Verification of Complex Analog Integrated Circuits

Ken Kundert
Henry Chang

Version 1a, 1 March 2005

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. This paper describes a verification methodology to address these challenges.

The is paper was presented at the 2006 Custom Integrated Circuits Conference in September 2006. It was last updated on September 26, 2006. You can find the most recent version at www.designers-guide.com. Contact the authors via e-mail at consulting@designersguide.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. 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, providing this at the lowest cost and power possible, and finishing before the competition. A fundamental change is occurring that is driving failure in A/RF designs. There is an explosion in functional complexity that results in existing ad hoc verification methodologies becoming overwhelmed.

Complexity is exploding in A/RF design in multiple dimensions simultaneously. The primary culprit for the complexity explosion is that the number of *modes of operation* of today's A/RF designs is increasing due to several factors. First, there is a need to support a large number new and legacy worldwide standards and each standard then has their own unique modes of operation. To increase battery life, all of today's ICs have various power saving modes. And A/RF designs themselves have different circuit operating modes to perform their basic functions. A simple analog block such as an Analog-to-Digital Converter (ADC) or a Phase-Locked Loop (PLL) can have 10 to 30 digital control pins while an analog functional unit such as a power management unit, RF transceivers, or coder/decoder (CO/DEC) could have hundreds of control pins.

Increasing performance requirements on the latest processes where the components themselves are dropping in quality is forcing designers to increase the size and behavioral complexity of their circuits to overcome the limitations of the components. For example, they may incorporate self-calibration or error cancellation schemes; or they may choose to use algorithmic sampled-data circuits rather than simpler continuous-time circuits because they are a better match to the features of advanced CMOS processes [1,2,6,8,9]. Finally, advanced processes make many more transistors available that are used to add new features or increase the level of integration.

Each of these things individually (number of operating modes, behavioral complexity, and circuit size) contribute to making the process of verifying a design substantially more difficult, but all three are at work simultaneously and combine to overwhelm all existing approaches at verification. Functional failures in A/RF are now taking precedent over failures stemming from not meeting performance specifications. Functional failures can come in the form of inoperable modes and are often due to "simple" errors such as inverted signals, swapped bit lines, and

incorrect power up sequencing. Functional failures tend to result in failed chips that are show stoppers as the end customer cannot begin bringing up the firmware that is typically run on the SoCs. In contrast, 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.

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. The question on the minds of design managers is no longer whether there is a error in the design or not, but rather how many there are and how many spins it will take to find them. In digital design, verification separated from design and today dominates the majority of the engineering effort. A similar process must now occur for analog designs.

This paper describes a systematic A/RF functional verification methodology and provides a starting point for the A/RF design team and verification engineer.

2 Existing Approaches

Many approaches and combinations of approaches are used today to address the complexity and teamwork challenges.

2.1 Design Approaches

Understanding that verification is a daunting challenge with no easy solution, designers try 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. And in many cases, although conceptually the design is simpler, the number of control bits actually increase, thus leaving open the possibility for more design bugs.

Also, knowing the limitations of the design tools, simulators, and other constraints, designers often limit their design choices to avoid potential problems. In particular, without a strong verification methodology, promising yet complex architectures may be avoided.

2.2 CAD Approaches

Almost all approaches center on the idea of accelerating simulation. They include mixed-mode simulation (mixing transistor and register-transfer level (RTL) descriptions); mixed-level simulation (mixing functional models and transistors); and using fast simulators that exploit the characteristics of the circuit, such as timing simulators and RF simulators. These approaches all require additional effort and engineering trade-off decisions on the part of the A/RF designer — usually accuracy vs. speed. Although these approaches can increase simulation speed, without a systematic verification process to provide context for the simulation, it is difficult to correlate additional simulation speed to how many bugs remain in the design.

3 Verification Methodology

The goal of the verification methodology is to systematically find errors in A/RF designs in a reproducible manner. It also provides several substantial collateral benefits. It tends to make the design process itself more efficient and predictable [4,5]. It produces verified models and testbenches of the design that aid in reuse, integration, and test program development. It can be used to accelerate performance verification. And it tends to uncover errors in the specification documentation.

3.1 Key Concepts

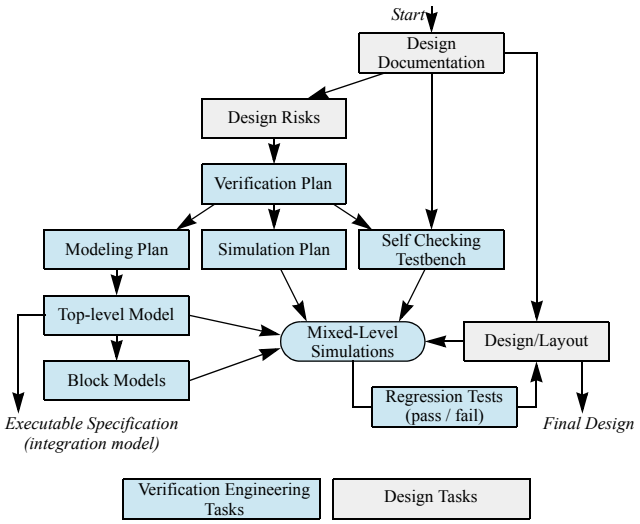
3.1.1 Verification Plan

An A/RF verification flow is illustrated in Figure 1. The verification flow begins with the *verification plan*. Design risks are identified by the designers. Verification strategies are developed to mitigate those risks. The verification plan lists the identified issues and presents a brief description of how verification is to be done.

3.1.2 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

FIGURE 1 *The A/RF verification flow.*



slow for the hundreds to thousands of tests needed to cover all the operating modes. At the heart of this methodology is a technique called *mixed-level simulation* [4,5] 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 currents or digital logic will be left at AHDL or HDL respectively. In this way, simulations can run hundreds to thousands of times faster. In this paper, we use Verilog-AMS [4] for our AHDL and Verilog for our HDL.

3.1.3 Modeling and Simulation 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 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.

3.1.4 Self-Checking Testbench

Full benefit from this methodology is derived with applying a *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. 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. 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 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 3.3. 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 key reason for choosing the self checking approach is to enable automated *regression testing*. Tests run automatically on a nightly or on an 'as needed' 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.

3.1.5 Verification Engineers

In Figure 1, note that the design and verification tasks are separated. To execute the verification tasks, a new engineer is required, 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 analog design concepts and techniques, terminology, and the use of analog design tools.

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 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.

Although this is an extra resource, the benefits can offset this burden. It is a means to train new analog designers. Also, designers can spend less time on functional verification. Instead they can focus on performance validation and on innovating new and more promising architectures. Finally, the overall efficiency of the design can be improved as new skills are being brought to the design team, skills that designers often lack.

3.1.6 Benefits

The key benefits of this methodology are:

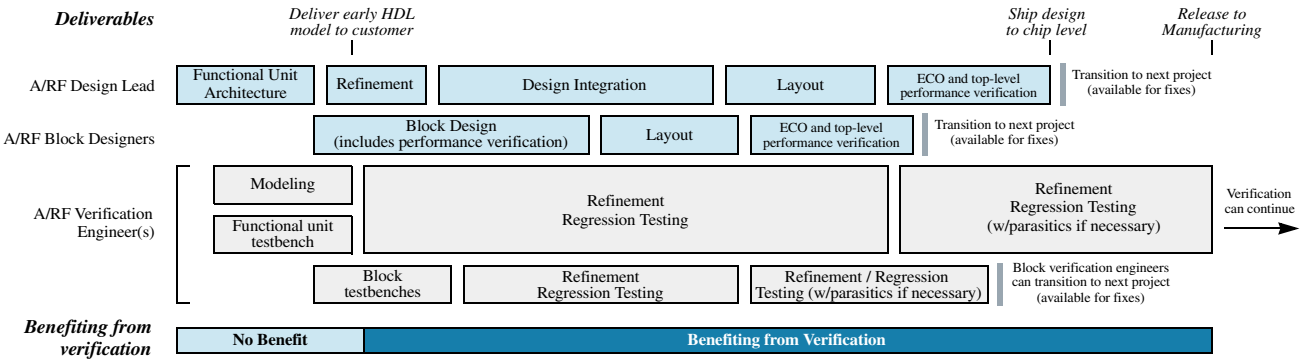
- Designers are always simulating their blocks in the context of the overall functional unit
- Extensive regression tests are always being run to make sure that interfaces are correct
- The design documentation, the HDL model and the design are always synchronized.
- Design time can be accelerated since errors are caught earlier.

3.2 Verification in Context of Design

The ideal design and verification project time line is shown in Figure 2. 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.

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, Ptolomy, 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

FIGURE 2 *Ideal A/RF design and verification project time line.*



parameters. 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 focus. 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 the design 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.

Besides developing and running tests, the verification engineers also develops and tests 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 the time to first customer shipment and allows them to contribute suggestions that would make the design more testable.

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.

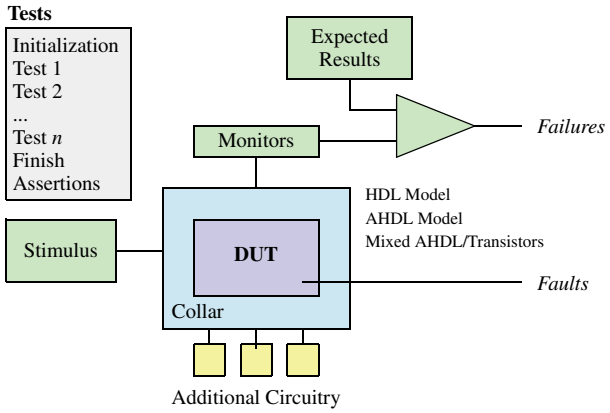
Maximum benefit is derived from this methodology if the verification engineer is brought into the project early as shown in Figure 2.

3.3 The Testbench

Apart from the *device-under-test* (DUT), the testbench (shown in Figure 3) contains all of the necessary components for verification. It is the responsibility of the lead verification engineer to assemble the 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 primarily using *assertions* and *tests* [7].

FIGURE 3 *Elements of the self-checking testbench.*



3.3.1 Assertions

Assertions have the property that they are always checked, regardless of what tests are running. An *assertion* is modeling code that continually observes one or more signals and raises a *fault* when it detects an error condition. 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, an assertion may monitor a bias line and an enable line to verify that the current through 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. Assertions can be placed in the model or circuits that are to be delivered to an end customer. to ensure that the circuit is properly interfaced to the remainder of the system and is being used properly.

3.3.2 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 be needed 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.

3.3.3 Transactors

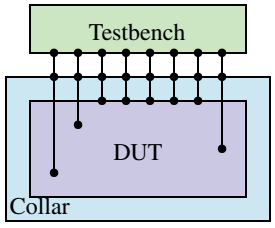
To enable re-use of tests within the same design or between designs, tests can be written in an abstract manner by employing a transaction-based interface to the DUT. A *transaction* is a sequence of signal transitions on a group of one or more pins that accomplishes a single task. A *transactor* [7] 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.

For example, a bit-error rate (BER) test that sends a random sequence of symbols to a transmitter and receives the sequence that has been processed by the DUT and looks for errors could be adapted to any number of different types of systems with the right transactors. For example, transactors that convert to and from an 8b/10b code could be used to apply the test to a SerDes. Other transactors could be used to apply the test to a wireless transceiver, data converters, or a wide variety of signal transmission systems. Similarly, these same transactors can be use as interfaces for different tests. For example, to stress the DUT 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 if the interface between the transactors and the tests are designed with an eye towards reusability.

3.3.4 Collars

A *collar* surrounds the DUT as shown in Figure 4. 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 would be created and go undetected if there are multiple versions of the collar than if there are multiple versions of the testbench.

FIGURE 4 *The testbench connects to the DUT through a collar, which is responsible for mapping signals internal to the DUT.*



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. For example, a typical adjustment is for the common-mode voltage at the output of one 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. This level of the hierarchy likely does not exist in the high-level Verilog model. 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. For the DUT models to behave properly, Verilog models have to have analog inputs and outputs. Fortunately, Verilog does support real variables. In this case, the function of the collar is to allow external blocks to read and write values into variables associated with these inputs and outputs.

3.3.5 Drivers and Loads

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.

The testbench represents a powerful first step toward design for re-use in analog. Unlike the transistor level design which will certainly change, the testbenches 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.

3.4 Writing Models, Testbenches, Regression Tests

One of the seemingly most daunting tasks is writing the models. This is not nearly as challenging and unbounded as generally perceived *if* the 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 in 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 function-

ality of the circuit. Use the verification and modeling plan as a guide to determine the level of detail required.

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.

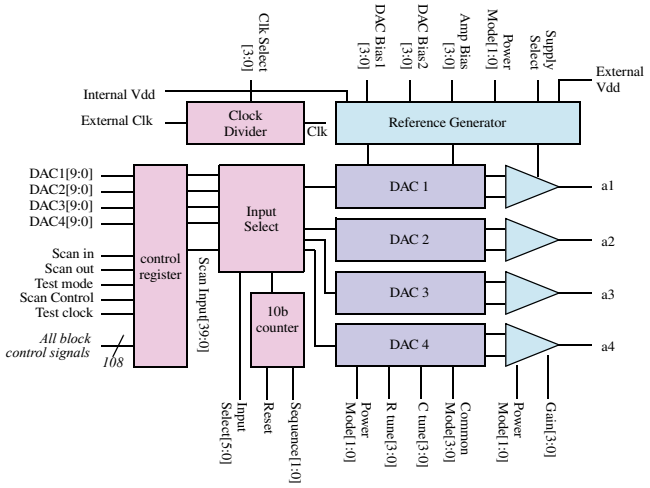
LISTING 1 *Verilog-AMS functional model of a flash ADC.*

```
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
```

4 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 5. 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 and any errors in this logic could not be corrected in software.

FIGURE 5 *Example A/RF functional unit.*



This design has 113 control pins, 40 signal inputs and 4 signal outputs. Therefore, there are 2^{113} possible control combinations with 2^{40} input combinations. Of course, most of the modes are independent and not all combinations have to be validated. This is why the verification plan is critical.

Finally, 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.

4.1 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
- 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 AHDL 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
- AHDL 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 1. The configurations show what 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 are 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.

TABLE 1 *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.

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.

Running all these configurations with the same test bench assures that they are all consistent in their behavior.

4.2 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 unit's 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.

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
    common_mode_voltage = V(dut_collar.cm_voltage);
    if (common_mode_voltage ...) ...// test the result
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
```

5 Conclusion

Applying verification in A/RF design allows 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. Testbenches written by domain experts 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.

This methodology is presented in considerably more detail in a paper that is expected to be published in the *Proceedings of the IEEE* in early 2007 [3].

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) then
            $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
```

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] Henry Chang and Ken Kundert. The rise of analog verification. Submitted to *The Proceedings of the IEEE* to be published in early 2007.

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

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

[6] 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.

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

[8] 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.

[9] 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.

About The Authors

Ken Kundert. Ken has been at Designer's Guide Consulting since 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.S. degree in 1983 and his B.S. in 1979.

Henry Chang. Henry has been at Designer's Guide Consulting since 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 ΔΣ 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