Extending New Verification Techniques to Mixed-Signal SoCs with VCS AMS

Pierluigi Daglio, Mauro Scandiuzzo, Alessandro Valerio *
Helene Thibieroz **, Massimo Prando, Carlo Borromeo ***

* STMicroelectronics - Agrate Brianza (MB) - Italy
** Synopsys - Mountain View (CA) - USA
*** Synopsys - Agrate Brianza (MB) - Italy

ABSTRACT

Progress in mixed-signal System-on-Chip (SoC) designs is determined today by several factors including cost, performance and power consumption. To survive with these constraints, a mixed-signal verification solution must offer not only performance and flexibility but also state-of-the-art verification techniques with innovative methodologies. By coupling industry best-performance Fast SPICE engine (CustomSim) with native integration of advanced technologies in digital functional and low-power verification, Synopsys VCS AMS offers a unique mixed-signal advanced solution for the growing complexity of the mixed-signal SoCs.

In this paper, we present the usage of proven verification techniques for mixed-signal designs, coupled with capabilities for time efficient simulations such as CustomSim "Save & Restore" technology. We also introduce the usage of digital-like "Assertions" in the AMS domain as well as the concept of coverage extended to analog. Finally, we will briefly announce our future works about AMS verification.

Keywords: *VCS AMS, CustomSim, Save & Restore, Assertions, AMS verification, Fast SPICE*

Table of Contents

Table of Tables

Table of Figures

1 Introduction

System level verification is steadily becoming the most critical point in the design cycle of complex AMS applications. This is due to several reasons such as:

- The complexity of the functions incorporated in a device; such a complexity is continuously growing, affecting the number of simulations required before tape-out in order to guarantee effective verification coverage.
- The number of transistors per device, which is rapidly increasing while the technology node is getting smaller.
- The complexity of the model cards which reflects the physic of the transistors; in fact, previously negligible effects need now to be taken into account for smaller geometries to better mimic the behavior of the transistor.

Function complexity, number of transistors and complex model cards are certainly challenging both SPICE and FastSPICE simulators, forcing designers to search for compromises in order to get a trade-off between performance and accuracy. A well-deployed mixed-signal simulator solution allows the combination of analog blocks with digital blocks at either transistor and/or behavioral level. The presence of analog and digital blocks on the same test bench is driving toward new frontiers of verification as it is becoming necessary to mix two different worlds known as digital and analog verification. In addition to extend digital verification techniques (test bench, assertions, coverage, ...) to analog, Synopsys offers a solution which aims not only to speed-up the simulation performance, but also to provide designers with a complete flow that allows a safe and well-structured verification methodology.

VCS AMS provides fast mixed-level simulation engines that support multicore technology, guaranteeing transistor-level accuracy and higher throughput. Both analog-on-top (AoT) and digital-on-top (DoT) approaches are supported as well as multiple behavioral languages, including Verilog, VHDL, SystemVerilog, Verilog-A and Verilog-AMS. Other important and meaningful capabilities include:

- Automated insertion of interface elements to avoid convergence or accuracy issues
- User defined properties for selected interface elements
- Supply sensitive interface elements (widely used for power-up simulations)
- Diagnostic reports that enable early detection of design errors caused by complex inter-block connectivity or analog-to-digital interfaces
- Save & restore features to further boost performance and ease AMS verification for regression tests
- AMS Testbench and assertions which extend digital verification techniques to AMS designs providing fast debugging mechanisms and coverage for mixed-signal
- VCS Native Low Power (NLP) technology, enabling designers to carry their UPF power intent into the analog domain

The addition of these capabilities is an enabler for a better debug of the state-of-the-art devices, requiring a substantial effort in verification, due to the explosion of scenarios needed to validate the design.

2 Evolution of Mixed-Signal Verification

2.1 Analog-driven AMS scenarios

AMS verification requirements are changing year-by-year as these are driven by the increasing complexity of AMS applications. In the past, analog teams verified their macro-cells using a pure SPICE or Fast SPICE simulation while digital teams were running pure digital RTL simulation of VHDL or Verilog blocks. Going forward, the majority of system bugs were the ones related to the interface pins between analog and digital parts of the designs. For this reason, the need of a real mixed-signal simulation, taking into account, at the same time, transistor level portion and digital RTL portion, led to couple together those two kinds of simulators. Today in STMicroelectronics, we mainly face with two different verification scenarios: transistor level IPs and analog macro-cells and/or full chip AMS co-simulation. In the first case, the whole verification is performed at transistor level using SPICE or CustomSim (XA). The main things to check are the functionality of the circuit, the static and dynamic electrical rules (ERC) with Synopsys circuit check capability (CCK), and the safe operating area (SOA). We try to push as much as possible the usage of analog behavioral languages and above all Verilog-A which is supported by almost all the SPICE and Fast SPICE simulators, making your code portable without any particular difficulty. In the second case, the top-level verification is performed at analog/digital co-simulation level; digital test benches are used to drive the verification process because they are easier to manage with respect to analog ones and give the verification engineers more flexibility. The first requirement for such a methodology is a fast and reliable engine on the transistor level part, which is usually the bottleneck of the entire simulation. CustomSim (XA) from Synopsys delivers both speed and accuracy. The usage of behavioral languages both in analog and digital worlds is strictly mandatory to reach reasonable simulation time, therefore we try to cross-fertilize teams to increase the usage of Verilog-AMS together with VHDL, Verilog and SystemVerilog. In Figure 1 and Figure 2, you can see the two different AMS verification scenarios typical of STMicroelectronics analog-driven mixed-signal products.

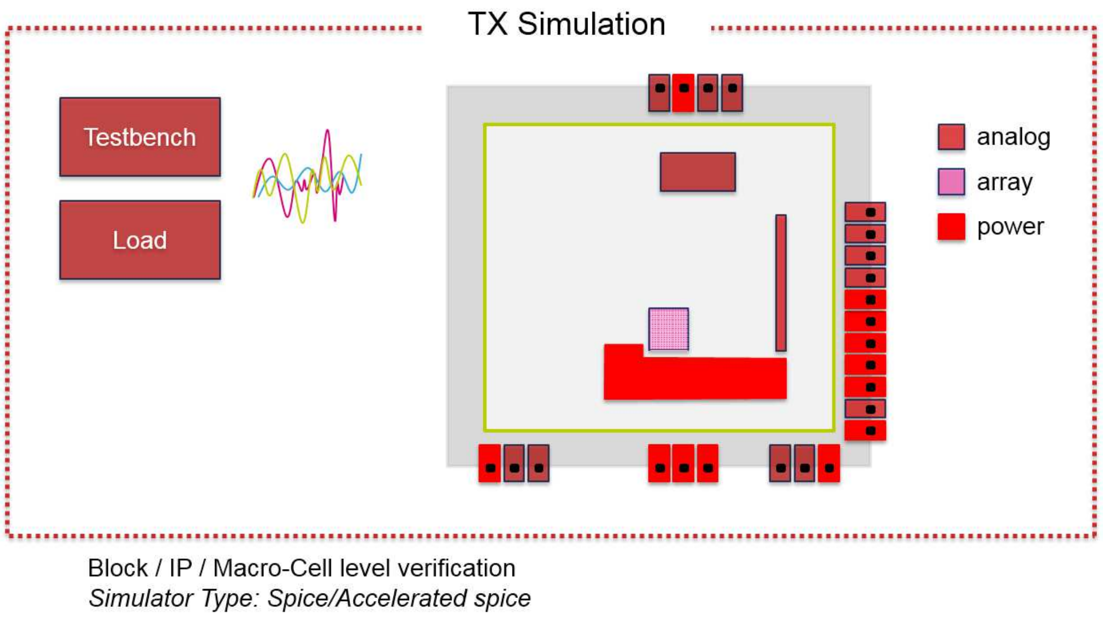

TX Simulation
Testbench
Load
analog
array
power
Block / IP / Macro-Cell level verification
Simulator Type: Spice/Accelerated spice

Figure 1 Block/IP/Macro-Cell Level Verification

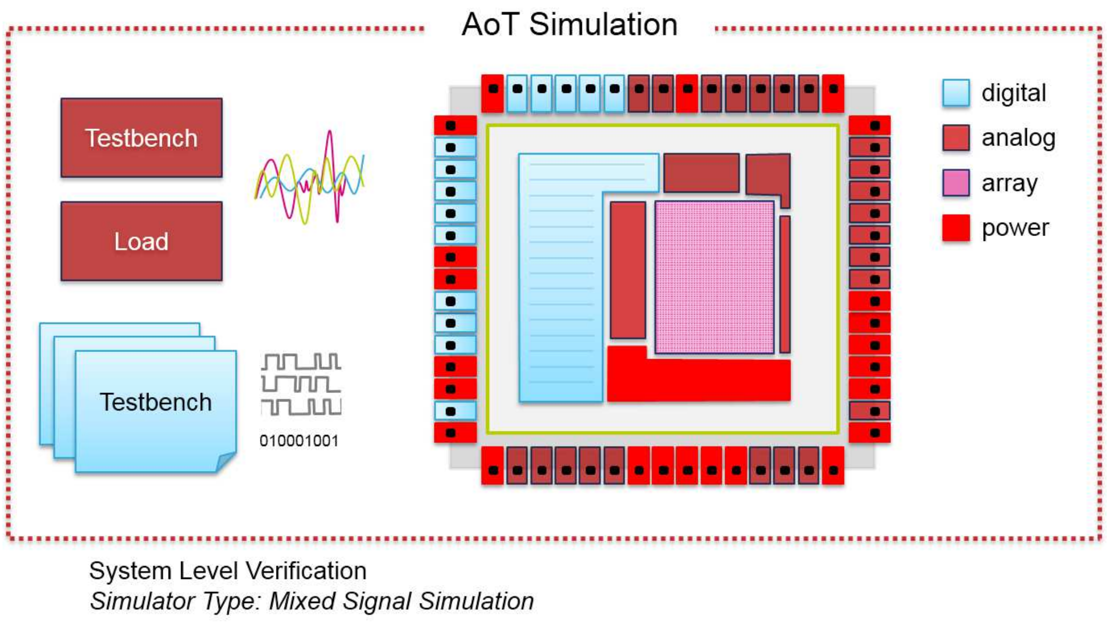


Figure 2 System Level Verification

2.2 Single-vendor and Multi-vendor Solutions

AMS verification solutions today can involve either a single-vendor or a multi-vendor mixed-signal simulation engine. In the second case, transistor level SPICE simulator belongs to one vendor whereas the digital simulator belongs to another vendor. This solution however presents some major limitations due to the fact that there is not a unique simulation kernel. In our case, we observed the following problems:

- Longer turnaround time for bug fixing and enhancement implementation
- Verilog-AMS configurations not fully supported

The single-vendor approach alleviates those problems as it is based on the same vendor solution and most often offers a single kernel that manages simultaneously the two engines. The main advantages of this solution are:

- Availability of save & restore capability
- Possibility to use Verilog-AMS extensively for simulation speed-up
- AMS assertions management to speed-up debug and increase coverage

The single vendor solution must offer a strong and reliable FastSPICE simulator, as the transistor level part simulation time is the bottleneck. For this reason we chose, as a solution for large system verification, Synopsys VCS AMS which assures impressive speed on the transistor level analog portion (CustomSim) as well as high capabilities in the digital simulator (VCS) including:

- Automated insertion of interface elements to better avoid convergence or accuracy issues due to the presence of *connect modules*, exploiting the capabilities of Verilog-AMS
- Diagnostic reports that enable early detection of design errors caused by complex inter-block connectivity or analog-to-digital interfaces

- Save & restore features to ease AMS verification for regression tests
- AMS Testbench which extends digital verification techniques to AMS designs
- VCS Native Low Power (NLP) technology, enabling designers to carry their UPF power intent into the analog domain

The target of a perfect mixed-signal co-simulation engine is to reach the accuracy of a pure SPICE simulator with, at the same time, a reasonable speed compared to the speed of a digital simulator. To achieve this target, we selected VCS AMS because it is fast and accurate with reliable and strong engines, as well as easy to setup. In addition, we also have a partnership and access to Synopsys R&D to improve the engine performance on our circuits and a regression test-suite at the vendor site to constantly test the environment. This insures a faster turn-around time for bug fixing and enhancement implementation, only one EDA vendor to interface, full support of Verilog-AMS capabilities, multi-scenario verification option to increase the coverage. Finally, if we compare in a table the multiple-vendor and Synopsys VCS AMS solutions we get the results that you can see in Table 1.

	CSIM-VPI MULTI-VENDOR	*CSIM-DKI MONO-VENDOR*
Simulation Speed	✓	✓
Flash Cell Model	✓	✓
Save & Restore	✗	✓
UPF/AMS features	✗	✓

Table 1 Multi-vendor and VCS AMS solutions comparison

3 AMS Design Flow

The classic AMS design process used in STMicroelectronics to simulate and to verify large IPs, macro-cells and big AMS systems is highlighted in Figure 3. The simulation and verification of AMS systems is done using a system-level analog/digital co-simulation with dedicated digital test benches, because this approach is definitely easier to manage. Therefore, CustomSim is being used, as we need fast and reliable simulation for the transistor level part. We also encourage the extensive usage of hardware description languages both for digital (Verilog and VHDL) and for analog blocks (Verilog-A and Verilog-AMS).

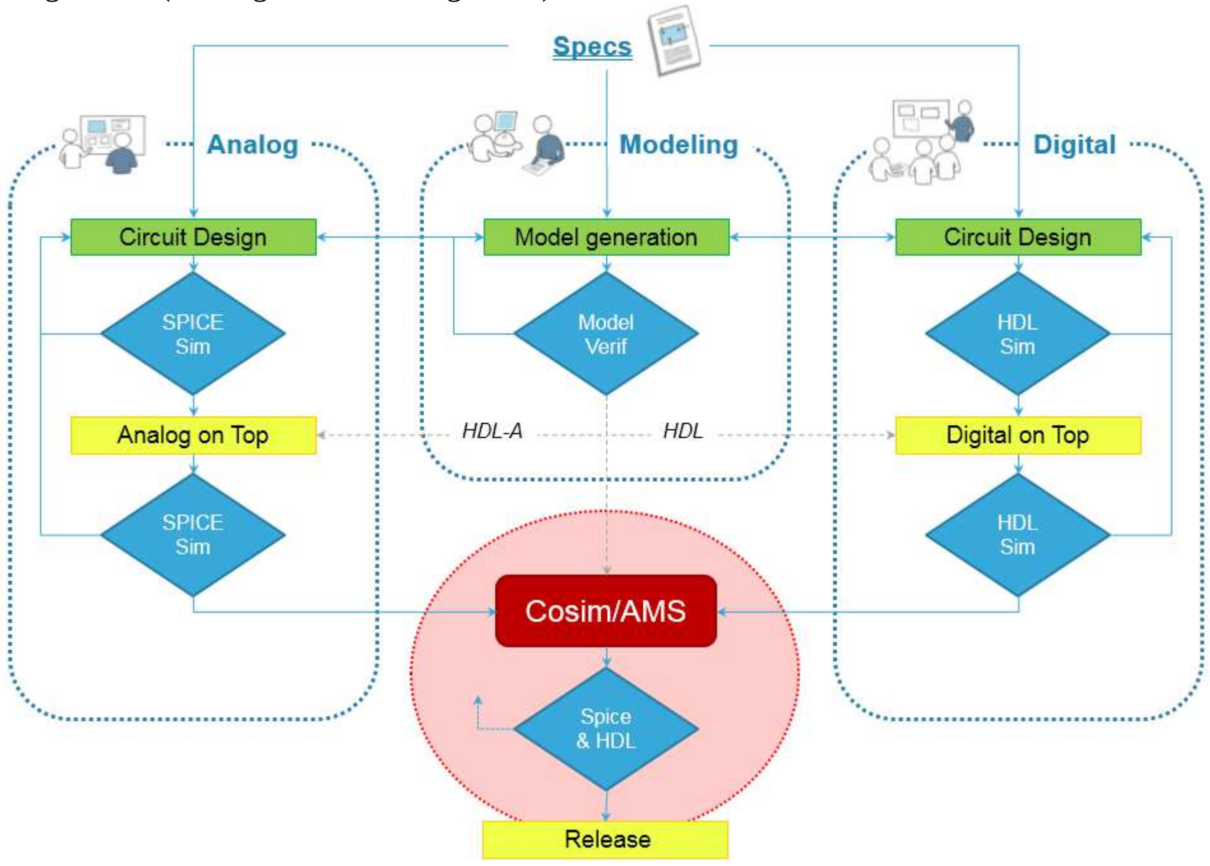


Figure 3 AMS Verification Flow

Analog blocks are always verified by pure SPICE simulation of the circuit at transistor level, sometimes using Verilog-A blocks. Digital blocks are, instead, implemented in VHDL or Verilog and checked with pure digital RTL simulation. Once both analog and digital blocks meet the right level of maturity, these are integrated at the top level for our analog-on top (AoT) verification flow. Obviously, we also have a dedicated flow for digital-on-top (DoT) applications that usually are large CMOS or NVM circuits for consumer or automotive market, which is not presented in this paper. Designers also develop VHDL and Verilog models for the digital parts of the design as well as Verilog-A and Verilog-AMS models for the analog parts. These analog behavioral models are then checked with respect to the transistor-level description of the analog blocks while the digital models are checked with respect to the specifications of the digital blocks through formal verification or using a metric-driven verification (MDV) approach.

Lastly, designers integrate the analog blocks at transistor level or at behavioral description level with the digital blocks in VHDL or Verilog creating a mixed-signal schematic analog-on-top inside a design framework. From the schematic view of the top level, partitioning is carried out, then the netlister is launched. This enables the verification of complex AMS designs, mixing elements such as a Verilog or VHDL digital netlist with a pre-layout and/or post-layout analog netlist, starting from the top level. In the digital domain, designers can also take into account the minimum and the maximum delays of the digital blocks for best and worst case simulations. In the analog domain, designers can take into account the process corners (typical, minimum, maximum, fast-fast, slow-slow, etc…), as well as the user modes, such as for a memory block, its boot, read, erase, and program modes. Designers can also discriminate among different modes of operations, like for instance the *usermode* and the *testmode* facilitating the testing of the application. The final purpose is to verify complex designs at top-level both with digital and analog parts.

4 Case study

The test case selected for our VCS AMS case study is an analog-on-top memory design. It contains an embedded non-volatile memory made of digital circuitry, defined in VHDL and Verilog, and analog circuitry including sense amplifiers, charge pumps, oscillators, band gaps and other related analog blocks.
All the verification effort is driven from a digital test bench written in VHDL because it offers an easier and more flexible solution (specifically for stimuli management). In Figure 4, you can see the block diagram of the applications, which contains a 136K single module memory block with two supplies (an analog supply from 1.6V to 3.6V and a digital supply from 1.08V to 1.32V). It also has an embedded program/erase code.

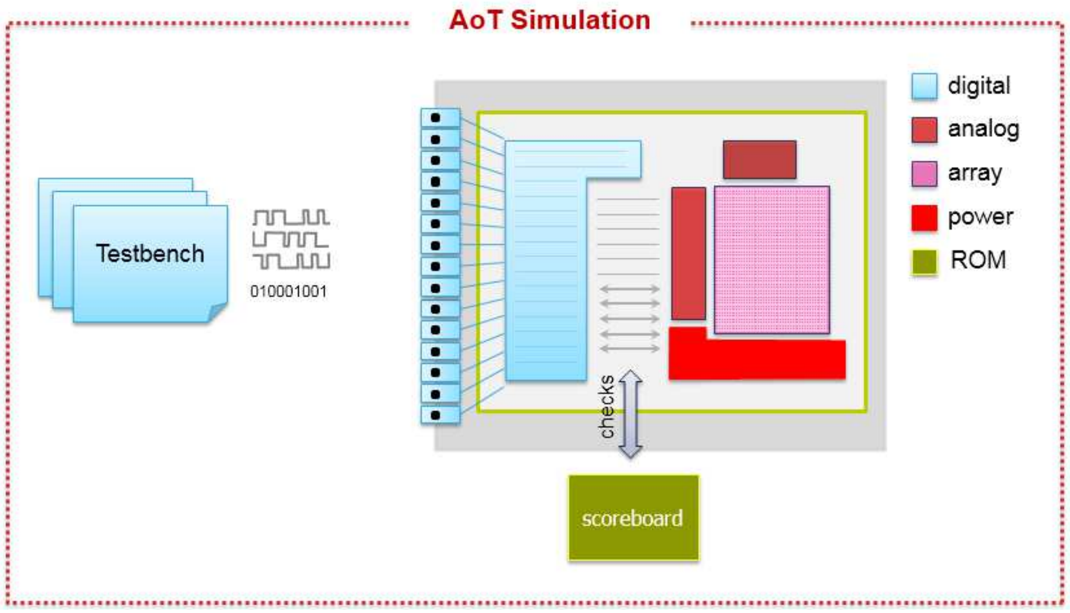


Figure 4 Analog-on-top application to verify with VCS AMS flow

The technology is a low power, double poly, triple-well, non-volatile memory technology at 90nm with differential oxide, MIM capacitors and high resistive poly.

A standard ROM block written in Verilog contains pre-charged information to check that the booting sequence of the application works properly. During the runtime of the booting phase, the sequential code coming out from the embedded memory gets compared with the one written in the ROM. If the two codes are equal, the booting sequence is progressing correctly. If there is a mismatch, there are problems in the power-up phase. This is a method to increase the verification coverage.

4.1 Power-up and Boot Operations

In Figure 5, we can have a look at the power-up sequence of the application. Notice the two power supplies (analog and digital) and the busy signal that indicates when the booting phase starts by raising from 0 to 1.

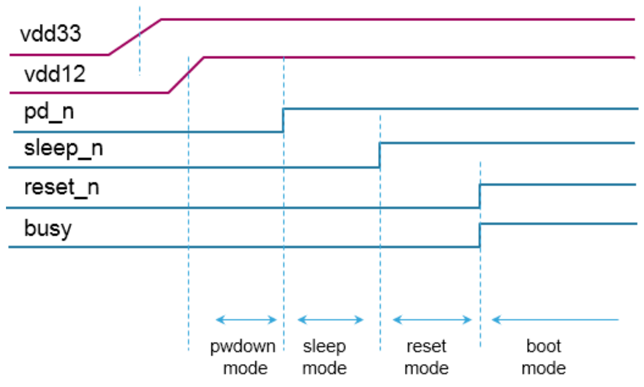


Figure 5 The power up sequence of the application under verification

In Figure 6, you can have a look at the details of the most important analog and digital waveforms during the boot operation (for example the busy digital signal is going down to zero at the end of the booting operation, the sense_out and the top level analog output are also being shown in the same waveform window).

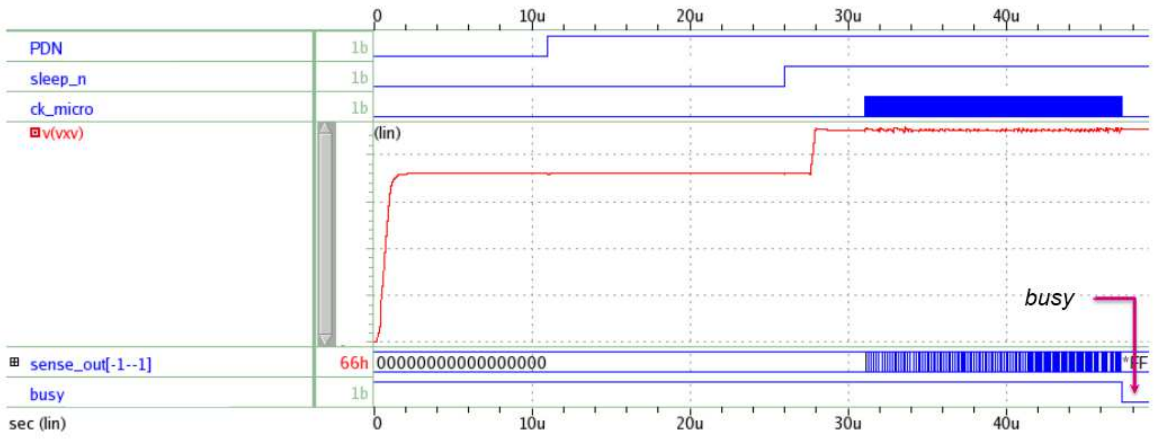


Figure 6 Details of the output waveforms during the simulation of booting operations

In Figure 7, you can see more details of the boot trimming by having a close look at the zoom of the sense_out signal. As you can see, some data are generated during the simulation. These numbers must fit with the code pre-charged in the ROM memory, otherwise there is an error in the boot operation. Details of those data can be seen in the lower part of the figure, where there is a large zoom of one portion of the sense_out signal shown in the waveform display.

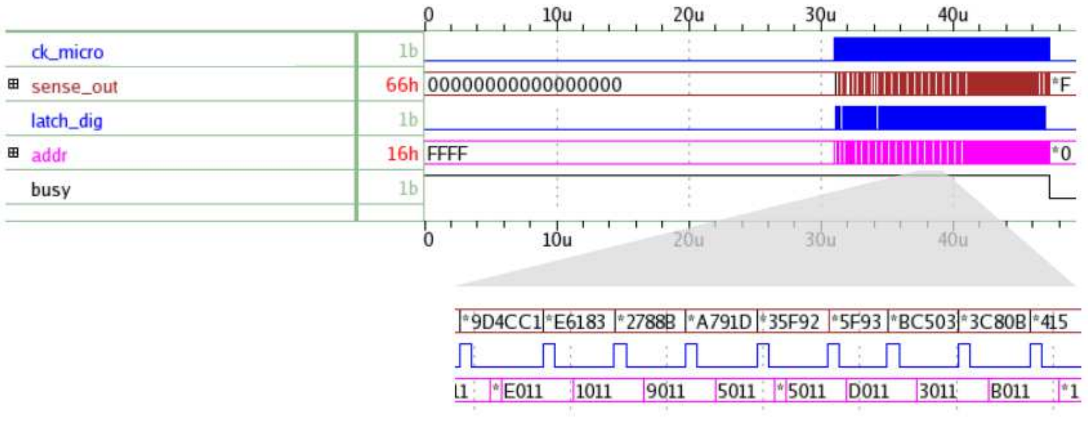


Figure 7 Boot trimming. Sense_out signal details

You can see the sequence of data pointed out by the sense_out signal. These data must match with the ones written in the pre-charged ROM memory.

In the next section, we will see how to use assertions to check automatically whether the output of the sense_out signal matches with the pre-charged code in the ROM memory.

4.2 Assertions

With the usage of assertions, we can easily implement automatic checks. In this case, we want to check that the results in the sense_out bus matches perfectly with the scoreboard contained in the ROM memory (Figure 8).

```
assign data = enable ? mem[addr] : 32'b0;
initial begin
  $readmemh("mem.dat", mem);
end
```

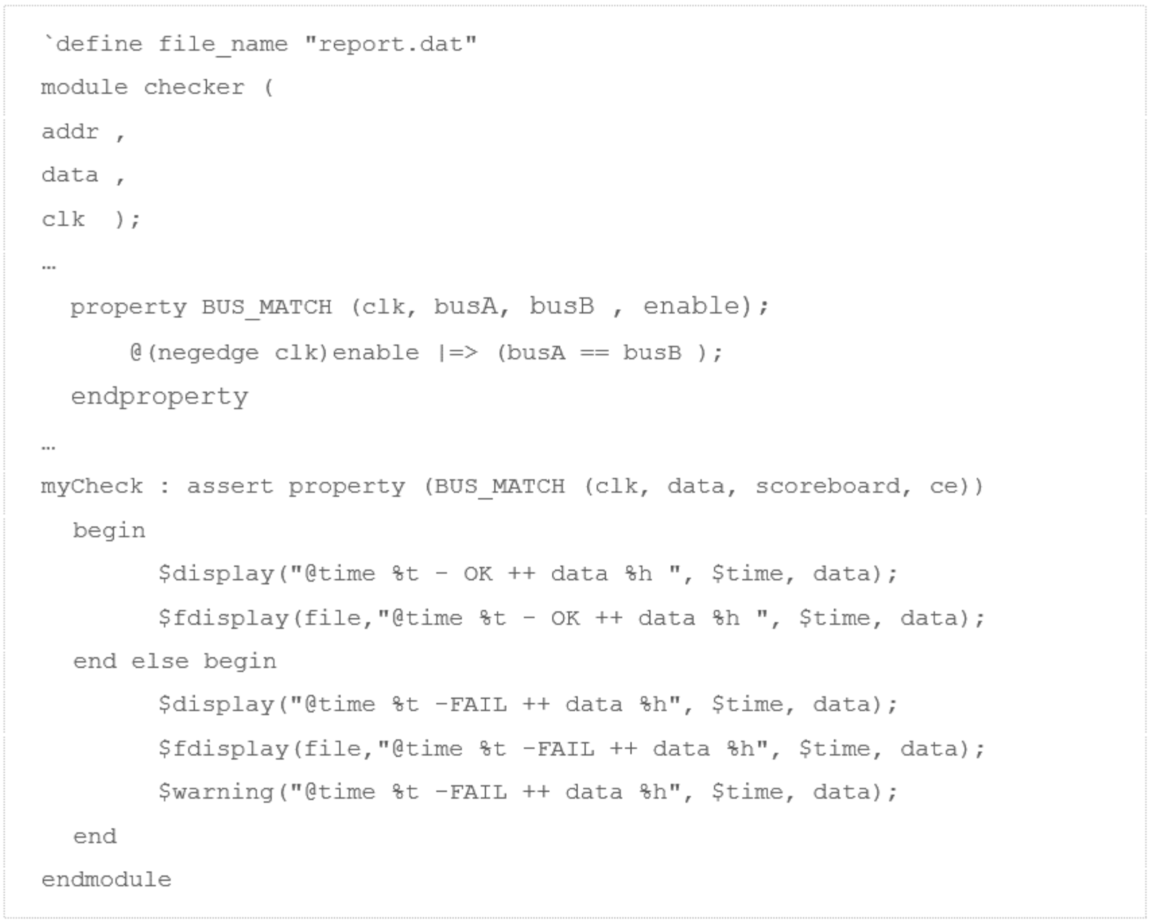
Figure 8 Verilog code fragment: ROM

Assertion specifications can be seen in the following section. We declare a digital input, an analog output and the trigger signal (Figure 9).

```
// addr      -> digital input
// sense_out -> analog output
// latch_dig -> trigger
checker CK (.addr(addr[`A:0]), .data(sense_out[`M:0]), .clk(latch_dig));
```

Figure 9 Verilog code fragment: checker instantiation

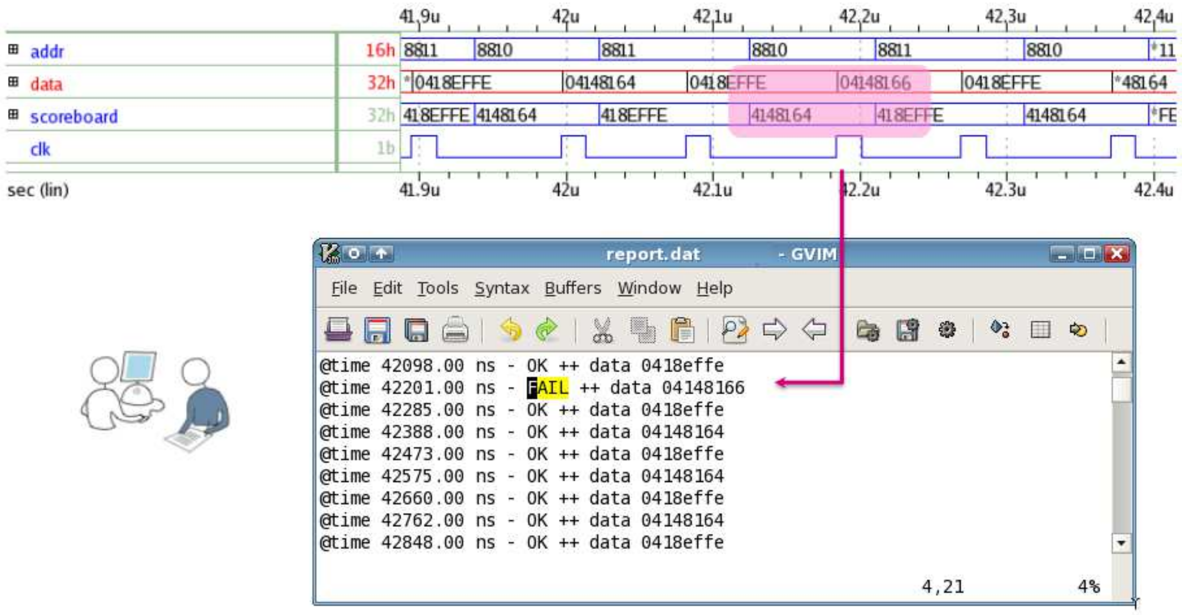
In the following code fragment, we specify the assertion used to check that the content of the memory, during the boot operation, is equal to the content of the ROM that we use as a scoreboard (Figure 10).



```
`define file_name "report.dat"
module checker (
addr ,
data ,
clk  );
…
  property BUS_MATCH (clk, busA, busB , enable);
      @(negedge clk)enable |=> (busA == busB );
  endproperty
…
myCheck : assert property (BUS_MATCH (clk, data, scoreboard, ce))
  begin
        $display("@time %t - OK ++ data %h ", $time, data);
        $fdisplay(file,"@time %t - OK ++ data %h ", $time, data);
  end else begin
        $display("@time %t -FAIL ++ data %h", $time, data);
        $fdisplay(file,"@time %t -FAIL ++ data %h", $time, data);
        $warning("@time %t -FAIL ++ data %h", $time, data);
  end
endmodule
```

Figure 10 SystemVerilog code fragment: assertion

With this mechanism, we can automatically raise a flag if there is a mismatch between the contents of the memory and the scoreboard. In Figure 11, we can have a look of what happens when a real problem is identified. In this case, the content of the sense_out signal is 04148166 while the number written in the scoreboard is 04148164. This mismatch causes a flag to be generated and therefore mismatches can be found automatically without having to visually check the output waveforms. This automated check saves a lot of time in our debugging phase and captures errors which potentially could be undetected if we relied on just examining output waveforms.


addr 16h 8811 8810 8811 8810 8811 8810
data 32h 0418EFFE 04148164 0418EFFE 04148166 0418EFFE 48164
scoreboard 32h 418EFFE 4148164 418EFFE 4148164 418EFFE 4148164
clk 1b
sec (lin)
41.9u 42u 42.1u 42.2u 42.3u 42.4u
report.dat - GVIM
File Edit Tools Syntax Buffers Window Help

```
@time 42098.00 ns - OK ++ data 0418effe
@time 42201.00 ns - FAIL ++ data 04148166
@time 42285.00 ns - OK ++ data 0418effe
@time 42388.00 ns - OK ++ data 04148164
@time 42473.00 ns - OK ++ data 0418effe
@time 42575.00 ns - OK ++ data 04148164
@time 42660.00 ns - OK ++ data 0418effe
@time 42762.00 ns - OK ++ data 04148164
@time 42848.00 ns - OK ++ data 0418effe

                                          4,21          4%
```

Figure 11 Flag raises up automatically in case of mismatch

4.3 Save & Restore

The AMS co-simulation with VCS AMS is very powerful and very flexible. For example, in our case study it allows us to combine a lot of different system configuration both in analog and digital domain. For the analog blocks, we can choose between a pre-layout and a post-layout netlist with parasitic components, then we can choose among several different process corners. For the digital blocks, we can choose VHDL or Verilog, and then we can choose, for example, the minimum or the maximum delay according to the scheme in Figure 12. At operational level, we can also choose the user mode where we check the boot, read, programming with algorithm, erase, user set of operations or the test mode where we can check DMA, MTX_CELL, DMA REF_CELL, Erase Reference matrix (IP validation).

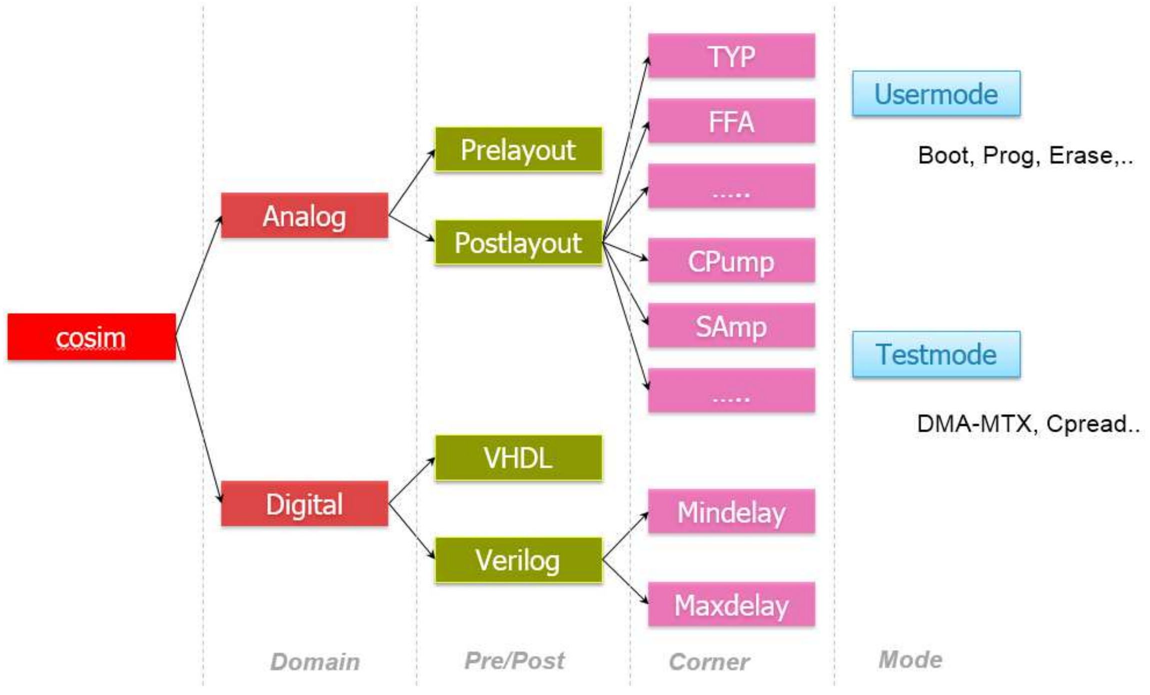
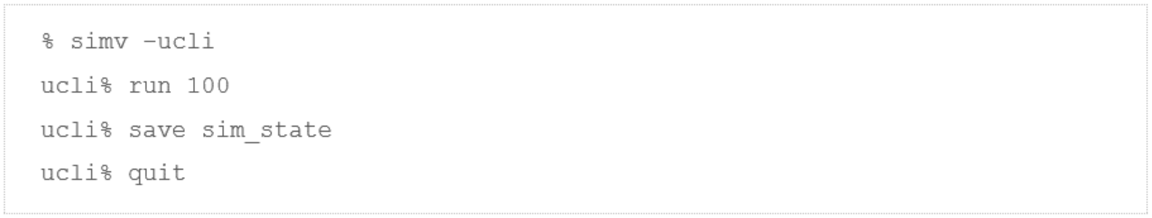


Figure 12 Flexibility of AMS co-simulation with VCS AMS

To check all these combinations, we also took advantage of VCS AMS save & restore capability, which enables designers to run a simulation, save the results at a certain time point, change something in the simulation set-up and then run another simulation with another target. With the save & restore capability, a memory image of the simulation can be saved and restored at a later time. Typically, it is used for running many functional simulations on the same design after the power-up or booting phase. When you restore the simulation, you cannot change the SPICE and the HDL netlists as they represent the device under test, but you can certainly change those things that allow the correct and efficient simulation of the device operation modes. In fact, you can change the following: stimuli, speed/accuracy trade-off, probes. Let us briefly analyze these points. In order to test different operation modes, you need to apply different stimuli, which drive the device under test to perform different functionalities. The stimuli may be in SPICE or HDL formats or even both. The main difference is that the HDL stimuli can be dynamic and adjust themselves according to the devices feedback; of course, once the HDL code is compiled, it is not possible to add additional stimuli once the compilation is done, so all the different stimuli driving the device functionalities must be available before starting the verification phase. New SPICE stimuli instead can be added at the restore level since they do not need to be compiled. However, they are static by their own nature, so they do not provide an easy way to read the device feedback and change accordingly. Therefore, it is clear that the choice of the stimuli type is done with respect to the verification goals. Another relevant feature is the capability to tune speed versus accuracy. In fact, some of these functionalities require high simulation precision at SPICE level, whereas some others may need less accuracy in favour of better performance. So in order to increase the number of simulation runs before tape-out and consequently increase the verification coverage, the possibility to tune the accuracy is now mandatory.

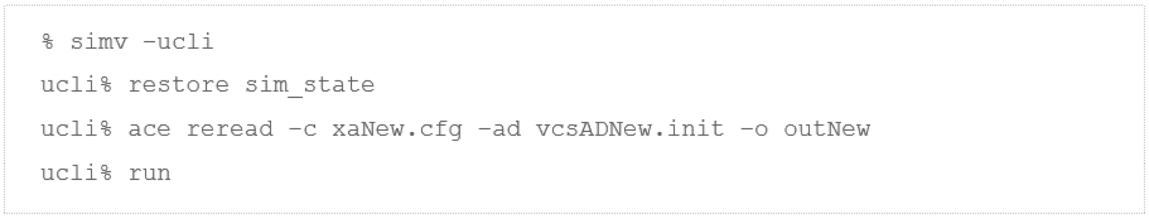
VCS AMS allows changing set_sim_level, set_tolerance_level, set_ccap_level and set_model_level, which help you to tune your simulation to better reach your verification goals. VCS AMS also allows changing the number of probes. In fact, each simulation may require different waveforms to be analyzed. At SPICE level, probing may affect performance, so they need to be carefully used when moving from one operation mode to another. For instance, you may want to probe signals of a certain block for a certain simulation and then you may not need to view them anymore for another simulation. VCS AMS allows you to first save the simulation status by the command save (Figure 13):



```
% simv -ucli
ucli% run 100
ucli% save sim_state
ucli% quit
```

Figure 13 UCLI commands: save

and then to restore it by the command restore (Figure 14):

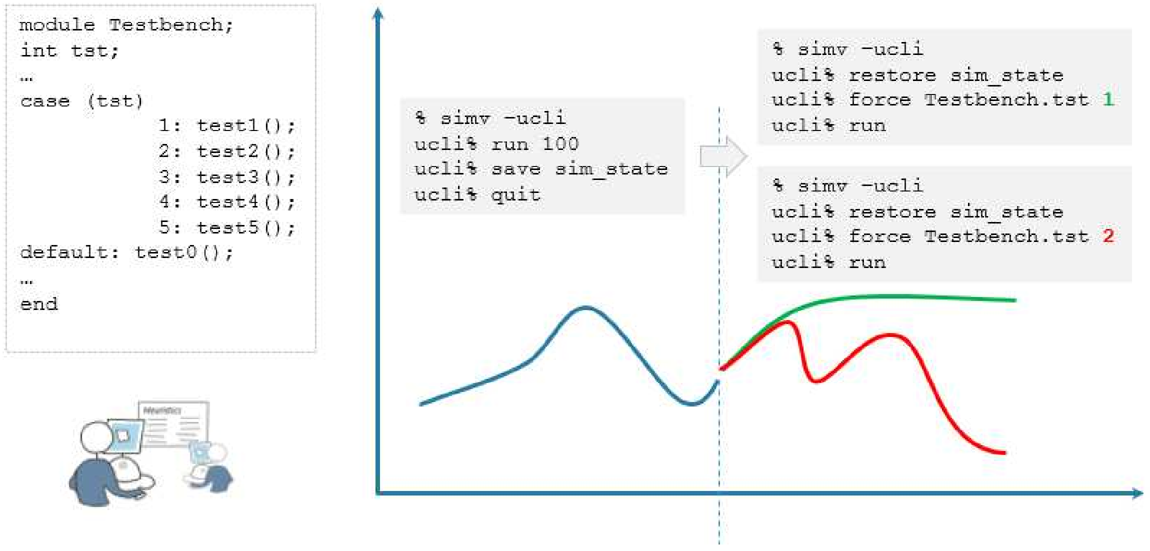


```
% simv -ucli
ucli% restore sim_state
ucli% ace reread -c xaNew.cfg -ad vcsADNew.init -o outNew
ucli% run
```

Figure 14 UCLI commands: restore

Finally, before running the simulation from the restored status, you can use the command reread to pass new simulation setup and stimuli as described above.

Usually for an AMS test bench, it is recommended to manage the stimuli at HDL level. Figure 15 shows a typical approach used for varying the digital stimuli: a variable is forced via UCLI (Unified Command Line Interface) in order to select a specific test.



```
module Testbench;
int tst;
…
case (tst)
          1: test1();
          2: test2();
          3: test3();
          4: test4();
          5: test5();
default: test0();
…
end
```

```
% simv -ucli
ucli% run 100
ucli% save sim_state
ucli% quit
```

```
% simv -ucli
ucli% restore sim_state
ucli% force Testbench.tst 1
ucli% run
```

```
% simv -ucli
ucli% restore sim_state
ucli% force Testbench.tst 2
ucli% run
```

Figure 15 VCS AMS case selection mechanism via UCLI

We use this powerful capability to run many functional simulations of our design application after the time-consuming power-up simulation. From a general point of view, we can apply this approach to any circuit that has multiple mode operations after a booting or a power-up. As you can imagine, after the boot we can select any mode of operations, then we can save again the status after mode one and simulate mode two, then re-save the status again and simulate, for example, mode one again or mode three and so on. In this way, we save tremendous simulation time because any new simulation takes advantages from the previous history without losing time to do again what you already did in the previous simulations. Now going specifically to our test case, we can perform after the booting operations, any read, program or erase simulation in any sequence starting from the previous one, after saving the status, so that we optimize the simulation time. Any sequence is allowed and it can be started from the previously saved simulation data. In the Figure 16, you can see the general methodology applied specifically to our flash memory test case.

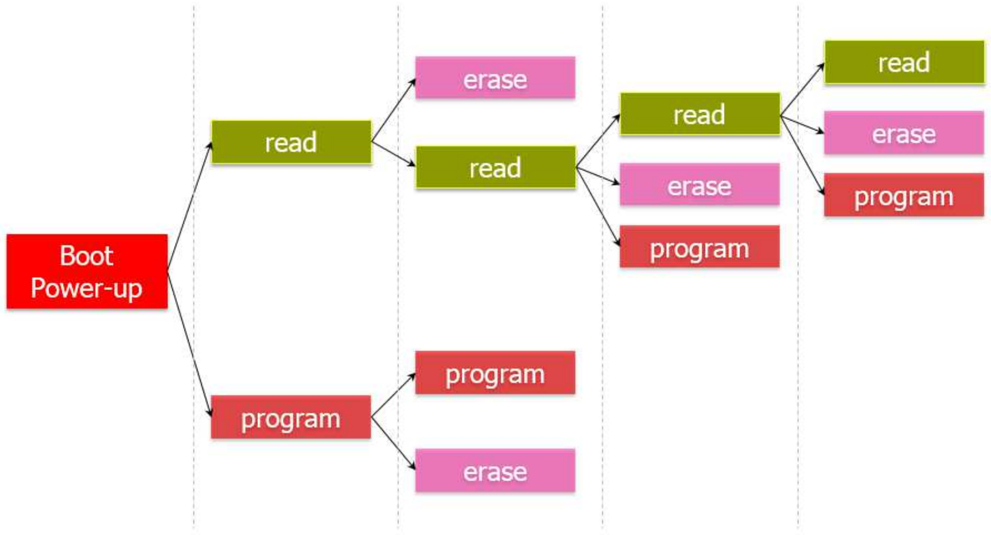
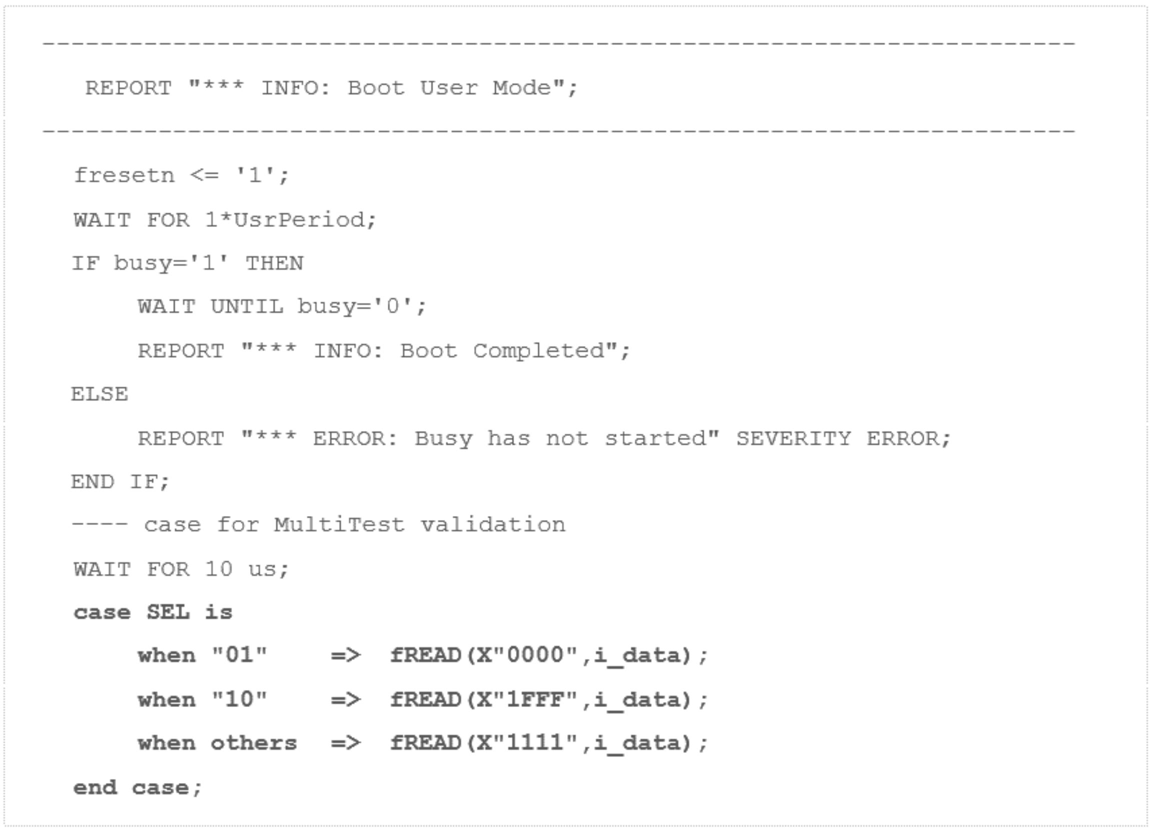


Figure 16 The Save and Restore capability

Basically, designers can define multiple simulation runs directly in the model test-bench, which can then be executed merely by changing the value of a test selector variable. In practice, after running the power-up of the memory and, for instance, a read operation, designers can save the status and then can restart from that point, running another read, programming or erase operation. As in this case a power-up simulation takes approximately 90 minutes to complete and a single read operation a further 10 minutes of simulation time, if we save the state at this point we have saved about 100 minutes every time we run a new simulation starting from this point. With this approach, we can save even more time when running more complex operations such as a program and an erase action. In the next section, we can have a look at the simulation results.

4.4 Simulation Results

At this point, we setup a simulation to check power-up and multiple read operations taking into account the save and restore capability of VCS AMS. In the following code (Figure 17), you can see the test bench where we define the case selection. In this specific example, we define three different possible cases like 01, 10 and others. After the power-up phase, we can save the results and then restore the simulation with the selection 01 (restore_1) or with selection 10 (restore_2).



```
-----------------------------------------------------------------------
   REPORT "*** INFO: Boot User Mode";
-----------------------------------------------------------------------
  fresetn <= '1';
  WAIT FOR 1*UsrPeriod;
  IF busy='1' THEN
       WAIT UNTIL busy='0';
       REPORT "*** INFO: Boot Completed";
  ELSE
       REPORT "*** ERROR: Busy has not started" SEVERITY ERROR;
  END IF;
  ---- case for MultiTest validation
  WAIT FOR 10 us;
  case SEL is
       when "01"     =>  fREAD(X"0000",i_data);
       when "10"     =>  fREAD(X"1FFF",i_data);
       when others   =>  fREAD(X"1111",i_data);
  end case;
```

Figure 17 VHDL code fragment: configurable test bench

As you can see in Figure 18, there is a start-up phase with a lot of activity, which needs about one hour and a half to be completed, then a read operation. In this specific simulation, save & restore capability of VCS AMS allowed us to save 1 hour and 40 minutes of simulation time.

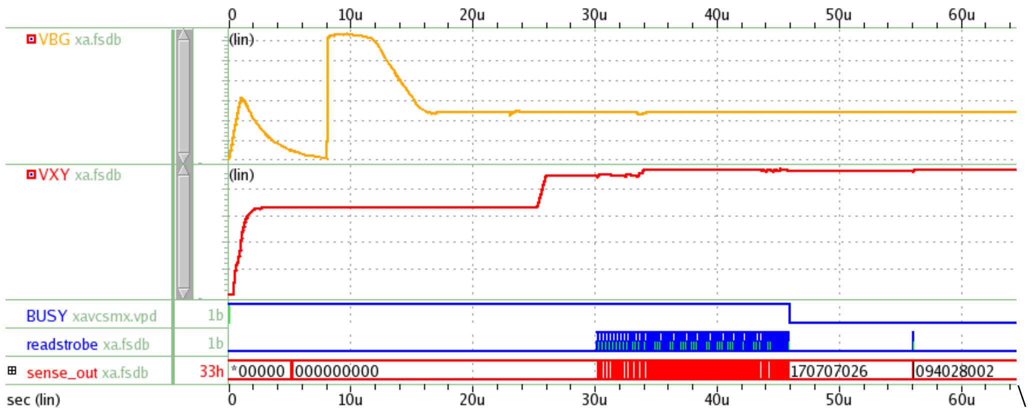
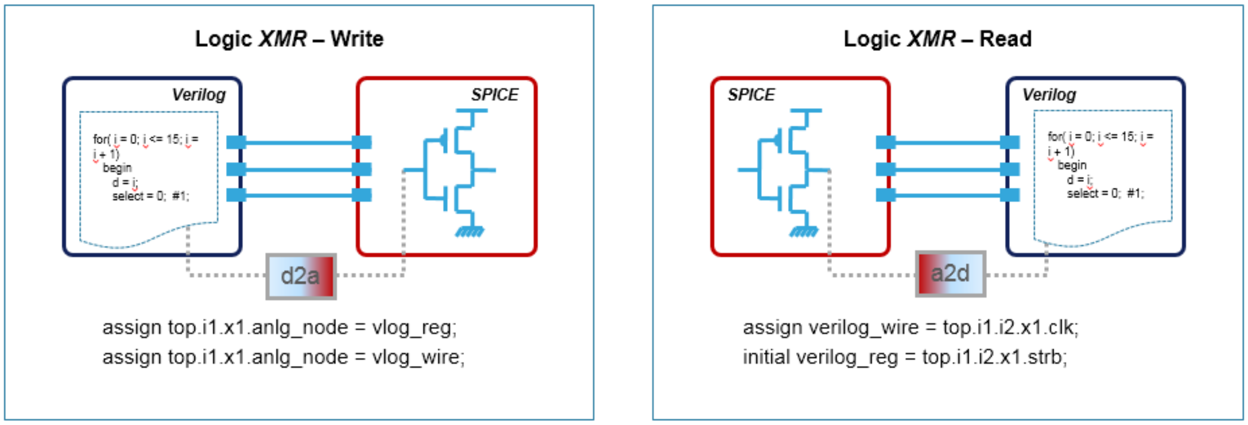


Figure 18 Power-up activity and single read operations after restore

5 Future works

Using digital assertions is a step ahead for the AMS verification, but it might not suffice for increasing the verification coverage. One further improvement is achieved by using analog assertions, which are a powerful additional tool to build a self-checking environment.

Analog assertions allow on one hand to improve the coverage by increasing the portion of the design that is automatically accessed and checked; on the other hand to speed-up the debug phase by turning part of the working load needed for viewing the waveforms into a pass/fail analysis that, for its construction, is more quickly checked. The analog assertions are triggered by an event or by a sequence of events occurring at the analog portion of the AMS design. The concept is similar to the one of the digital assertions, with the difference that in case of analog assertions, a tunneling through the SPICE and HDL border must be granted. For this purpose, VCS AMS provides a set of functions to perform the analog XMR (cross module reference); in short, a communication channel between physical entities (voltage and current) and HDL processes is established. Analog XMRs can be of two types: logic and real. The logic XMRs only deal with voltages and are transparent to the user who simply has to assign a SPICE net to a wire for reading from SPICE or, the other way around, for writing into SPICE. VCS AMS then automatically places A2D (Analog to Digital) or D2A (Digital to Analog) interface elements in order to read or to force "digital" voltages.



Logic XMR – Write
Verilog
for(i = 0; i <= 15; i = i + 1)
begin
d = i;
select = 0; #1;
SPICE
d2a
assign top.i1.x1.anlg_node = vlog_reg;
assign top.i1.x1.anlg_node = vlog_wire;
Logic XMR – Read
SPICE
Verilog
a2d
assign verilog_wire = top.i1.i2.x1.clk;
initial verilog_reg = top.i1.i2.x1.strb;

Figure 19 Logic XMR

Logic XMR example:

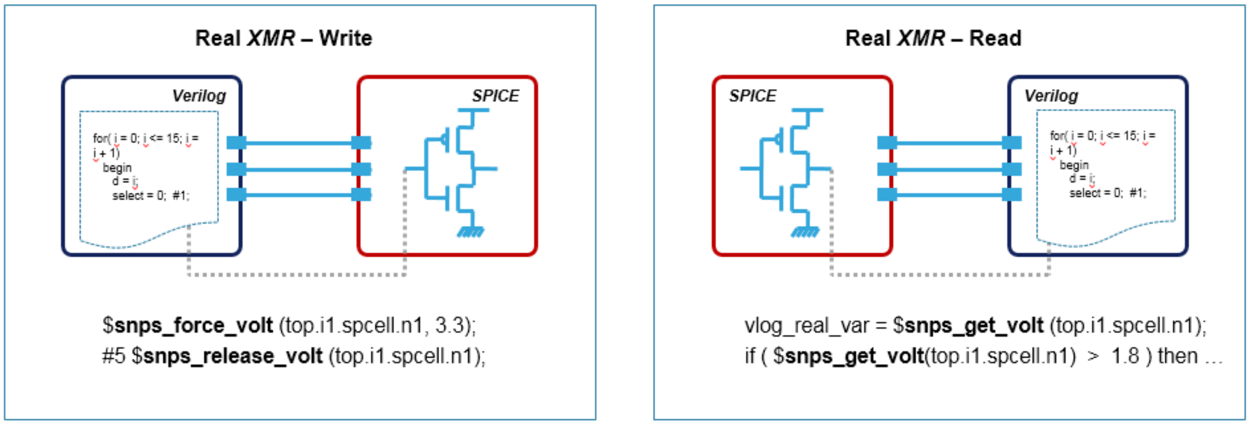
```
assign top.i1.x1.anlg_node = vlog_reg;
assign top.i1.x1.anlg_node = vlog_wire;
```

Figure 20 Verilog-AMS code fragment: XMR

The analog XMRs allow instead the management of both voltages and currents by using the following functions:

- ✓ snps_force_volt
- ✓ snps_get_volt
- ✓ snps_get_port_current
- ✓ snps_above
- ✓ snps_across

Voltages and currents are assigned to real variables that will be used for writing analog assertions. These can be synchronous, which means triggered by a digital event like a clock, or asynchronous, which means triggered by an analog event like a voltage crossing through a pre-defined threshold.



```
for( i = 0; i <= 15; i = i + 1)
  begin
    d = i;
    select = 0;  #1;
```

```
$snps_force_volt (top.i1.spcell.n1, 3.3);
#5 $snps_release_volt (top.i1.spcell.n1);
```

```
vlog_real_var = $snps_get_volt (top.i1.spcell.n1);
if ( $snps_get_volt(top.i1.spcell.n1)  >  1.8 ) then ...
```

Figure 21 Real XMR

Features like analog assertions and coverage (this last one is not part of this article) naturally lead to a wider concept: the AMS test bench (in digital verification environment the test bench is the core of the verification process). In fact, the increasing complexity of ASIC and SoC is generating a compelling need for a self-checking environment that reduces at the minimum the risk of failure on the silicon.

The assertions can be fully implemented by the user; however VCS AMS already provides the user with a set of AMS predefined checkers. These checkers are shown in Table 2.

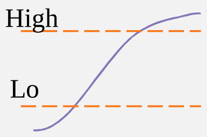
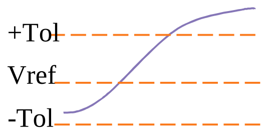
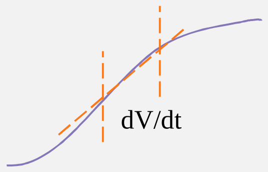
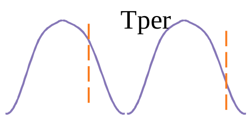
CHECKERS		
sv_ams_threshold_checker	Checks that analog signal remains within a given high and low threshold. Can perform this check synchronously or asynchronously	
sv_ams_stability_checker	Checks that analog signal remains below or above a given threshold. Can perform this check synchronously or asynchronously	
sv_ams_slew_checker	Checks that analog signal rises/falls with a given slew rate(+/- tolerance). Can perform this check synchronously or asynchronously	
sv_ams_frequency_checker	Checks that analog signal frequency is within a given tolerance	

Table 2 Example of VCS AMS pre-defined checkers

In STMicroelectronics, not all the AMS designing teams have real verification engineers inside them. Sometimes, verification is simply demanded to the designers themselves, but this approach in our opinion is not correct. Any designing team should have in its staff expert verification engineers. This attitude will be the key factor to accomplish first silicon success for complex AMS applications in nanometer technologies in the next years. This is the bet to win for the next decade.

6 Conclusions

In this paper, we have shown how the usage of VCS AMS enhanced our AMS verification capabilities. We also analyzed the state-of-the-art for mixed-signal verification, taking into account VCS AMS as a superior solution to drive AMS verification challenges. We introduced VCS AMS *assertions* and *save & restore* in our flow, describing them through a dedicated case study. By exploiting automatic checks, we allowed designers to save the time, which would otherwise be required to perform manual checking. Different memory operations (read, write, erase) have been performed without repeating any time the power-up, the boot and the previous actions. This methodology allows designers to save a substantial amount of time in the circuit verification phase. Finally, we pushed the barrier to what is next, introducing the concept of cross-module reference (XMR) both for analog and digital signals, also showing VCS AMS set of pre-defined AMS assertions which designers could get advantages from. We also pointed out the importance of forming qualified AMS verification engineers for any designing team to improve and to drive the first silicon success, which will be the challenge for the next decade.

7 References

[1] Mixed-Signal Simulation User Guide, J-2014.09-SP3

[2] CustomSim™ Command Reference, J-2014.09-SP3

[3] CustomSim™ User Guide, J-2014.09-SP3

[4] VCS® MX/VCS® MXi™ User Guide, J-2014.12, J-2014.09-SP3

[5] VCS® MX/VCS® MXi™ Unified Command Line Interface User Guide, J-2014.09-SP3

[6] HSIMplus Based Mixed-Signal Design and Verification, SNUG 2008 - Munich

[7] NVM-IP mixed signal design and verification, ISWMM 2013 - Agrate Brianza

[8] Complex Mixed-Signal SoCs - How to Conquer the Next Verification Frontier
Circuit Simulation Luncheon at DAC 2014 - San Francisco

[9] Extending Proven Digital Verification Techniques for Mixed-Signal SoCs with VCS AMS
DVCON 2014 - Munich

[10] VCS AMS for Advanced SoC Mixed-signal Verification - Synopsys Verification Webinar 2014

8 Appendix

For the sake of completeness, we share in this appendix the following code fragments in text format for an easy *‘cut & paste’*

Verilog code fragment: ROM

```
assign data = enable ? mem[addr] : 32'b0;
initial begin
  $readmemh("mem.dat", mem);
end
```

Verilog code fragment: checker instantiation

```
// addr      -> digital input
// sense_out -> analog output
// latch_dig -> trigger
checker CK (.addr(addr[`A:0]), .data(sense_out[`M:0]), .clk(latch_dig));
```

SystemVerilog code fragment: assertion

```
`define file_name "report.dat"
module checker (
addr ,
data ,
clk  );
…
  property BUS_MATCH (clk, busA, busB , enable);
      @(negedge clk)enable |=> (busA == busB );
  endproperty
…
myCheck : assert property (BUS_MATCH (clk, data, scoreboard, ce))
  begin
        $display("@time %t - OK ++ data %h ", $time, data);
        $fdisplay(file,"@time %t - OK ++ data %h ", $time, data);
  end else begin
        $display("@time %t -FAIL ++ data %h", $time, data);
        $fdisplay(file,"@time %t -FAIL ++ data %h", $time, data);
        $warning("@time %t -FAIL ++ data %h", $time, data);
  end
endmodule
```

UCLI commands: save

```
% simv –ucli
ucli% run 100
ucli% save sim_state
ucli% quit
```

UCLI commands: restore

```
% simv –ucli
ucli% restore sim_state
ucli% ace reread –c xaNew.cfg –ad vcsADNew.init –o outNew
ucli% run
```

VHDL code fragment: configurable test bench

```
-----------------------------------------------------------------------
   REPORT "*** INFO: Boot User Mode";
-----------------------------------------------------------------------
  fresetn <= '1';
  WAIT FOR 1*UsrPeriod;
  IF busy='1' THEN
      WAIT UNTIL busy='0';
      REPORT "*** INFO: Boot Completed";
  ELSE
      REPORT "*** ERROR: Busy has not started" SEVERITY ERROR;
  END IF;
  ---- case for MultiTest validation
  WAIT FOR 10 us;
  case SEL is
      when "01"     =>  fREAD(X"0000",i_data);
      when "10"     =>  fREAD(X"1FFF",i_data);
      when others   =>  fREAD(X"1111",i_data);
  end case;
```

Verilog-AMS code fragment: XMR

```
assign top.i1.x1.anlg_node = vlog_reg;
assign top.i1.x1.anlg_node = vlog_wire;
```