Coverage Driven Verification for Mixed Signal Systems

Cadence Design Systems
Walter Hartong, Nils Luetke-Steinhorst,
Hannes Froehlich

cadence™

Presented at

cadence designer network

cdn LIVE™

Silicon Valley 2007

Coverage Driven Verification for Mixed Signal Systems

Walter Hartong, Nils Luetke-Steinhorst, Hannes Froehlich

Cadence Design Systems,
Munich, Germany
{hartong, nls, hannes}@cadence.com

Abstract

Even though analog parts of today's mixed-signal chips are relatively small in terms of area and complexity, the design and verification effort for those parts is increasing. Moreover, chips operate in complex and mostly analog environments that must be taken into account in many cases. At the same time, reliability demand is zero-defect, e.g. for automotive components. Advanced verification methods like coverage driven verification are currently used to address the growing verification problem in the digital domain. However, these techniques are not yet addressing the problem of integrated analog parts in the digital context. This paper describes the use of coverage driven verification for analog/mixed-signal systems. The first part contains a general discussion of the digital verification method as of today and the possible mapping to an analog/mixed-signal design scenario. The second part describes a demo example that shows the coverage driven verification approach for a true mixed-signal/mixed domain system. The system contains electrical as well as mechanical and software parts and is therefore a good representation of today's complex system designs.

1 Introduction

Significant efforts have been made over the past couple of years to improve the quality and the productivity of functional digital verification. Only few years ago engineers wrote directed testcases which were composed of simple sequence of '0' and '1' as input stimulus for the design. In addition checking was done manually. It was part of the test and verification engineer's task to explicitly predict the expected response of the DUT for a specific test. In some cases it required even visual inspection of a waveform. Since this extra work had to be repeated for each and very testcase in case of a design change, engineers tended to minimize the amount of hard coded checks within their tests.

Advanced verification techniques have been developed and introduced into today's digital design flows to overcome most of those limitations and productivity restrictions. Moreover, the prediction of verification quality is a major improvement in the state of the art verification methods. Main components of these verification techniques are:

- Automated Stimulus Generation
- Automated Self-Checking (Assertions, Reference Models, etc.)
- Automated Coverage Measurements and Tracking

The status of analog/mixed-signal verification as of today is very similar to the scenario described above. Given that, the obvious question arises: Is it possible to improve verification efficiency and quality using the same or similar measures as used in the digital world?

It will be shown below, that there is no simple answer to that question. There is definitely room for improvement in the analog/mixed-signal verification strategy and a lot can be learned and adopted from the advanced methods described above. However, there are also some fundamental differences in the design and verification problem that lead to different requirements and implementations of the verification solution.

This contribution is structured as follows: Section 2 elaborates the verification process and the differences of analog and digital approaches in general. In Section 3 the use of those approaches inside Specman and AMS Designer will be highlighted. An automotive design example will be presented in Section 4 and implementation details and results will be discussed. Finally, Section 5 discusses advantages and disadvantages of this approaches and points out other possibilities.

2 Verification Goals and Approaches

Verification has basically two goals (Figure 1): Firstly, to check that the system fulfills the specification. That means that it does what it is required/specified to do and does not exceed certain limits, e.g. the gain of an amplifier must be above 10db. Secondly, it has to be verified that the system does not do anything "bad" that might have a negative influence on other components or the environment. E.g. the amplifier starts oscillating during power up phase. This second category covers mostly implicit assumption that are naturally made but not explicitly specified.

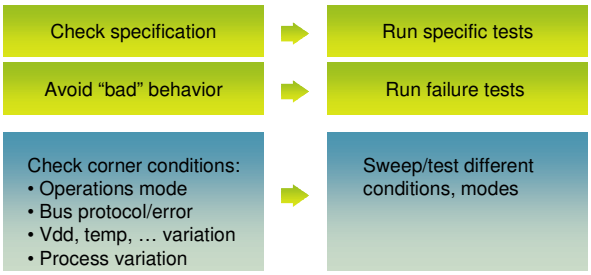

Figure 1: Verification goal and methods.

Specific simulation runs and tests are required to implement the verification of those two categories above. Moreover, it has to be assured that the verification goals are met in all different operating modes and corners of the system, such as:

- System's operation modes
- Different phases of the system operation
- Silicon process variations and device mismatch
- Varying environment parameters, like temperature, supply voltage

To ensure this, the tests have to be repeated in those different constellations. It is already obvious that an exhaustive search through the whole space of different tests, operation modes and corners might be impossible to do. Trade-offs have to be made between the verification effort and the level of confidence in the correct behavior of the system.

2.1 Verification Methodology

As mentioned above, verification is based on simulation runs. The design under test (DUT) is stimulated with some input data and simulation is run for a specific time, frequency range, amount of data points etc. Finally, the simulation results are stored. Beside the simulation setup itself that should not be considered here, there are two important tasks:

- Generation of input stimuli
- Checking the results against expectations

All above applies for analog as well as for digital verification and is also valid for advanced automatic and manual methods. The third task mentioned in Figure 2 is functional coverage to measure which verification goals have been achieved during simulation.

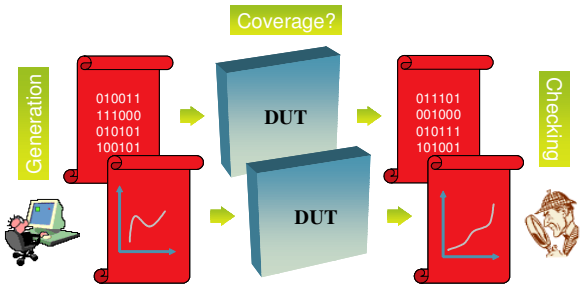

Figure 2: Generation, Checking, and Coverage.

2.1.1 Checking

The first step for verification automation is the automation of results inspection. Only if the task is formalized and implemented, further automation can be applied.

As said above, main parts of analog circuit verification still rely on manual waveform inspections. There are several reasons for that:

- Necessary measurements and calculations on analog waveforms are sometimes hard to implement.
- Waveform inspection is a major part of the analog workflow. People are used to do that.
- Analog design is not very formalized and still relies on expert knowledge. This implies that there are significant amount of implicit assumptions beside the specification.

Even though a complete replacement of manual waveform inspection in analog design is not realistic today, a lot of results checking tasks can be relatively easy automated. That enables automatic verification, e.g. regression runs, on those automatic checks, while the analog designer can focus on the non-formalized checks.

2.1.2 Generation

The input stimuli generation can be either manual or automatic. The straight forward way is the manual method, where the designer or verification engineer defines the input stimuli for a specific test.

The automatic method requires a certain amount of freedom for the algorithm to create the input signals. In other words, the user has to define certain constraints for the inputs and the generator creates – in most cases randomly – the stimuli within the given limits. Section 2.1.4 will discuss the use of random stimuli generation in more detail.

It should be noticed that the checking task is also influenced by the generation method. In case of a manually created fixed stimulus it is sufficient to verify the output against a well defined and fixed output behavior. However, if the input stimulus is varying, the expected output has to be defined according to the input.

There are two ways to address this problem. If the check is measuring derived values, e.g. amplifier gain, those performance numbers may be – within certain limits – independent of the input stimulus itself, thus, those types of checks are applicable with automatic stimuli generation. Secondly, if the check is implemented to compare the output against a reference, this reference value has to be generated according to the input stimulus. This could be achieved by the use of a reference behavioral model that is used as executable specification.

2.1.3 Coverage

Theoretically, functional coverage is defined as the ratio between the visited or verified states of the systems state space, divided by the total amount of states. This definition can be easily understood for finite state machines. Figure 3 shows a little example on the top where states A and B have been verified, while the whole state space has 4 states: A, B, C, D. Thus, coverage is 50% in this case.

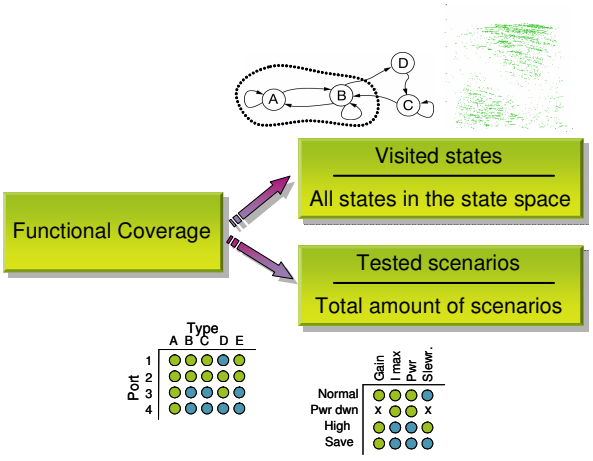

Figure 3: Functional Coverage in Digital and Analog.

For complex digital systems, the state space can be very big, e.g. the exploding number of states in a memory block. For analog circuits the situation is even worse, since the state space is a continuous vector space. See the little vector space of a tunnel diode on the top right corner of Figure 3. Still the theoretical definition holds for those systems, however the practical use is very limited.

It is common praxis not to take the whole state space into account but to define the verification goals and possible scenarios to be covered by the user. E.g. the verification goal might be to check 4 ports of a system applied with 5 different types of data, A-E (bottom left picture). The functional coverage is defined as the ratio of the verified scenarios divided by the total amount of scenarios – 20 in this case. A similar measure can easily be applied to analog systems (see bottom right picture).

Even though this definition is of more practical use, it is likely that the amount of desired verification scenarios is extremely large. E.g. it might be the goal to check all combinations of the 5 data types applied to the 4 ports (AAAA, AAAB, AAAC, …) including their transitions (AAAA followed by AAAB, etc.) in the example above. The result is $(5^4)^2 = 390625$ scenarios. As a result a simple for-loop checking all scenarios one by one is not applicable.

2.1.4 Coverage driven verification flow

Given that an exhaustive search through the whole verification space is not practical implies that the verification process is limited in time. However, the coverage figure still provides an accurate number of the verification quality with respect to the defined goal.

Figure 4 assembles the pieces together that have been discussed above. The simulation results are automatically checked and problems are being reported. An automatic stimuli generator creates tests on a random basis within given constraints. On top, the designer might have a certain amount of tests that are pre-defined and need to be run (directed tests) to reach certain corner cases.

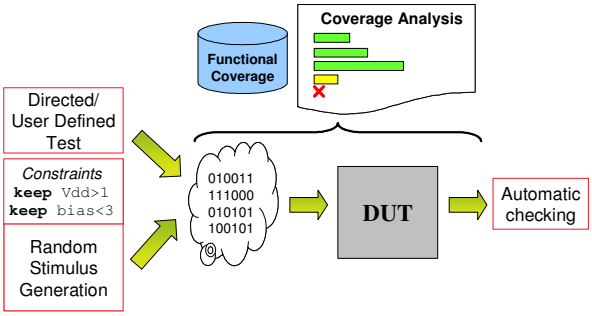

Figure 4: Coverage driven verification flow.

The verification process, as described without coverage measure, can run independently for the given amount of time. However, it is unclear how far/close the overall verification goal might be. Adding the coverage into that picture provides this information and shows what has already been achieved in the given timeframe. By this, the approach becomes

scalable, e.g. doing quick checks during design phases, running daily overnight regressions and a long final sign-off regression for the project.

3 Using Specman with AMS Designer

The described coverage driven verification approach is well known and supported by Incisive Enterprise Specman® and the verification language "e". Specman basically controls the simulator, generates input stimuli, performs the output checks, and summarizes the coverage measures.

Combining the Specman approach with the single kernel, mixed-signal and mixed-language simulator Cadence® Virtuoso® AMS Designer Simulator, which is a component of Virtuoso Multi-Mode, provides the capability to do mixed-signal coverage driven verification.

Indirect Analog Verification

The simplest way to achieve that is to include analog blocks into the digital design, where all checks and stimuli are applied to the digital part only. In this case, there is no need to change anything in the verification setup. Only the simulator is switched from pure digital NCSim to AMS Designer and the analog block is included (see Figure 5, middle block "A"). In this configuration, the analog behavior is stimulated and checked through the digital part of the system. This limits the amount of checking capabilities but is still a useful setup.

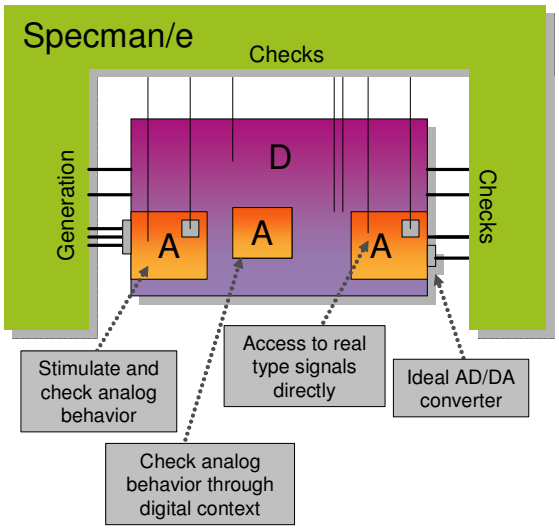

Figure 5: Mixed-signal verification setup.

Direct Analog Verification

The scenario where Specman interacts directly with the analog part is more complex but also more powerful (see Figure 5 left and right "A" block). Since the 6.1 version of Specman, the usage of real valued data is supported in e. Additionally, real values, voltages etc. can be directly probed inside analog blocks. For other probes like currents and driving analog nets, ideal AD/DA converters have to be added into the testbench. It should be noted that the driving and probing capabilities of Specman in the analog domain are constantly extended, so that the necessity of manual inserted converters will soon be obsolete.

4 Experimental Results

The methodology described above and the practical use of this approach should be illustrated using an automotive application. It is a window winder system with all it's components, including the mechanical chain and a software part.

The schematic of the top level design is shown in Figure 6. The following list provides a brief description of the blocks on the bottom row (from left to right).

- Micro controller. modeled as simple SystemC program.
- Motor control unit. Including transistor level, VerilogAMS and digital Verilog blocks.
- Motor modeled in VerilogAMS.
- Mechanical chain modeled in VHDLAMS.
- Rotation sensor and window model (VHDLAMS)

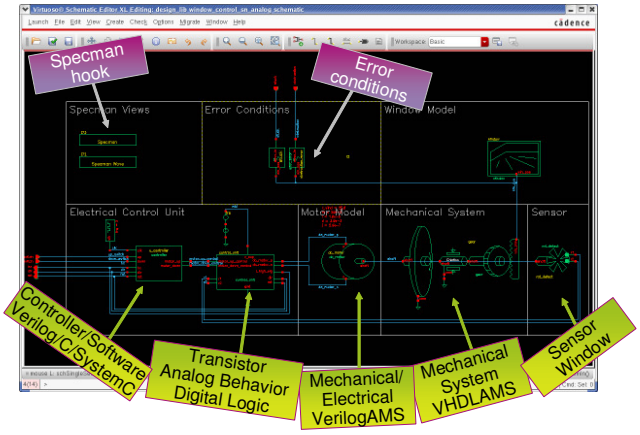

Figure 6: Window winder system.

The two blocks in the middle of the top row model two error conditions:

- Obstruction: Some object is blocking the window on moving up. The window can still move down.
- Stuck: the window is totally stuck, it can't move up or down.

Specman hooks into the design through the upper left blocks.

The micro controller block gets two input signals (up_switch and down_switch). It starts and stops the motor accordingly and controls the window position. To avoid motor damage the micro controller detects if the motor current is too high for a certain amount of time and switches off the motor. Finally, if the window does not fully close, even though the motor is on, the controller will switch the motor "down", assuming that someone might have tucked his arm out of the window. If that is not possible either it will fully switch off the motor.

4.1 Verification Approach

The system's input values are:

- Up switch
- Down switch
- Obstruction error
- Stuck error

The supply voltage of the system is varied randomly within certain limits inside the testbench.

To simplify matters, only 3 specifications values are checked:

- If no error condition is set and the up switch is set, the window must be closed after 100 clock cycles.
- If no error condition is set and the down switch is set, the window must be open after 100 clock cycles.
- The motor current shouldn't be too high for more than 10 clock cycles.

4.1.1 Generation

The e code for the generation part looks like follows (only essentials are shown). The four free input variables are all of type bit. The "keep" constraint avoid the illegal setting of up and down at the same time.

```
struct window_ctrl_s like
  any_sequence_item {

  up : bit;
  down : bit;
  obstruction : bit;
  stuck : bit;

  keep up == 1 => down == 0;
  keep down == 1 => up == 0;

  event item_driven;
};
```

For the checking itself various scenarios are defined. The first test in the code below is a directed test, since all the free parameters are fixed. Thus, as first test the window will move up with no error condition switched on. After 100 clock cycles the second test will be started.

The second test requires the two error conditions to be off, however, the decision on moving the window up or down is left open. It will be chosen by Specman on a random basis.

After those two directed or partly directed tests, the next 100 runs are completely unconstraint. The error condition and the up/down decisions are taken randomly.

```
extend MAIN window_ctrl_sequence {
  !item : window_ctrl_s;
  body() @driver.clock is only {

  do item keeping {
   .up == 1;
   .obstruction == 0;
   .stuck == 0;
  };
  wait[100];

  do item keeping {
   .obstruction == 0;
   .stuck == 0;
  };
  wait[100];

  for i from 1 to 100 {
      do item;
      wait[100];
  };

  };
};
```

4.1.2 Checking

The three specification values are checked in the e code below. The implementation is based on events. E.g. the event hc_too_long is triggered when the current was too high longer than 10 clock cycles. This event will then issue an error accordingly.

```
event hc_t is
 true(p_agent.smp.hc$==1)@clk;
event hc_too_long is
 {[10]* @hc_t;}@clk;

on hc_too_long {
dut_error("hc is high too long");
};
```

In the same way the correct behavior of the window open and close function is described. It should be noticed that the window_position signal is a real value that is directly probed inside the top level schematic.

```
event button_up is true(
 p_agent.smp.up_switch$.as_a(bool))
 @clk;
event win_should_be_up is
 {[100]*{(@button_up and
 not @stuck_on and
 not @obstruction_on)}
 }@clk;
on win_should_be_up {
 check that window_position >= 1.0
else
 dut_error(
 "windows should be up by now!!!");
};
```

4.1.3 Coverage

The four most obvious coverage items are the four input signals up, down, obstruction, and stuck. It must be ensured that each of these signals have be switched on and off once.

```
extend window_ctrl_s {
      cover item_driven is {
      item up;
      item down;
      item obstruction;
      item stuck;
```

As said above, the vdd voltage varies during simulation time between about 11.9 and 12.4 V. Covering those variations as well ensures the correct behavior of the system within all different supply voltage levels. Since vdd is a real signal we have to define lower and upper boundary as well as the precision we want to use for the coverage.

```
      item_rld vdd p_agent.smp.vdd$
       -from 11.9
       -to 12.4
       -precision 0.1;
```

So far all coverage items are independent, meaning that the occurrence of up=1 and up=0 provides a 100% coverage for this item regardless of the values of the other items. Crossing of different coverage items provides the needed correlation between the items.

The example below covers all scenarios with the up button pressed. This adds up to: 2 possibilities each for stuck and obstruction and 5 regions inside the vdd range, thus, 20 test scenarios.

```
cross up, stuck, obstruction,
 vdd_rl using ignore=(up==0);
```

4.2 Verification Results

Figure 7 shows the simulation results after a few minutes of verification time. The first four lines are the input signals, followed by the high current signal and the rotation signal generated by the sensor. The motor voltages are displayed as analog waveforms below. Finally, the window position and the vdd voltage is displayed.

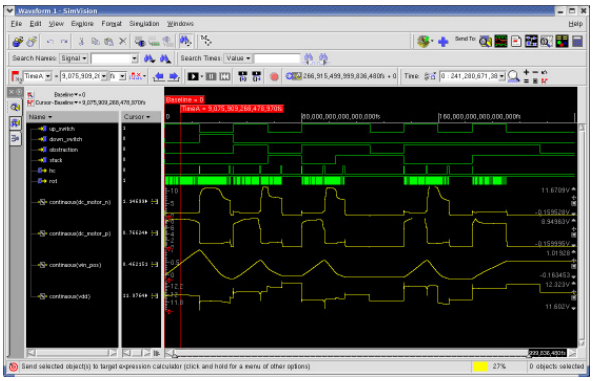

Figure 7: SimVision showing waveform results.

It is clearly visible that the two error conditions are off for the first two cycles and the window moves up in the first cycle, as defined in the generator code. The rest of the switching activity is chosen randomly by the generator.

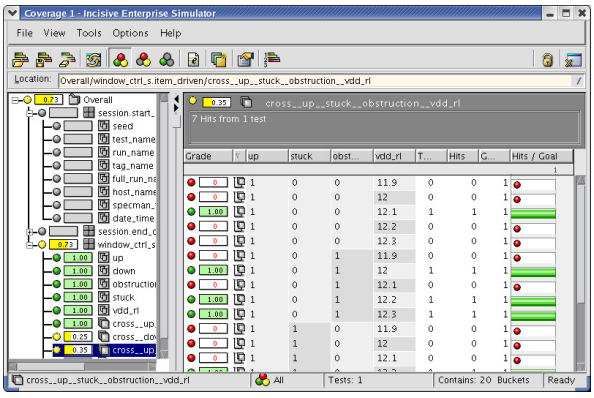

Figure 8: Coverage values after short simulation.

After the given simulation time, the coverage results look as in Figure 8. The first 5 coverage items are complete, meaning the coverage is 100%. However, the crossing discussed in the previous section has only reach coverage of 30%. The green bars indicate a coverage item that has been hit and the numbers of hits, while a red circle highlights items that have not been reached in current simulations.

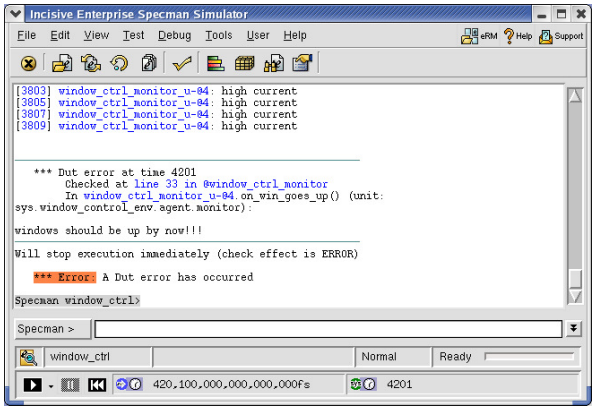

Figure 9: Design error found.

Figure 9 shows the output message of a problem found by the verification code. In this case the two error scenarios are switched off and the up-button is pressed. Still the window did not reach the uppermost position after 100 clock cycles (Figure 10)

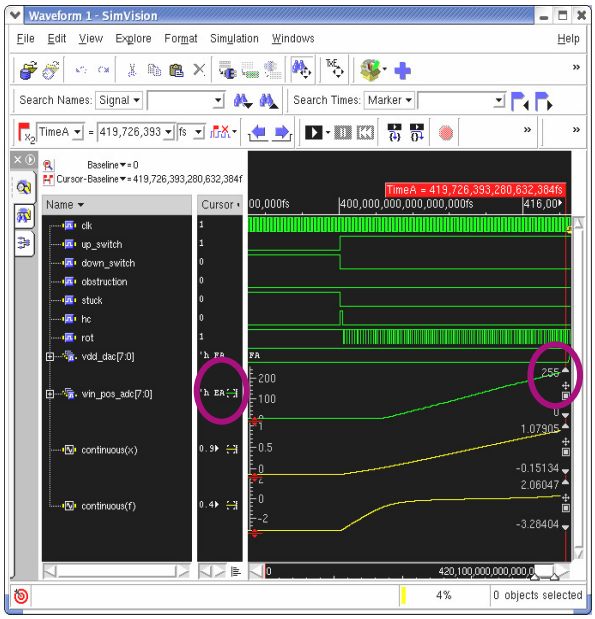

Figure 10: Simulation results showing the problem.

A detailed analysis shows that the window position went into the negative region before the up-button is pressed again. That is the reason for the failure. Even though possible, the designer of that demo system had not thought of this possibility and had never tested this scenario manually. This shows clearly the advantage of the random base generation approach on a relatively simple example.

5 Conclusions

As shown in the previous sections, coverage driven verification for mixed-signal designs is relatively easy to setup and run using Specman and AMS Designer. The advantages of the approach are obvious and have already been discussed before:

- Advanced verification method
- Well defined verification strategy
- Random generation
- Automatic checking
- Coverage matrix
- Natural integration into digital verification flow based on Specman/e
- Only little modification needed in the design

However, there are also some disadvantages that need to be taken into account:

- The integration into the analog centric flow that is mainly based on Analog Design Environment (ADE) is very limited today.
- Implementation effort for automatic checks of analog values ranges from simple to extremely complicated. Thus, a complete automatic checking requires high effort.
- The verification language e is unknown in the analog flow.

5.1 Simulation Performance

Simulation performance has to be considered as well in this context. It is well known that the simulation performance of a digital circuit is far better than a transistor level analog block. Thus, even the integration of a single analog block into a digital simulation environment can slow down simulation significantly.

There are several measures to improve the analog simulation performance, e.g. behavioral modeling, FastSpice solvers, etc. This matter is of high importance but should not be discussed in this context. However, there are several concurrent desires in verification that needs to be traded off:

- The system that is verified should be as complete as possible to catch problems that result from the interaction of different block (integration problems).
- Each individual component should be modeled on the highest level of accuracy.
- Simulation performance should be as high as possible to be able to verify many scenarios in short timeframe.

Obviously, individual tradeoffs have to be made for those trends, depending on the application and the current design phase.

5.2 Application scenarios

Given the pros and cons discussed above, leads to an application scenario that looks like follows:

Coverage driven verification on mixed-signal circuits does not – and does not intended to – solve the simulation performance problem. Thus, only system configuration that simulate in a reasonable timeframe enable advanced verification methods.

This approach is well suited to verify mixed-signal integration problems where the digital verification approach is already based on Specman/e. In those scenarios, a detailed check of the pure analog behavior is done in the classical Analog Design Environment (ADE) which is will known by the analog designer. Some selected checks are integrated into the Specman/e verification environment to ensure the correct behavior of the analog parts after integration in the digital environment.

Another application area is an analog centric design flow with straight forward analog performance checks and many digital control and input ports. In this configuration it is relatively easy to provide realistic digital stimuli through the verification environment and run random based verification. This provides a significant advantage over the manual verification approach done in the analog environment.

5.3 Alternative Approaches

As mentioned before, ADE is the standard environment for analog design. The new ADE XL environment as part of the Virtuoso Custom Design Platform IC 6.1, provides an analog centric verification suite, enabling automatic checking, multiple runs, corner and Monte Carlo simulation.

Key features are:

- Support for transient, DC, AC and RF analysis
- Specification-driven design
- Supports multiple tests/analysis & measurement inside a single ADE XL state
- Parasitic aware design flow
- Constraint-driven design
- Automatic characterization and model generation
- Integrated sizing and optimization capabilities
- Report generation
- History archive

The ability to automate the verification process combining a complete set of testbenches/tests/analyses into a single simulation run is a huge productivity benefit for the analog designer. Combining this with a complete set of measurement checking the specification values results in a simple pass/fail summary for each specification value. A rich set of predefined checking functions and a straight-forward scripting language (SKILL) allow users to implement circuit specific checks in a very efficient way. Moreover, the scripting language SKILL/OCEAN provides an easy and flexible way of automating the ADE XL based verification in batch mode.

6 Summary

A coverage driven verification approach for mixed-signal systems has been presented. In the first section the general verification task has been analyzed in detail and the similarities and differences between analog and digital verification are considered.

The practical implementation is based on Specman/e as verification environment and AMS Designer as mixed-signal simulator. An automotive example was used to present the experimental results. As mentioned, the implementation of analog checks and coverage is straight forward in e. Design modification are not necessary, except of the instantiation of the Specman blocks inside the top-level schematic.

The approach enables advanced verification methods for mixed-signal design and closes a gap in the increasing demand of system integration and reliability goals. However, this approach is clearly not a replacement but a complementary measure for pure analog verification efforts. Those analog focused tasks are easier performed in environments like ADE. The target applications are mixed-signal integration tasks and analog design scenarios with a significant amount of digital controls that are hardly handled in analog environments as of today.

Over the last 10-15 year the analog and digital working environments have been more and more separated form each other focusing on the core design challenges in the particular area. However, in the last couple of years, the opposite trend is clearly visible. More and more mixed-signal problems require a closer interaction between the two environments. This contribution clearly targets in that direction, but it seems to be only one of may starting points of a longer phase of more integrated analog and digital functionality.