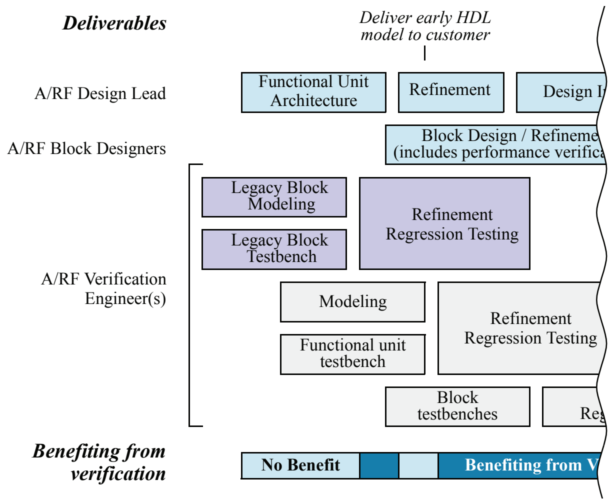
TABLE 4 *Simulators in Verification*

SIMULATOR	PERSON AND PURPOSE
circuit simulation (includes RF)	Block designers and design lead Design and performance analysis
mixed-signal simulator (mixed RTL / circuit) [36,37]	Verification engineers and designers Used to run regression tests designers can use it for performance analysis
RTL simulator	Verification engineer Used to develop RTL testbench if one needs to be delivered
timing simulator	Block designers and design lead Used to accelerate simulations where reduced accuracy and robustness can be tolerated

4.2 Using Legacy Blocks

There is almost always design re-use. It is a tried and true method of getting designs completed more quickly. Entire designs can be re-used to produce derivative products, designs can be moved to smaller process geometry to reduce costs, or blocks and functional units from a design are used as a starting point in a new design. Whatever the reason, virtually no design starts from scratch.

FIGURE 10 *Ideal time-line with re-used blocks*


Deliverables
Deliver early HDL model to customer
A/RF Design Lead
Functional Unit Architecture
Refinement
Design I
A/RF Block Designers
Block Design / Refineme
(includes performance verifica
Legacy Block Modeling
Legacy Block Testbench
Refinement Regression Testing
A/RF Verification Engineer(s)
Modeling
Functional unit testbench
Refinement Regression Testing
Block testbenches
Reg
Benefiting from verification
No Benefit
Benefiting from V

The key to verification is to provide models for those blocks. Because these are functional models, the cost is relatively low to developing these models. The ideal time line is modified and shown in Figure 10. The cost characteristic for the re-used *blocks* is similar to the one shown in Figure 5 since they are already designed. There is still benefit, since even without the verification approach described as blocks usually cannot be reused without some level of characterization. Once the blocks have been modeled and for all designs following the verification methodology, this initial penalty will no longer have to be paid for re-use.

4.3 Handling Engineering Changes

Engineering changes are inevitable. Changes are made throughout the design process. The goal is to set up guidelines, so that change can be made while keeping the design synchronized, and minimizing the impact to other designers on the project. We recommend the following:

Block Design Changes (with no port changes) — block designer runs the testbench at mixed-level with his/her block at transistor level before checking in.

Block Design Change with interface change — block designer communicates desire to change with lead designer. They make a decision. If change is required, they tell the verification engineer. The lead designer changes the documentation and the functional unit schematics. The verification engineer changes the functional unit models and AHDL model, testbench, and adds tests as needed. The block designer makes the block changes, and re-runs the new testbench. Often the functional unit AHDL model is just a netlist, so that the functional unit and schematic are identical.

Top-level change (with only top-level port changes) — The lead designer communicates with the customer of the functional unit and agrees on the change. The lead designer changes the documentation. The verification engineer changes the functional unit models.

Top-level change (with change to lower level ports) — This is similar to block design change with interface change except that the lead design initiates the process.

Testbench change — The verification engineer makes the change, and re-runs on the functional unit models before calling it “golden” again. The verification engineer should confer with the lead designer to inform him/her of the change. Future regression tests use the “golden” testbench.

Block level, Functional unit, HDL model changes — These are results of design changes illustrated above.

The data management “check-in/check-out” process should be consistent with this procedure. In particular, version control should be employed to protect designers uninvolved in a particular change from being disrupted by that change. Once a change is complete, a check out will retrieve the updated blocks, models, top-level schematic, and testbenches. Before that time, only the engineers involved with the change would check out the updated versions.

4.4 Verification Review Added to the Design Review

A design review is a typical "best practice" employed by design engineers. It is where the design team, and other senior designers gather to go over aspects of the design. These occur throughout the design process. Towards the beginning, the discussion is around architecture, functionality, and interfaces. Near the end it is about design details and layout. A verification review should be included as part of the design review. In it, the verification plan, simulation plan, and the testbench should be reviewed. Results to-date should be examined to point out where faults or failures are occurring. Sometimes, it is decided that a fault or failure is tolerable in a design. This is a decision that should be reviewed during the design review.

Model reviews should also be done where the verification engineer(s) review the models developed with the block designer so that the designer understands what is modeled and also for the block designer to point out missing or incorrect items. This can be done separately from the design review.

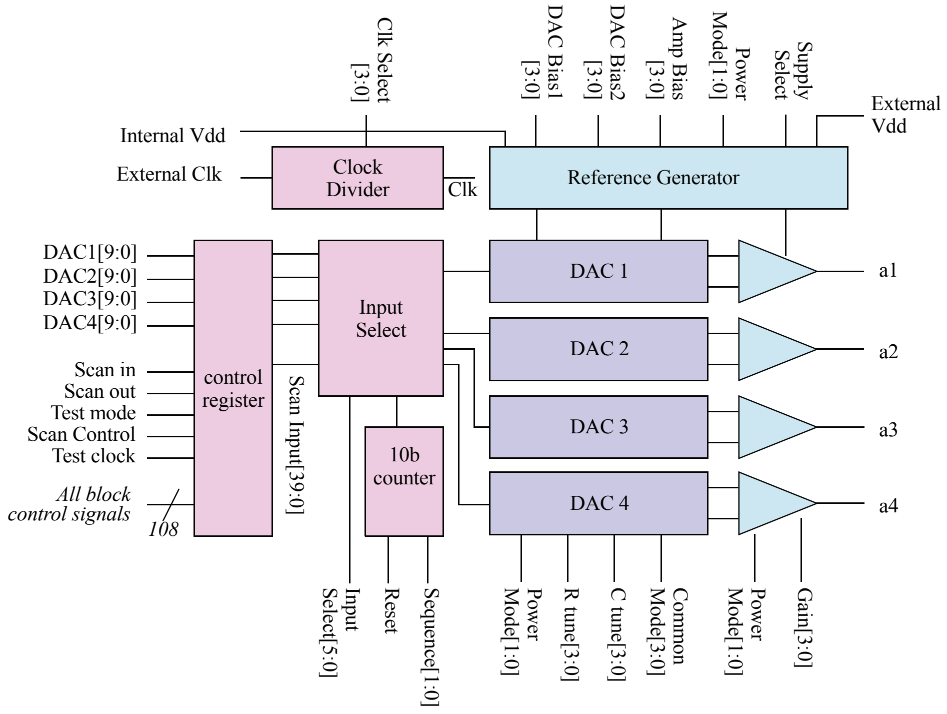
4.5 Cultural Changes

Adding the verification engineer greatly helps in adopting A/RF verification. The designers are not required to instantly become experts at modeling, test-bench writing and software scripting. However, a few key changes are required of the design team. The design lead must now be more decisive in certain steps of the design process, making decision, and sticking to them before making needed changes. The block designers must be open to sharing their work with the verification engineer. There is a great variance in how design engineers work since in the past they only had a few main milestones where communication happened. They receive the specifications, do the design, and check it in. They may not have to check in a design for weeks. Working with a verification engineer requires regular check-ins, and requires regular communication with the verification engineer. Key changes such as pin changes, or functional changes must be communicated directly. Minor changes to the design must be checked in regularly so that regression testing can occur.

5 Example

To illustrate the verification methodology, we provide a relatively simple example of an A/RF functional unit. The block diagram is shown in Figure 11. The inputs/outputs (I/O) of the functional unit are shown on the left and right. The control lines for each of the blocks are shown at the top and the bottom. All of the block control lines, a total of 108, go through the control register on the left so that all blocks can be controlled externally. This example illustrates the style of design where all control pins are exported. The number of control bits could be reduced by adding more control logic internally, but the same number of modes would still need to be tested.

FIGURE 11 *Example A/RF functional unit.*



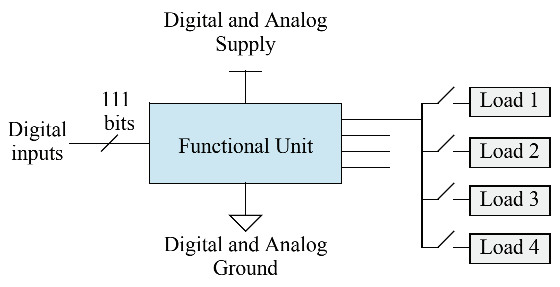
The number of signal I/O, and control pins for each of the blocks and for the functional unit is shown in Table 5. The number of signal I/O and control pins quickly add up. Not counting supply voltages and the clock input, the functional unit has 113 control pins with 40 signal inputs and 4 signal outputs. In addition,

TABLE 5 *Number of Signal I/Os and Control Pins*

Block	Signal Inputs	Signal Outputs	Control Pins
Clock Divider	1	1	4
Reference Generator	2	3	15
Input Select	90	40	6
Counter	1	10	3
4 x DAC	4 x 12	4 x 2	4 x 14
4 x Amplifier	4 x 2	4 x 1	4 x 6
Control Register	148	188	5
Functional unit	40	4	113

the amplifier has to work under four loading conditions as shown in Figure 12. This gives 4 more modes that need to be tested. Therefore, there are 2^{115} possible control combinations with 2^{40} input combinations. Of course, most of the

FIGURE 12 *Functional unit loading*



modes are independent and not all combinations have to be validated. This is why the verification plan is critical.

Finally, we assume that there are 4 design engineers on this design — a design lead, a block designer for the DACs, a designer for the amplifiers, and a designer for all the digital circuitry.

This example focuses on functional unit level verification where the blocks are verified in the context of the functional unit. This best demonstrates the power of the methodology.

5.1 The Verification Flow

The verification flow begins with the verification plan. The verification engineer works with the lead designer to determine what is most critical. The following is determined to be critical:

- Nominal operation — all control bits are set at default values and the DACs must run through all codes.
- Mode Tests — each of the control bits must be exercised from the top-level and compared against expected results
- Special Modes — certain combinations of modes are critical and must be tested from the top-level and compared against expected results
- Special Sequences — power-up and power-down are very common
- Test Mode — the functional unit must be able to be controlled via the scan control lines

Next, the modeling plan is developed. The following are the models that need to be built.

- Functional unit HDL model — this is to be delivered to the integrators of this design.
- Functional unit AMS model — this design is too large to be simulated at the transistor level. Mixed-level simulation is required to simulate at the functional unit level

- AMS model of the DAC, amplifiers, and the reference generator. For the digital control registers, counter, input selector, and clock divider, the gate or RTL representation will be used.

An example simulation plan is shown in Table 6. The configurations show what

TABLE 6 *Example Simulation Plan*

#	Configuration	Tests	Time Required
1	HDL Model	ALL	5 min.
2	AHDL Model	ALL	15 min.
3	AHDL + DACs at transistor level	(a) DAC Tests (b) remaining tests	(a) 4 hr. (b) 2 day
4	AHDL + amplifiers at transistor level	(a) Amp tests (b) remaining tests	(a) 4 hr. (b) 2 day
5	AHDL + reference generator at transistor level	(a) RefGen tests (b) remaining tests	(a) 1 hr. (b) 12 hr.
6	AHDL + DAC1 + AMP1 + reference at transistor level	All Tests	2 days
7	All analog at transistor level	Basic top-level tests	1 wk.

is to be simulated. Configurations 1-2 are used to verify the models. They start being used during the architecture and modeling phase. Cfgs 3-5 make sure all of the blocks are simulated at transistor level. These can be used by the block designers to verify their block interfaces are correct. Cfg 6 looks for circuit interaction issues. This can be run as soon as a first pass design has been completed of all of the blocks. Cfg 7 is a safety net check done near the end of the design process. The tests refer to test suites that will be created. The time required is an estimate of simulation time for each of the test and configuration combinations.

In all, there are a total of 10 simulations with 1, 2, 4a, 4b most critical to run in overnight regression mode. The remaining tests should be done as often as possible given the available computing resources. The tests are to be written in a modular manner, so if it is critical that Cfg #6 run more quickly, the tests can be run in parallel. This approach allows utilization of more resources to increase throughput.

5.2 Testbench Details

We now describe some of the testbench details. We begin with the collars. An example of a collar to wrap Verilog code is shown in Listing 2. In all of the examples, only code fragments are provided to illustrate the key points. The analog pins of the Verilog are logic pins. They are converted to electrical pins via the Verilog-AMS wrapper. We use “aout” (analog outputs) as the name of the functional unit.

LISTING 2 *Collar fragment that demonstrates using hierarchical references in the collar to reach into HDL model for real values.*

```
module aout_collar (dac1, ..., a1, ...);              // Collar for HDL model of DUT
    input [9:0] dac1; logic [9:0] dac1;
    output a1; electrical a1;
    wor fault;
    assign fault = (a1_digital ===‘bz) || (a1_digital === ‘bx));// assure line is
connected

    aout_hdl dut (.dac1(dac1_value), .a1(a1_digital), ...)// Instantiate DUT

    analog V(a1) <+ dut.a1_real;                         // Grab the analog value
endmodule

module aout_hdl (dac1, ..., a1, ...);                    // Verilog model of DUT
    input [9:0] dac1;
    output a1;
    real a1_real;
    assign a1 = ‘b1;                               // Indicate that output is connected

    always @ (...)                                 // condition such as the clock
        a1_real = dac1 ∗ full_scale / 1024;
endmodule
```

More contents of the collars are shown in Listing 3 (these are different abbreviations of some of the same modules shown in Listing 2). In this case, the verification plan calls for the observation of signals not at the functional units ports. Hierarchical references are required to access these ports, but not all of the models have the same hierarchy. In this example, the collars create a consistent location to check for the common mode voltage of the DAC and properly maps to the DUT representation.

Listing 4 provides an example of a simple test. The test consists of stimulus to thoroughly exercise all the modes and settings of the functional unit and an assertion to look for any errors. It should be noted that most of the code written in the testbench is in standard Verilog “initial” and “always” blocks. In these cases the “analog” section is only used to set and add extra circuitry to the design. The Verilog-AMS testbench is a true mix of digital and analog constructs.

With the verification, simulation, modeling plans and testbench developed, the verification flow can be executed.

LISTING 3 *Collar fragments demonstrating how testbench can access signals completely contained within the DUT.*

```
module self_checking_testbench;    // Testbench
    real common_mode_voltage;
    aout_collar dut_collar (...);       // instantiate the collar

    initial begin
        common_mode_voltage = V(dut_collar.cm_voltage);
        if (common_mode_voltage ...)// test the result
        ...
    end
endmodule

module aout_collar (...)                // Collar for the AHDL model of DUT
    real cm_voltage;
    aout_ahdl dut (...)                 // instantiate the aout AHDL model

    analog begin
        cm_voltage = V(dut.dac1.common_mode_ref);
                                        // hierarchical reference to the actual
voltage
    end
endmodule

module aout_collar (...);               // Collar for HDL model of DUT
    real cm_voltage;
    aout_hdl dut (...)                  // instantiate the aout HDL model
    always @ (common_mode_control)// react to common mode control
        cm_voltage = ...;               // calculate directly based on control
settings
endmodule
```

6 Conclusion

Functionality continues to increase in A/RF designs. The rise of the analog verification engineer is just beginning, but the need will become greater and greater over time as functional complexity continues to increase.

Applying verification in A/RF design will allow designers to continue designing more complex designs, try new architectures, and more rapidly explore new circuit techniques. More science and thought is brought to verifying the entire design, greatly reducing the risk of a chip failure. The efficiency of the design team is improved as new skills are brought to the team. Designers can focus on meeting performance objectives while verification engineers focus functional verification. Golden testbenches written by domain experts can allow designers new to the design or design types to get up to speed quickly. A methodology based on modeling smooths the growing communication issues between engineers. Finally, the models developed can also be used to help the test and product engineers for the IC.

LISTING 4 *Self checking test example.*

```
module self_checking_testbench;    // Verilog-AMS
    reg [9:0] dac1; electrical a1;
    aout_collar dut_collar(.dac1(dac1),..., .a1(a1), ...);

    initial begin
        for (i = 0; i < 16; i = i + 1) begin
            DAC_bias1 = i;             // exercise dac bias1 setting
            for (j = 0; j < 1024; j = j + 1) begin
                dac1 = j;              // sweep through input codes
                @(posedge clk) #1
                    ;                  // wait for assertion to catch errors
            end
        end
    end

    always begin                       // assertion
        @(posedge clk);
        if (dacEnable && abs(V(a1) – dac1∗full_scale / 1024) > tol)
            $display (“FAULT: dac1 code=%b”, dac1);
                                       // same for channels 2, 3 & 4
    end

    analog begin
        I(a1) <+ V(a1)/Rload + Cload∗ddt(V(a1));
                                       // same for channels 2, 3 & 4
    end
endmodule
```

Little CAD support exists for this methodology. In implementing this, we used as our basis the Verilog-AMS language, a generic scripting language, Tcl, and a mixed-mode/mixed-level simulator. There is great opportunity for new tools to be developed to support this methodology.

In writing this, we realize that there are a lot of detailed “gotchas” and decisions that need to be made that we do not have the space to include. We, however, believe that this paper can be used as a starting point for the methodology. In addition, we believe the best way to get started in using A/RF verification is to choose one project and apply dedicated verification resources to it. Make it a goal that the methodology will be used and assessed.

Glossary

The digital and system world of verification is vast. Great effort [11] has been expended in defining terminology to prevent confusion and facilitate communication between verification and design engineers. We narrow down and extend some of these definitions for A/RF Verification.

Analog Function — First order behavior of a design. For example, the function of a 10 bit D/A converter is to generate 1024 different analog levels given a 10 bit digital code.

Analog Performance Specifications — In analog, these are specifications such as Signal-to-Noise ratio (SNR), total harmonic distortion (THD), Noise Figure (NF), integral non-linearity (INL), and jitter.

Assertion — A statement of design intent that can be checked in transient simulation

Behavioral Model — An abstract representation of the design that models the function of the design and model some or all of the performance specifications.

Expected Results — These are the expected results from the device-under-test. The results could be in the form of a golden model, documentation, or conversation.

Failure — An error as a result of a particular test

Faults — An error arising from an assertion

Golden — A pre-verified reference

Monitors — Probes that observe signals in design during transient simulation

Regression Tests — Verification tests run in batch mode, with minimal human intervention with results analyzed reporting pass/fail outcomes in an automatic way.

Self Checking Testbench — A testbench that checks the response of the circuit against the expected results.

Tests/Test Suites — Combinations of transactions, monitors, and assertions together with other tests are used to verify the design.

Testbench — The overall system for applying stimulus to a design and monitoring the design for correct responses.

Transaction — A set of stimulus to take the design from one state to another

Verification — Ensuring that the designer's intended functionality and performance specifications has been correctly captured in the design.

References

[1] V. Aparin, G. Ballantyne, C. Persico, A. Cicalini, "An Integrated LMS Adaptive Filter of TX Leakage for CDMA Receiver Front Ends," *Proceedings of RFIC*, 2005.

[2] R. Bagheri, A. Mirzaei, S. Chehrazi, M. Heidari, M. Lee, M. Mokhemar, W. Tang, A. Abidi, "An 800MHz to 5GHz Software-Defined Radio Receiver in 90nm CMOS," *Proceedings of the International Solid States Conference*, 2006.

[3] J. Lim, Y. Cho, K. Jung, J. Park, J. Choi, and J. Kim, "A Wide-Band Active-RC Filter with a Fast Tuning Scheme for Wireless Communication Receivers," *Proceedings of the IEEE Custom Integrated Circuits Conference*, 2005.

[4] A. Mirzaei, R. Bagheri, S. Chehrazi, and A. Abidi, "A Second-Order Anti-aliasing Prefilter for an SDR Receiver," *Proceedings of the IEEE Custom Integrated Circuits Conference*, 2005.

[5] K. Muhammad, et al, “A Discrete Time Quad-band GSM/GPRS Receiver in a 90nm Digital CMOS Process,” *Proceedings of the IEEE Custom Integrated Circuits Conference*, 2005.

[6] Y. Palaskas, et al, “A 5GHz Class-AB Power Amplifier in 90nm CMOS with Digitally-Assisted AM-PM Correction,” *Proceedings of the IEEE Custom Integrated Circuits Conference*, 2005.

[7] Kenneth S. Kundert and Olaf Zinke. *The Designer’s Guide to Verilog-AMS*. Kluwer Academic Publisher, 2004.

[8] Ken Kundert. Principles of top-down mixed-signal design. *www.designers-guide.org/Design*.

[9] H. Chang, L. Cooke, M. Hunt, G. Martin, A. McNelly, L. Todd, *Surviving the SoC Revolution*, Kluwer Academic Publishers, 1999.C. Force, T. Austin. Testing the design: the evolution of test simulation. *International Test Conference*, Washington 1998.

[10] VSIA Analog/Mixed-Signal Development Working Group. *Analog/Mixed-Signal VSI Extension Specification Version 1 2.2 (AMS 1.2.2)*. *http://www.vsi.org*, Februrary, 2001.

[11] B. Bailey, G. Martin, T. Anderson, *Taxonomies for the Development and Verification of Digital Systems*, Springer Science + Business Media, 2005.

[12] Andreas S. Meyer. *Principles of Functional Verification*. Elsevier Scientific, 2003.

[13] J. Holmes, F. James, and I. Getreu. Mixed-Signal Modeling for ICs. *Integrated System Design Magazine*, June 1997.

[14] R. B. Staszewski, C. Fernando, and P. T. Balsara. Event-driven simulation and modeling of phase noise an RF oscillator. *IEEE Transactions on Circuits and Systems — I: Regular Papers*, vol. 52, no. 4, April 2005, pp. 723-733.

[15] K. Muhammad, T. Murphy, R. B. Staszewski. Verification of RF SoCs: RF, analog, baseband and software. *IEEE Radio Frequency Integrated Circuits (RFIC) Symposium*, June 11-13, 2006, pp. 361-364.

[16] T. Murayama, Y. Gendai. A top-down mixed-signal design methodology using a mixed-signal simulator and analog HDL. *Proceedings EURO-DAC’96, The European Design Automation Conference and Exhibition*, 1996, pp. 59-64.

[17] Open Verilog International. *Verilog-AMS Language Reference Manual: Analog & Mixed-Signal Extensions to Verilog HDL*, version 2.0, *www.eda.org/verilog-ams*.

[18] IEEE. *Standard Description Language Based on the Verilog^{TM} Hardware Description Language,* IEEE Standard 1364-1995.

[19] E. Christen, K. Bakalar. VHDL-AMS — a hardware description language for analog and mixed-signal applications. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 46, no. 10, Oct. 1999, pp. 1263-1272.

[20] Peter J. Ashenden, Gregory D. Peterson, Darrell A. Teegarden. *The System Designer's Guide to VHDL-AMS*. Morgan Kaufmann, 2002.

[21] IEEE. *VHDL Language Reference Manual,* IEEE Standard 1076-1993.

[22] IEEE. *IEEE Standard for System Verilog: Unified Hardware Design, Specification and Verification Language*, IEEE Standard 1800-2005.

[23] Open SystemC Initiative, *SystemC 2.1 Language Reference Manual*, www.systemc.org, May, 2005.

[24] Mathworks. Matlab and Simulink, *www.mathworks.com*.

[25] N. Chandra, and G. W. Roberts. Top-down analog design methodology using Matlab and Simulink. *Proceedings of the 2001 IEEE International Symposium on Circuits and Systems* (ISCAS'01), vol. 5, 2001, pp. 319-322.

[26] E. Lee. *Overview of the Ptolemy Project*. Technical Memorandum No. UCB/ERL M03/25, University of California, Berkeley, July 2, 2003.

[27] G. G. E. Gielen. Modeling and analysis techniques for system-level architectural design of telecom front-ends. *IEEE Transactions on Microwave Theory and Techniques*, vol. 50, no. 1, part 2, Jan. 2002, pp. 360-368.

[28] G. G. E. Gielen. System-level design tools for RF communication ICs. URSI International Symposium on Signals, Systems, and Electronics (ISSSE'98), 1998, pp. 422-426.

[29] C. Force. Integrating design and test using new tools and techniques. *Integrated System Design*, February 1999.

[30] C. Force, T. Austin. Testing the design: the evolution of test simulation. *International Test Conference*, Washington 1998.

[31] Teradyne. SpectreVX and SaberVX virtual test environments, *www.teradyne.com*.

[32] H. Chang, E. Charbon, U. Choudhury, A. Demir, E. Felt, E. Liu, E. Malavasi, A. Sangiovanni-Vincentelli, I. Vassiliou, *A Top-Down, Constraint-Driven Design Methodology for Analog Integrated Circuits*, Kluwer Academic Publishers, 1997.

[33] K. Kundert, H. Chang, D. Jefferies, G. Lamant, E. Malavasi, and F. Sendig. Design of mixed-signal systems-on-a-chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 12, Dec. 2000, pp. 1561-1571.

[34] E. Chou and B. Sheu. Nanometer mixed-signal system-on-a-chip design. *IEEE Circuits and Devices Magazine*, vol. 18, no. 4, July 2002, pp. 7-17.

[35] J. E. Franca. Integrated circuit teaching through top-down design. *IEEE Transactions on Education*, vol. 37, no. 4, Nov. 1994, pp. 351-357.

[36] Cadence Design Systems. AMS Designer, *www.cadence.com*.

[37] Mentor Graphics. ADVance MS simulator, *www.mentor.com*.

About The Authors

Ken Kundert. Ken is an IEEE Fellow and co-founded Designer’s Guide Consulting in 2005. From 1989 to 2005, Ken worked at Cadence Design Systems as a Fellow. Ken created Spectre and was the principal architect of the Spectre circuit simulation family. As such, he has led the development of Spectre, SpectreHDL, and SpectreRF. He also played a key role in the development of Cadence's AMS Designer and made substantial contributions to both the Verilog-AMS and VHDL-AMS languages. While in school he authored *Sparse*, an industry standard sparse linear equation solver and created Agilent’s harmonic balance simulator. Before that Ken was a circuit designer at Tektronix and Hewlett-Packard, and contributed to the design of the HP 8510 microwave network analyzer. He has written three books on circuit simulation: *The Designer's Guide to Verilog-AMS* in 2004, *The Designer's Guide to SPICE and Spectre* in 1995, and *Steady-State Methods for Simulating Analog and Microwave Circuits* in 1990; and created *The Designer's Guide Community* website. He has also authored eleven patents and over two-dozen papers published in refereed conferences and journals.

Ken received his Ph. D. in Electrical Engineering from the University of California at Berkeley in 1989, his M. Eng. degree in 1983 and his B. S. in 1979. Ken was elevated to the status of *IEEE Fellow* in January 2007 for contributions to simulation and modeling of analog, RF, and mixed-signal circuits.

Henry Chang. Henry co-founded Designer’s Guide Consulting in 2005. From 1995 to 2005, Henry worked at Cadence Design Systems, Inc. in research and development, methodology services, product marketing, corporate strategy, and in the office of the Chief Technology Officer. He has also worked at Micro Linear and GE Lighting. He is the author of three books: *Winning the SoC Revolution: Experiences in Real Design* in 2003, *Surviving the SoC Revolution: A Guide to Platform Based Design* in 1999, and *A Top-Down, Constraint-Driven Design Methodology for Analog Integrated Circuits* in 1997. He is on the steering committee of the IEEE Custom Integrated Circuits Conference, serving presently as the general chairman. He holds 10 US patents, has authored 14 technical papers, and has participated at conferences giving tutorials, sitting on panels, and giving keynote addresses.

Henry received his Ph. D. and M. S. in Electrical Engineering from the University of California at Berkeley in 1994 and 1992 respectively. He received his Sc. B. degree in Electrical Engineering from Brown University in 1989.

About Designer’s Guide Consulting

We help design teams overcome difficult verification challenges. Those challenges can involve either functional or performance verification. A difficult functional verification problem might be assuring that a mixed-signal circuit consisting of tens or hundreds of thousands of transistors with multiple $\Delta\Sigma$ converters operates correctly in each of its thousands of distinct operating modes. A difficult performance verification might be assuring that a large behaviorally complex circuit meets some demanding specification, such as a SerDes operating with a bit-error rate of less than 10^{-18}.

Designer’s Guide Consulting
101 First Street, #150
Los Altos, CA 94022
+1 650-968-8291
consulting@designers-guide.com
www.designers-guide.com