Automatic Mixed-Signal Design Verification Instrumentation with Observation Specification Language

Jonathan David
Scintera Networks, Inc.
1154 Sonora Ct.
Sunnyvale, CA 94086
j.david@ieee.org
Senior Member IEEE

ABSTRACT

A rigorous approach for quantifying the completeness of design verification for mixed-signal circuits is proposed. A metric, Enclosure, is proposed. A methodology for its measurement is described, using Verilog-A wrappers and commercial simulators to capture observation data in an XML format. A prototype Observation Specification Language is used as a simple description to specify the contents of verification wrappers or instrumentation modules, allowing the necessary automation. A tool to build the instrumentation modules from this description is developed in perl. Using a CML-Buffer circuit as an example, the resulting data is presented in a format useful for guiding the verification effort.

1. INTRODUCTION

A high fraction of design bugs identified in Mixed-Signal integrated circuits are attributed to the Analog or Mixed-Signal sections or to the interfaces to these sections[9,10]. While the digital design verification approach uses the concept of "coverage" for determining how much of the verification has been done, the analog approach has been primarily "running corners" on a couple of the top level simulations, and carefully crafting checklists to record all the simulations that need to be run[6]. The fact that these simulations take the longest to run doesn't help matters much.

Many solutions to these issues have been tried, each improving the situation a little bit. Simulator vendors tend to focus on improving simulator throughput, while design managers may focus more on making better checklists, and trying to manage all the results. Digital design teams are extending the Coverage metric to include the mixed signal interface[4,11,12], and mixed-signal behavioral models can be utilized to speed up functional simulation[7,10,13], and report problems[13] but none of these have proposed a verification methodology that really deals with the analog circuits and all their complexities.

As we look towards Analog verification, the number of parameters involved quickly leads to the conclusion that this is a problem of much larger dimensions than is typical for digital. While a logic gate will typically have min, typical and max timing based on all the process, voltage and temperature conditions, in analog, we will typically have to evaluate design performance for multiple supply voltages, one or more bias currents, best and worst case corners for several types of transistors, resistors, capacitors and inductors, and analog and digital control signals. Since this is a problem of larger dimensions, well adopt a terminology of volume, rather than area. We propose to call our metric of analog/mixed-signal verification, "enclosure"[8] in contrast to the digital verification metric of "coverage"[3].

In addition to the factors that affect the actual circuit performance, we include in our dimensions both the context in which the unit is simulated (Unit level, block level, or top level) and the design representation used. (behavioral, schematic, or with extracted parasitics). These will factor into the interpretation of the verification score [see page 6].

Traditionally analog simulators write their results based on the design hierarchy of the testbench. not by the circuit being tested, requiring development of a unique set of save, measure or other postprocessing commands, for each testbench. For this methodology, we'll create a "wrapper" module that can be instantiated around the design, in every testbench run, that will capture all the data we will use in our enclosure analysis, into files of our own format.

In selecting a format for the captured data, we need to consider both the capabilities of Verilog-A/AMS[1,2] to write the data, and the program that will analyze the data and generate our enclosure report. Rather than create our own data format, we'll use XML[5] as our data language for its clear structure, easily parsed by software programs,

978-1-4244-1567-0/07/$25.00 © 2007 IEEE

and its internally documented readability, for trouble-shooting when things go wrong.

2. METHODOLOGY

Instrumenting our design with verification wrappers, the output data will record "points" of observation in the multi-dimensioned design space. From these point we determine a volume of enclosure, and compare this to the total volume of that design space.

2.1 Design Enclosure

The enclosure metric[8] is defined as the ratio of the volume enclosed by the shape connecting our observation points, to the total volume defined by the shape connecting the points at the extremes of the allowed range of the observation variables. It should be noted that running a "best", "worst" and "typical" set of simulations, will generate only 3 points, which can at best define a plane the in 3-dimensional space of "Process, Voltage and Temperature." Since the purpose of this metric is to help us improve the quality of our verification, we'll define a "fallback" metric, based on the difference between the dimension of the total volume, and the volume enclosed. For the case where 3 of 5 dimensions are enclosed (for an enclosure in the 3 dimensions of 50%) we'll note that the enclosure is $50\%^{-2}$ In the other extreme, where randomness is introduced into the simulations, we may have to measure a density of observation, rather than just simple enclosure.

Rather than simply accepting this a problem in 10 or 20 dimensions, it helps to classify the dimensions in a reasonable scheme, if for no other reason that is will simplify our wrapper development. The variables in the first category are those that account for manufacturing variations, or "process" variables. Both supply voltage and temperature are "environment" variables, but to this category, for most circuits, we add bias currents. A third Category will include digital control bits, as well as analog control signals. While we will generally expect the "control"s to be static, when the change is large enough, we should record a new observation point. Output values, or characteristics of the circuit itself, or its signal path, we will classify as "measures." For the purpose of our prototype design verification environment, we will not factor "measures" into the observation volume.

2.2 Design Instrumentation For Enclosure Measurement

Following the example of PSL's early development [3], we will use a Verilog-A/AMS[21] language wrapper module to capture all of the data. By separating the function of data capture from the actual design, we allow for several representations of the design module to be used in the simulation, with minimal impact to the wrapper and the data format. In our methodology, we use a separate "vamsunit" as the source for our wrapper, but with all the statements hidden as comments from the simulator, we could put the specification in a behavioral model just as well.

By using Verilog, we eliminate dependence on a particular simulator. While all the development of this methodology has been done with the SPECTRE® simulator, we expect to be able to use these wrappers in AMS Designer, (and other Verilog simulators) with changes only where signals are represented as logic, or wreal.

The actual wrapper creates internal pins (and zero volt sources) for interfaces where current measures are needed, otherwise connecting the DUT directly to the external interfaces. In our prototype we need information about the process point being simulated. In order to keep this information in one file, we have added process monitor subcircuits to the model files, and instaniate them in our wrapper. The remaining behavior in the wrapper model is just that needed to capture the data we want in our observation point. It is expected that simulators, which already have access to this information internally, would eventually provide a native capability for a similar capability, once this methodology proves its value.

2.3 Observation Data Format

Four considerations drove the format selection. First, we needed a way to associate each measurement with the conditions under which it was measured, while encoding those conditions (supply voltages, temperature, process info, and control conditions) with every single measurement would insert an extreme amount of redundant data in the format. Second, we require a data format that is relatively free-form, as the conditions and measurements from different blocks, will vary significantly. Third, we wanted a format that could be produced from any relevant tool, not just these wrappers, so that this methodology could be easily extended to include the output of those tools. Finally, we want a format that can be viewed/ and interpreted directly, to simplify the development process. XML[5], a self-describing, tree structured data format, fits all of these needs, and can even be viewed directly as text, or in most web-browsers.

The observations are grouped into "points" , consisting of a set of conditions info, and the measurements associated with those conditions. For each unit recorded we want the SAME condition items recorded each time. For ease of use, we classify condition items into 3 categories; environment, process, and controls. In the XML tree structure, an observation includes 1 or more points, each of which have conditions (environment, process, controls) and measurements. Within each of these categories, there are specific types of information allowed/or expected.

The environment category allows; temperature, supply, reference (1 only) and bias elements. The process category allows the "section" element. The control category allows realsig, realdiffsig and the binsig elements. Elements in the measurement category include Iq, Vds, gain, amplitude and delay as shown in Listing 1.

Listing 1 Sample Observation XML file

```
<?xml version='1.0'?><observation><dut>
    <library>CHRONOS_top_sim</library>
    <cell>CML_demobuf</cell>
    <view>schematic</view></dut>
    <instance> DUT </instance>
    <vwrapper>$Header:
      /scn/projects/chronos/rev1/design_VSVAULT/CHRONOS_top_sim/
      CML_demobuf_vwrp/veriloga/veriloga.va,vs 1.6 2007/07/20
      11:58:35 jbdavid Exp $ </vwrapper>
    <point count="0" >
        <environ>
            <temperature name="Tsub" units="C">  50.0 </temperature>
            <reference name="Vgnd_a" units="V" atport="gnd_a" >
             0 </reference>
            <supply name="Vdd_a" units="V" atport="vdd_a" > 1.5 </supply>
            <bias name="Ibref500u" units="A" atport="iref500u" >
                500 u </bias>
        </environ>
        <process>
            <section type="cmos" name="mos">
                <tox units="Ang"><p> 28.5 </p><n> 28.1 </n></tox>
                <Cj units="pf/um^2"><p> 0.00136925 </p><n>
                    0.001376 </n></Cj>
                <dVth units="V"><p> 0 </p><n> 0 </n></dVth>
            </section>
            <section type="passive" name="scv_vars">
                <captol> 1 </captol><indtol> 1 </indtol><restol> 1 </restol>
            </section>
        </process>
        <control>
            <signal name="pwrdn" type="binlogic" size="1" > 0 </signal>
        </control>
        <measures>
            <Iq name="Iq_vdd_a" analysis="static tran" units="A" >
              0.14608 m </Iq>
            <Vdc name="Vbref500u" analysis="static" units="V" >
              47.1934 </Vdc>
            <gain name="out_in" analysis="tran" fromport="in" toport="out"
                units="none" samples="26" diff="true" >
                <r> 1.35985 </r><f> 1.35993 </f></gain>
            <delay name="out_in" analysis="tran" fromport="in" toport="out"
              units="s" samples="26" diff="true" >
              <r> 21.5284 p </r><f> 21.5278 p </f></delay>
            <amplitude name="out_in_in" analysis="tran" atport="in"
              units="V" samples="26" diff="true" >
                <diff><r> 0.294222 </r><f> -0.294222 </f></diff>
                <comn><r> 1.34987 </r><f> 1.34987 </f></comn>
              </amplitude>
            <amplitude name="out_in_out" analysis="tran" atport="out"
              units="V" samples="26" diff="true" >
                <diff><r> 0.400097 </r><f> -0.400122 </f></diff>
                <comn><r> 1.18602 </r><f> 1.18597 </f></comn>
              </amplitude>
        </measures>
    </point >
</observation>
```

2.4 Known Limitations

The measurement capability of Verilog-A is quite limited for small signal analyses. As a result, the current methodology focuses on the static (operating point) and transient analyses. This is adequate for a demonstration, but will need to be addressed if the methodology is to become widely adopted.

3. OSL – OBSERVATION SPECIFICATION LANGUAGE

Modeling the language on PSL, and the rest of Verilog, we start with the assumption that all OSL statements will (for now) be comments in a Verilog model that includes as a minimum, the module and port declarations. Since our prototype currently only builds a Verilog-A wrapper, we also require discipline declarations for each port. As shown in the example in Listing 2, OSL comments are recognized by the "// **msdv**" keyword at the beginning of a comment line. the OSL statement continues with the lable, followed by a colon (":"), then the command observe or measure. This is followed by the command type identifier, which may have a dot separated subtype. Next we have the parameter list (identified with the leading #) in set by name (explicit) format, followed with the connections for the measurement. The statement is ended with a semicolon. The statement may continue over multiple lines, where each continuation line starts with the standard "//" comment identifier. To allow multi-line statements, no other comments are allowed within an OSL statement.

The label is used in forming the name attribute in the observation statement. The connection information varies according to the type of measurement or observation point. Most measures have either an "**atport**" or "**fromport, toport**" set of connections which identify the ports at which the measurement is occurring. Some Measures have a "when" connection which is used to limit the measure to specific analysis types. Many measures have a "while" connection which contain an expression of control signals which determines valid control conditions for the measure.

Getting the process information available to the model required adding special subcircuits to the models files

with voltage sources set to the process parameter values we wish to monitor. In this case, the subcircuit name is the final parameter. Where the pin names of the subcircuit are not predetermined, the "what" connection specifies the parameters which will be monitored.

Listing 2 Example Vamsunit with OSL statements

```
// VerilogA for CHRONOS_top_sim, CML_demobuf, vaunit
// msdv dut  CHRONOS_top_sim.CML_demobuf:schematic ;
`include "constants.vams"
`include "disciplines.vams"
module CML_demobuf(out_n, out_p, gnd_a, vdd33, vdd_a, in_n, in_p, iref500u,
pwrdn);
output out_n, out_p;
inout gnd_a, vdd33, vdd_a,
input in_n, in_p, iref500u, pwrdn;
electrical out_n, out_p;
electrical gnd_a, vdd33, vdd_a,
electrical in_n, in_p, iref500u, pwrdn;
// msdv Tsub:     observe environ.temperature;
// msdv Vgnd_a:   observe environ.reference #(.units("V")) (gnd_a);
// msdv Vdd_a:    observe environ.supply #(.units("V")) (vdd_a);
// msdv Iq_vdd_a: measure dcamps #(.units("A"),.scalar("m"))
//         when(analysis("static","tran")) (vdd_a);
// msdv Vdd33:   observe environ.supply #(.units("V")) (vdd33);
// msdv Iq_vdd33: measure dcamps #(.units("A"),.scalar("m"))
//         when(analysis("static","tran")) (vdd33);
// msdv Ibref500u:   observe environ.bias #(.units("A"),.scalar("u"))
//              (iref500u);
// msdv Vbref500u:  measure dcvolts #(.units("V")) when(analysis("static"))
//         (iref500u,gnd_a);
// msdv mos:      observe process.cmos #(.tox_units("Ang"),.cj_units("pf/um^2"),
//         .dvth_units("V")) mos_pmonitor;
// msdv mos_33:   observe process.cmos #(.tox_units("Ang"),
//         .cj_units("pf/um^2"), .dvth_units("V")) mos_33_pmonitor;
// msdv dis_rpoly: observe process.cres #(.order({"p","n"}),.rsh_units("ohms/sq"),
//    .what({[disr_rppolywo,disr_rnpolywo],[disr_rppolys,disr_rnpolys]}))
//      dis_rpoly_pmonitor;
// msdv scv_vars: observe process.passive #(.count(3),
//      .what({captol,indtol,restol})) passives_monitor;
// msdv pwrdn:    observe control.binsig #(.vth(0.75)) (pwrdn);
// msdv out_in: measure sigpath.gain_delay #(.samples(25),.diff('TRUE),
//       .inv('FALSE),. firstsample (10), .units("none"), .timeaccy(5p)  ,
//       .dly_units("s"),.ampl_units("V"),.dscalar("p"),.stdy_dly(50p),
//       .start_time(100p)) while(!pwrdn) from(in_p,in_n) to(out_p,out_n);
endmodule
```

When specified in this way, the OSL statements serve as a concise definition of the information we need from each observation for the module being monitored without regard to the test-bench, level of hierarchy, the simulations, or even the type of actual circuit definition (behavioral, schematic, or extracted layout) used inside the wrapper for that simulation. From this we can build the wrapper itself.

4. AUTOMATING THE WRAPPER BUILD

While writing one or two wrappers to generate the desired XML would not be too troublesome to do by hand, the real value in this process comes when one has many elements used in many blocks of a larger design. By the time one is finishing the second such module, most of the process consists of copying a block of code, and substituting in the appropriate variables, which is a process not too difficult to automate.

4.1 Module design

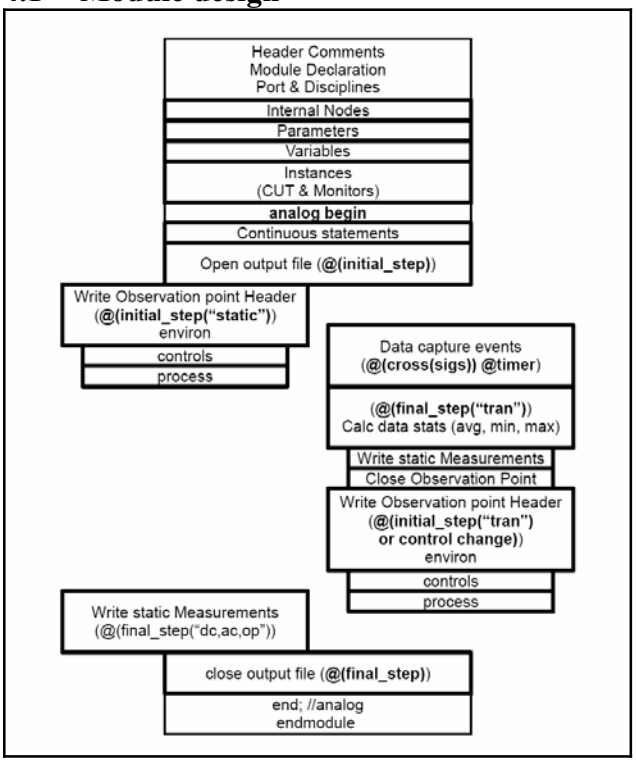

Figure 1 Outline of Auto-Generated Verilog-A Wrapper.

For our first prototype, we are doing only Verilog-A wrappers, so all of the behavior sits in the analog block. The structure of the wrapper is shown in Figure 1. Prior to the analog block we need the subcircuit instance for the DUT, and the instances for the process monitors. Where we want to measure terminal currents, we define an internal net, with a zero volt source. In the analog block we start with the behavioral zero-volt sources, and any variable expressions we want evaluated at all timesteps. Next we open the output file on the initial step, and dump the header info, and the observation data for the first observation point. Next we have events to capture transient measures, and an event to detect control changes that imply a new observation point. We wrap up with an

event at the final_step which writes any remaining measures, and then closes the output file.

4.2 Generating XML output

At the present time all xml data is created by the use of formatted print statements, assembled into the correct order by the wrapper generation program. Example statements for the Gain, Delay and Amplitude measures (all resulting from the sigpath.gain_delay measure statement labeled "out_in" in Listing 2) are shown in Listing 3. Checking the XML syntax validity was simply a matter of running a simulation, then loading the output files into a web browser. If the file doesn't' follow valid xml syntax rules, the browser error can guide debugging.

Listing 3 Formatted fstrobe statments for XML output

```
    $fstrobe(VFILE, "\t\t\t<gain name=\"out_in\" analysis=\"tran\" fromport=\"in\"
toport=\"out\" units=\"none\" samples=\"%d\" diff=\"true\" ><r> %g </r><f> %g
</f></gain>",
        Niout_in_m+1, Gain_out_in_m_r_sum/Nsout_in_m,
        Gain_out_in_m_f_sum/Nsout_in_m  );
    $fstrobe(VFILE,  "\t\t\t<delay  name=\"out_in\"  analysis=\"tran\"
fromport=\"in\" toport=\"out\" units=\"s\" samples=\"%d\" diff=\"true\" ><r> %g p
</r><f> %g p </f></delay>",
        Niout_in_m+1,  Dly_out_in_m_r_sum/Nsout_in_m * 1T,
        Dly_out_in_m_f_sum/Nsout_in_m * 1T   );
    $fstrobe(VFILE,  "\t\t\t<amplitude  name=\"out_in_in\"  analysis=\"tran\"
atport=\"in\" units=\"V\" samples=\"%d\" diff=\"true\" ><diff><r> %g </r><f> %g
</f></diff><comn><r> %g </r><f> %g </f></comn></amplitude>",
        Niout_in_m+1,
        Vdm_from_out_in_m_r_sum/Nsout_in_m,
        Vdm_from_out_in_m_f_sum/Nsout_in_m,
        Vcm_from_out_in_m_r_sum/Nsout_in_m,
        Vcm_from_out_in_m_f_sum/Nsout_in_m    );
```

4.3 Program design

Since a workable wrapper could be built by hand by copying sample code in the right order and substituting variables, we used this as the model for our program. All of the "template" code is defined in one file, which is parsed into a lookup hash as the first step of running the program. Once this is complete, each vamsunit file is parsed, for each OSL statement in the file, a substitution table is created, the appropriate templates are pulled from the hash, substitutions made and added to their section of the output file. When the "endmodule" statement is encountered, all of the sections are written to the output file in their proper order. Listing 4 shows the definition entries for the "observe environ.bias" statement.

Listing 4 Wrapper definition file entry for bias observation

```
// begin observe.bias vardeclare
  real %%VAR%%;
// end observe.bias vardeclare
// begin observe.bias nodedeclare
  electrical %%SIG_INT%%;
// end observe.bias nodedeclare
// begin observe.bias analogmain
    V(%%SIG%%,%%SIG_INT%%) <+ 0;
    %%VAR%% = I(%%SIG%%,%%SIG_INT%%);
// end observe.bias analogmain
// begin observe.bias tagbody
$fstrobe(VFILE, "\t\t\t<bias name=\"%%LABEL%%\" units=\"%%UNITS%%\"
    atport=\"%%SIG%%\" >  %g %%SCALAR%% </bias>",
    %%VAR%%*%%SCALEFCTR%% ); // print the current on DC runs
// end observe.bias tagbody
```

5. CASE-STUDY

MSDV verification score for CHRONOS_top_sim . CML_demobuf : schematic

category		about #	unique points		list of unique points
instance		1	DUT		
environment		9	1.6 3.5 500 u	5 1.5 3.3 500 u	0 1.6 3.5 500
supply.Vdd_a	units:\	3	1.6	1.4	1.5
supply.Vdd33	units:\	3	3.3	3.1	3.5
bias.Ibref500u	units:A	1	500 u		
temperature.Tsub	units:C	3	50	100	0
reference.Vgnd_a	units:\	1	0		
process		4	1.1 27.83 319.5	1 28.5 319.55	1 27.83 319.55
section.scv_vars.restol	type:p	3	0.9	1.1	1
section.mos.tox.p	type:c	3	29.17	27.83	28.5
section.dis_rpoly.disr_rpolyw	type:c	1	319.55		
section.mos_33.tox.n	type:c	3	71.5	65.5	77.5
section.scv_vars.captol	type:p	2	1.1	1	
section.dis_rpoly.disr_rpolys	type:c	1	260.09		
section.dis_rpoly.disr_rpolys	type:c	1	319.55		
section.mos_33.dVth.n	type:c	3	-0.08	0.08	0
section.mos.Cj.p	type:c	3	0.0014377	0.0013008	0.00136925
section.mos.tox.n	type:c	3	27.43	28.1	28.77
section.mos.Cj.n	type:c	3	0.0014448	0.0013072	0.001376
section.dis_rpoly.disr_rpolyw	type:c	1	260.09		
section.mos_33.Cj.n	type:c	3	0.0008645	0.0009555	0.00091
section.mos.dVth.n	type:c	3	0.025	-0.025	0
section.mos_33.tox.p	type:c	3	71.5	65.5	77.5
section.mos_33.dVth.p	type:c	3	0.12	-0.08	0.02
section.mos_33.Cj.p	type:c	3	0.0013304	0.0012037	0.001267
section.scv_vars.indtol	type:p	2	1.1	1	
section.mos.dVth.p	type:c	3	0.026	0	-0.026
control		2	1	0	
signal.pwrdn	type:b	2	1	0	
observations		24	1.6 3.5 500 u	5 1.5 3.3 500 u	1 1.6 3.5 500

Figure 2 Verification Score Spreadsheet for example circuit

To simplify the examples and results for the purpose of this paper, we use as our example a simple CML buffer of the type described in [7] which has two positive supplies, a bias current, and a power-down control. The primary measures of concern are the delay, gain, and input output amplitudes when not powered-down. Of course we always want to know the supply current, and bias current input voltage as well. The OSL statements for this model were shown in Listing 2, and the abbreviated output for a transient simulation was shown in Listing 1.

After writing an OCEAN script to run a set of corner simulations on this block, a perl script is used to generate

a results summary spread sheet and a verification scorecard. The Verification Score summary is shown in Figure 2, where it can be quickly seen that for the 24 unique points in the observation volume, only 1 bias current, only 1 poly-resistor section, only 2 values of captol, and indtol have been used and only 4 unique process points have been used. We can also see that the pwrdn control bit has seen both of its possible conditions.

As a guide for the verification effort, the verification team can keep in mind the following questions:

- Is the Unit level, Schematic Enclosure complete for Environment, process and controls?
- Are the Unit Level Behavioral results consistent with the Schematic design across Environment and controls?
- Are the Unit level Extracted results consistent with the Schematic design for the typical process and dominant control setting ?
- For the top level functional (ie with behavioral models) simulations is the control Coverage high?
- Are there Schematic or Extracted results for at top-level simulation? Does that include the dominant operation modes?
- Are the results for Schematic in higher-level tests consistent with the unit level tests?

6. CONCLUSIONS

A methodology for generating and using verification observations for blocks in a design by creating instrumentation wrappers that report observation data in XML format has been demonstrated. This method is now easily extended to multiple blocks for a larger chip design, allowing for capture of the entire design's "state of verification" Using this technique, design teams can easily get a handle on what simulations have been run, guiding their efforts towards improving their design verification into the most fruitful areas.

6.1 Future Work

There are already several areas where improvements can be planned. First, we need to add information about expected range to the "vamsunit"[see page 2] so that an expected total volume for the design element can be easily calculated, and possibly so that out of range measures can be reported during the simulation runs. Second, we need to be able to generate Verilog-AMS[2] wrappers for cases where the design uses AMS models in behavioral simulations. Third, we need to improve the reporting and summarization process, to generate more concise, and more useful results. The author expects to publish a separate paper outlining more details about the current status of the reporting methodology around the time this paper is actually published[8].

7. ACKNOWLEDGMENTS

I would like to thank Jay Singh (of Plato Networks) for many fruitful discussions as this methodology was developed, and the design team at Scintera Networks. Without our chip designs to work on, there would have been no necessity to serve as mother to these inventions.

8. REFERENCES

[1] *IEEE Standard Verilog® Hardware Description Language, IEEE Std. 1364-2001 rev C,* New York, 2001

[2] *Verilog-AMS Language Reference ManualVersion 2.2, November 2004* Napa: Accelera, 2004.

[3] *Assertion-Based Verification for Simulation Using PSL; Lecture Manual, Version 5.3, October 2004* San Jose: Cadence, 2004.

[4] G. Bonfini, M. Chiavacci, R. Mariani, R. Saletti, "A New Verification Approach for Mixed-Signal Systems", *BMAS2005 Web-only Publications:* http://www.bmas-conf.org/2005/web-only-pubs/BMAS2005_21.pdf .

[5] E.T. Ray, *Learning XML (Second Ed)* Sebastopol: O'Reilly, 2003.

[6] J. David, "Efficient functional verification for mixed signal IP**",** *BMAS 2004. Proceedings of the 2004 IEEE International Behavioral Modeling and Simulation Conference,* Oct. 2004, Pages: 53- 58.

[7] J. David, "Verification of CML circuits used in PLL contexts with Verilog-AMS", *Proceedings of the 2006 IEEE International Behavioral Modeling and Simulation Workshop,* Sept. 2006, Pages: 97-102.

[8] J. David, M. Singh, "A Methodology to Measure the Verification Gap in Analog and Mixed-Signal Designs" submitted for publication: *CDNLive,* September, 2007 http://www.cdnusers.org.

[9] D. Leenaerts, G. Gielen, R.A. Rutenbar, "CAD solutions and outstanding challenges for mixed-signal and RF IC design", *ICCAD 2001. IEEE/ACM International Conference on Computer Aided Design,* Pages:270-277.

[10] K. Kundert, H. Chang, D. Jefferies, G. Lamant, E. Malavasi, F. Sendig, "Design of mixed-signal systems-on-a-chip**"** *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,,* Vol.19, Iss.12, Dec 2000, Pages:1561-1571.

[11] T.E. Bonnerud, B. Hernes, T.Ytterdal, "A mixed-signal, functional level simulation framework based on SystemC for system-on-a-chip applications", *IEEE Conference on Custom Integrated Circuits*, 2001,Pages:541-544.

[12] S. Gupta, B.H. Krogh, R.A. Rutenbar, "Towards formal verification of analog designs**",** *Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on*, 7-11 Nov. 2004, Pages: 210- 217

[13] R.O. Peuzzi, "Verification of Digitally Calibrated Analog Systems with Verilog-AMS Behavioral Models", *Proceedings of the 2006 IEEE International Behavioral Modeling and Simulation Workshop,* Sept. 2006, Pages: 7-16.