

Utilizing Digital Techniques for Analog and Mixed-Signal Verification

July 2010

Author Introduction

Andy Milne,
Damian Roberts
Application
Consultants, AMS
Circuit Simulation
Synopsys, Inc.

As the analog and mixed-signal content in modern SoC designs increases, the verification task becomes significantly more complex and time-consuming. Multiple power domains, block interfaces, design size and unfamiliar hardware description languages all add to the difficulties for the verification engineer. The world of digital verification has developed many methodologies to help tackle these problems, e.g., assertions, coverage-driven testbenches and model abstraction. The verification of analog/mixed-signal (AMS) systems has traditionally relied on a more manual approach. This white paper discusses how digital techniques can be applied to enable thorough verification of the most complex mixed-signal designs in a timely and efficient manner.

Utilizing Digital Techniques

Despite mixed-signal SOC's being commonplace, there is still no standard methodology for verifying these chips. This has led to a growing percentage of design respins due to increased analog content. Designers need to sign off on tapeouts as quickly and confidently as possible, but too often the verification of the analog and digital blocks is carried out as two separate, disjointed processes.

The most common approaches for verifying mixed-signal blocks are either to simulate the entire chip at the transistor level using a FastSPICE tool (e.g., CustomSim) or to write simple digital HDL models of the analog blocks and simulate the system entirely in a digital simulation tool (e.g., VCS). Both of these solutions have their pros and cons. Although FastSPICE technology does make it possible to accurately simulate multi-million-transistor designs, the simulations are relatively slow when compared to gate-level digital simulation, so it is not always possible to achieve comprehensive functional coverage. Producing digital models to describe the analog functionality is efficient in terms of runtime but the models are often difficult to correlate to the analog implementation and lack the required fidelity. This results in bugs being overlooked.

The ability of CustomSim to co-simulate with Synopsys' VCS digital simulator opens up the possibility of a "best of both worlds" approach enabling the majority of the chip to be simulated in the digital domain while the FastSPICE tool handles the analog blocks. Using this mixed-signal approach, many techniques and philosophies that are common in the digital world can be applied to the verification of mixed-signal systems.

Verilog Top, Verilog Structural Netlist

Many of the techniques presented below can be used in an 'Analog Top', SPICE netlist flow. However, the use of Verilog for the design structural netlist format, and so a Verilog top view of the design, does yield significant advantages for both the implementation and verification of the design. 'Verilog Top' allows the integration of digital and AMS blocks to be carried out within the digital implementation flow. The drive and load properties of the AMS blocks, together with their timing requirements, can be captured. This allows the designer to perform accurate area/power/timing tradeoffs. More design tasks can be automated with reduced risk. Shorter development times and lower overall development costs are the result. Verilog has the advantage of allowing the passing of port connectivity explicitly by name. SPICE netlists typically only support passing of connectivity by port order. This removes the potential port mismatch that can occur when blocks from different sources are instantiated. 'Verilog Top' makes life easier for verification engineers. The objective is to extend the existing verification techniques and methodologies into the AMS part of the design. Maintaining the same design hierarchy and netlist format helps to make this more 'natural'. Verilog is case-sensitive and SPICE is not, therefore using signal/module/instance names in a Verilog netlist that differ only by case should be avoided.

Keep Testbenches and Models Digital

Achieving the required simulation throughput for an AMS design is a significant challenge. Digital event-driven simulations run much faster than continuous-time analog simulations. In addition, writing true continuous-time analog models, with all the associated concerns of convergence and satisfying value conservation, is a nontrivial task. Taking a signal flow approach to AMS block modeling using real numbers to represent analog quantities is much simpler. It's therefore suggested that, wherever possible, digital constructs should be used instead of analog ones for models, testbenches and sources, allowing the entire simulation to be run in the digital simulator. Replacing the abstract model with the SPICE implementation is as simple as switching the netlisted view. This 'SPICE on Demand' approach simplifies model correlation and debugging issues because the same testbench is used no matter what level of abstraction is used.

```
Verilog top
testbench1 testbench (.sine_out(audio));
SPICE testbench
.subckt testbench sine_out
    Vsin1 sine_out 0 1.5 0 1k
.ends
```

Since a SPICE primitive is used, this requires both analog and digital simulation, even if the design being simulated is entirely digital

```
Verilog sine_source
module sine_source (sine_out);
    output sine_out;
    wreal sine_out;
    parameter real frequency = 1k, amplitude = 0.5;
    parameter real offset = 0.5, steps=1000;
    real cycle,update;
    reg tick;
    initial begin
        update = 1e9/(frequency * steps);
        tick = 0;
        forever #update tick = ~tick; end
    always @( tick) begin
        cycle = cycle + 2 * `PI / steps;
        if(cycle > 2*PI) cycle =cycle - 2*PI;
    end
    assign sine_out = sin(cycle) * amplitude + offset;
endmodule
```

The Sine Source Verilog module is highly reusable and only requires digital simulation, therefore simulating orders of magnitude faster

Below are three examples of measuring the clock frequency generated by a PLL modeled in SPICE:

```

module testbench (clock_in);
input clock_in;
parameter    real threshold = 1.5;
real time_old;
real time_new;
real period;

analog begin
@(cross(V(clock_in) - threshold, +1) begin
time_old = time_new;
time_new = $realtime();
period = time_new-time_old;
$strobe("%f", 1/period);
end
end
endmodule

```

**Frequency Measurement
Option 1**

```

module testbench (clock_in);
input clock_in;
real time_old;
real time_new;
real period;

@(posedge(clock_in) begin
time_old = time_new;
time_new = $realtime();
period = time_new-time_old;
$strobe("%f", 1/period);
end
endmodule

```

**Frequency Measurement
Option 2**

```

module testbench (clock_in);
input clock_in;
real time_old;
real time_new;
real period;

always@(snps_cross($snps_get_volt(tb.DUT
.clock_in) - 0.5, +1)) begin
time_old = time_new;
time_new = $realtime();
period = time_new-time_old;
$strobe("%f", 1/period);
end
endmodule

```

**Frequency Measurement
Option 3**

Option 1 uses the Verilog-AMS @(cross) construct to detect the rising edge of the clock signal and so measure the clock period and frequency. However, even though the PLL is SPICE and co-simulation is required, it would still be better to do the measurement digitally, as the @(cross) construct is much more CPU ‘expensive’ than @(posedge) on a digital signal. Option 2 illustrates the use of @(posedge). Alternatively, the snps_cross function together with the \$snps_get_volt system task could be used and is illustrated in Option 3. These digital measurement methods provide even more benefit if a digital or real number abstract of the PLL was substituted later in the verification process, as the entire simulation could now run in the digital simulator.

Analog Signals in a Digital Netlist

Even though we have chosen to use a primarily digital format as our preferred netlist, many of the pins in the design will represent analog signals. These analog signals could be represented as Verilog reals in our models, but they still need to be correctly passed around the design hierarchy. Traditionally, Verilog does not allow the use of a real as a port.

The obvious solution to this is to encode the real into a 64-bit vectored signal and pass that around. However, this has the serious drawback in that a 64-bit bus has now been introduced where there was previously a single wire. The netlist that is verified is no longer the netlist that is implemented.

A possible workaround to this is to encode the real values onto a serial digital bitstream instead. This keeps the port as a single bit and allows the real value to be passed around the hierarchy as required, just requiring a serial to real encoding at the receiver. One minor drawback of this method is that conversion of a real to/from the serial bitstream is not implicit, there is the requirement to add an encoder/decoder wherever the signal is required to be converted.

Another, perhaps better, method may be to use a concept borrowed from VHDL, real ports. Both SystemVerilog and VerilogAMS bring this functionality to Verilog, but in slightly different ways. SystemVerilog allows a real to be defined as a port, whereas VerilogAMS has the concept of wreal (wired real) ports. This means that real values can now be passed around the hierarchy as a single wire.

System Verilog Assertions

System Verilog Assertions (SVA) are primarily used as an efficient way of specifying checks to validate that a digital system is functioning as specified. SystemVerilog coverage checks are also useful as a technique for ensuring that a testbench (or set of testbenches) covers all of the functional conditions outlined in the test plan for the system.

As part of the system specification, assertions can be used to capture the design intent and capture any assumptions that have been made about interfaces between design blocks. They describe events, or sequences of events, that either should or should not occur if the system is functioning correctly. Therefore, if an assertion is triggered, the verification engineer can then debug where the assertion occurred and identify the conditions that led to the unexpected behavior. Tools such as VCS can also generate coverage reports for all of the assertions in a design, e.g., passed, proven, failed and vacuous.

In a mixed-signal system, the concept of SVA can be applicable not just to the digital blocks in the design but also to the analog sections. An example of a possible AMS assertion would be to check that a bias signal, or internal supply in an analog block, is above a certain current or voltage level before an enable signal is triggered in one of the digital sections. Other checks could include checks on analog slopes, frequencies, current and voltage ranges, etc.

One major advantage to using SystemVerilog assertions for analog checks is that the same testbench can be used no matter how the analog blocks are modeled in the system, i.e., the same assertion that is used to check a Verilog model of an analog block can be used later on in the design process when a SPICE netlist for the analog block is available.

The concept is best illustrated using a simple example. In this example, we have a design made up of an analog block called “ramp” being driven by a clock signal from a Verilog testbench. Let’s say that the analog designer knows that during normal operation of the circuit, the node called “cap_node” inside the SPICE block should never rise above 0.7V. An assertion could be added to the testbench to check that the internal node “cap_node” inside the analog block never rises above 0.7V. The Verilog netlist and SPICE netlist are as follow:

```

`timescale 1ns/1ps
`define CYCLE 100
`define HALF_CYCLE (`CYCLE/2.0)
`define SIM_TIME 4000
`define STEP (`SIM_TIME/100.0)

module top();
    parameter bias_thresh = 0.7;
    reg fin;
    ramp x1 (.in(ain), .vdd(vdd), .vss(vss));
    supplies x2 (.vdd(vdd), .vss(vss));
    not x3 (ain, fin);

    always #`HALF_CYCLE fin <= ~fin;
    always @(fin) begin
        cap_volt <= ($snps_get_volt(x1.cap_node));
        a1: assert(cap_volt < bias_thresh);
    end
    initial #0 fin = 1'b0;
    initial #`SIM_TIME $finish(2);
    initial $vcdpluson;
endmodule //top

```

```

.inc 'cmos35t.mod'

.subckt inv in out vdd gnd
    mn1 out in gnd gnd n w=5.00u l=1.00u
    mp1 out in vdd vdd p w=10.00u l=1.00u
.ends

.subckt ramp in vdd vss
    xinv in res_node vdd gnd inv
    r1 res_node cap_node 100k
    c1 cap_node 0 10p
    r2 cap_node 0 500k
.ends ramp

.subckt supplies vdd vss
    vpower vdd 0 3v
    vground gnd 0 0v
.ends supplies

```

At every edge of the clock “fin” the value at the “cap_node” is checked to see if it is more than 0.7V. This example uses the \$snps_get_volt system task. This task returns the voltage value on the specified analog node as a real number. Other available system tasks include, \$snps_force_volt (along with the associated \$snps_release_volt) and \$snps_get_port_current. These can be used to set voltage values on analog nodes from the Verilog testbench and access the current flowing through an analog port, respectively.

The use of Verilog-AMS in conjunction with assertions enables some further capabilities. It is possible to instantiate a Verilog-AMS monitor inside an analog block and then include assertions directly inside this block. This then gives the verification engineer access to all of the Verilog-AMS analog functions that can then be used to carry out complex analog measurements. This methodology enables measurement and self-checking of complex analog behaviors.

This technique can be demonstrated using a simple circuit. In this test case, we have now inserted a Verilog-AMS monitor called “morethan” into our SPICE netlist as shown below:

<pre> .inc 'cmos35t.mod' .subckt inv in out vdd gnd mn1 out in gnd gnd n w=5.00u l=1.00u mp1 out in vdd vdd p w=10.00u l=1.00u xtest out 0 morethan .ends .subckt ramp in vdd vss xinv in res_node vdd gnd inv r1 res_node cap_node 100k c1 cap_node 0 10p r2 cap_node 0 500k .ends ramp .subckt supplies vdd vss vpower vdd 0 3v vground gnd 0 0v .ends supplies </pre>	<pre> Verilog-AMS monitor `include "constants.vams" `include "disciplines.vams" module morethan (a, b); input a, b; electrical a, b; parameter real cth=80e3; real diffvolt; analog begin if (analysis("dc")) diffvolt = 0; else diffvolt = abs(ddt(V(a, b))); end always @(cross(diffvolt - cth, +1)) begin DiffVoltCheck : assert(0) ; end endmodule </pre>
---	--

This module monitors the rate of change of voltage on the output of the inverter and the assertion checks that this rate of change does not exceed a given threshold. Therefore, in this simple monitor we have made use the Verilog-AMS time-derivative operator, as well as triggering the assertion using the Verilog-AMS “@cross” function. By using this analog event as the trigger, the monitor is checking the assertion continuously rather than on discrete clock edges.

One drawback of the example above is that it was necessary to edit the SPICE netlist in order to instantiate the Verilog-AMS monitor. Rather than editing the design netlist, it is possible to use a “virtual instantiation” via the SystemVerilog “bind” capability. Bind can be used to apply assertions on mixed-signal blocks as well as digital modules and can be a useful aid to reuse of the mixed-signal checker modules.

Conclusion

SystemVerilog is a powerful verification tool that should not be ignored by analog and mixed-signal engineers, particularly when used in conjunction with Verilog-AMS. A number of digital techniques that can be used to help verify AMS designs have been discussed. The adoption of many of these significantly improve the efficiency of verifying AMS chips, not only by standardizing methodologies between the digital and AMS domains, but also by increasing the confidence that a thorough and comprehensive verification of the complete design has been conducted.