Assertion Based Self-checking of Analog Circuits for Circuit Verification and Model Validation in SPICE and Co-simulation Environments

Lakshmanan Balasubramanian
Texas Instruments India Pvt. Ltd.
Bagmane Tech Park, 66/3,
Byrasanrdra,
C V Raman Nagar Post,
Bangalore – 560 093, India
+91-80-25099565
lakshmanan@ti.com

Pooja Sundar
Texas Instruments India Pvt. Ltd.
Bagmane Tech Park, 66/3,
Byrasanrdra,
C V Raman Nagar Post,
Bangalore – 560 093, India
+91-80-25048435
p-sundar@ti.com

Timothy W Fischer
Texas Instruments Inc.
12500 TI Blvd
Dallas, TX 75243, USA
+1-214-567-6681
tfischer@ti.com

ABSTRACT

In this paper we propose a methodology to simplify the verification process by creating a library of small, generic Verilog-A (VA) based assertion modules that can be connected together to form more complex checkers for any circuit. This serves as a good infrastructure for designers to easily build their own checkers. A Cadence infrastructure with schematic elements like symbols and forms are built to make the use of the library of assertions for a module level verification more intuitive and user friendly. Using the above infrastructure the required assertion based checkers can be embedded in the module design itself as an integral part and remain with the design hierarchy during the entire product life cycle at all integration levels of system-on-chip (SoC). This makes the module designs self-checking for verification purposes. This method enables to check the correctness of integration of the IP/design in question at the higher levels and automatic verification of these modules in the context of the SoC, avoiding or minimizing the manual checks at SoC level and / or limiting such checks to specific system scenarios that might not have been otherwise checked at the module level.

Categories and Subject Descriptors

B.7.2 **[Integrated Circuits]:** Design Aids – *Simulation, Verification.*

General Terms

Assertion Based Verification (ABV), Self-checking analog designs.

Keywords

Simulation, Verification, Assertion, Assertion based verification (ABV), Co-simulation, AMS, Self-check, verification and validation (VnV).

1. INTRODUCTION

There has been a trend of increasing Analog, RF and power management (PM) content integration [1] into SoC necessitated by solution cost, system flexibility, higher performance, and low power requirements. Complex SoCs of today with such high levels of integration necessitates thorough pre-silicon verification to achieve low operating costs by avoiding costly silicon tape-out iterations and ensuring high quality design.

All analog circuits are usually extensively verified by Spice simulations. The quality and functional sign-off of the analog circuits usually involve manual inspection of simulation results, waveforms against the requirements. There have been some automation available for analog simulation analysis based on the proprietary, vendor specific waveform calculators, post processing engines like the Cadence™ ADE [2] calculators and proprietary flows. There are also some run time support like the *.measure* statements of spice based tools like HSpice™ [3], HSIM™ [4], Nanosim™ [5] and Ultrasim™ [6].

Detailed spice based simulations at full-chip level are time consuming and may not be possible at all in many cases. Hence there is a necessity of higher levels of abstraction using analog behavioural models for the analog contents. This include logic level abstractions of analog interfaces using basic VHDL or Verilog, more accurate VHDL-AMS, verilog-AMS, system-C etc. The more accurate models need a separate analog engine for simulation. Hence there have been recent developments in using real number features of VHDL (VHDL-RN) for analog behavioural modeling that can be simulated with the basic VHDL simulation engine resulting in faster turn around times. These models are usually developed in early stages of SoC and module development and later updated regularly as significant maturity is achieved in the design. In all these stages the models have to be validated against the specification in early stages and against the actual design at mature stages of the design. The behavioural model validation (BMV) is usually done in a manual, iterative process where in the results of model simulations are checked against the spice simulation results. There have been some attempts to automate the BMV [7] like automation of test bench generation, test case equivalence but they still necessitate manual waveform inspection for final analysis and sign-off.

Manual inspection of simulation results to check that the analog circuit/model meets specifications, is time consuming and error-prone. This becomes more complex and cumbersome with the recent trends of increasing integration of analog, RF and power management contents in a SoC. Further all the checks for various specification goals have to be manually repeated at module level verification and all higher levels of integration making the whole

verification process error prone, iterative and poor coverage due to complexity of manual checking process.

Automation of this process aims at making analog verification more efficient in terms of time and accuracy. This paper proposes an assertion based self-checking methodology for model validation and circuit verification (VnV) based on generic library of assertions and infrastructure based on a popular EDA platform for analog design development and mixed-signal SoC integration. This paper primarily presents the assertion based self-checking methodology and its application and will not give the details of all the library of assertions.

The rest of the paper is organized as follows: Section 2 describes the concept of and motivation for self-checking design, Section 3 contains the practical implementation details of the proposed methodology, Section 4 discusses the proof-of-concept implementation, Section 5 has the results from a test case application, and finally Section 6 concludes this paper with key takeaways and scope and suggestions for improvements.

2. CONCEPT OF SELF-CHECKING DESIGN

2.1. ABV for Analog Circuits and Systems

Usually assertions are written in a HDL or verification languages like Specman e-language [8], PSL [9], OVM [10], VMM [11] to serve as run-time checker modules. However, to verify analog behavioral models and even analog circuits with such checkers, a co-simulation environment is required. Co-simulation overheads like the insertion and handling of interface modules pose a major challenge in keeping the verification cost like run time to reasonable levels. For analog module checks the tool specific native measurement command infrastructure may be used, but is a challenge when there is a need to support multiple simulation tools and environments.

Existing verification languages listed above and C-based systems currently extensively support circuit and system level checks [12, 13, 14] that are predominantly digital in nature, but are difficult to extend for checking analog quantities. They also need extensive support from EDA tool vendors to support AMS simulations. Existing methods and infrastructure necessitate manual, multiple developments or coding of the same set of assertions using different languages and tools to enable checks at various abstraction levels like the actual transistor level circuit, and behavioural models; and various design hierarchies like the module/IP level, sub-system, SoC and board level. Such a method as is evident involves repeated manual efforts, hence error prone and inefficient. Prior work [7] on assertions for analog design verification and BMV has been reported to use SystemVerilog (SV) [15] based assertion modules by creating a parallel hierarchy of the whole SoC design in SV linked to the original design hierarchy. It uses a proprietary script based automation to build the parallel hierarchy in an automated fashion. Though this methodology has been used successfully for analog IP level verification and BMV, it needs rewriting or manually porting the assertions for SoC level verification. Each time the automation engine generates a new parallel design hierarchy. It just falls short of the concept of "self-checking design with very minimal manual intervention".

2.2. Self-Checking Design for VnV

To overcome the difficulties observed in section 2.1 above, a concept of self-checking design is introduced as illustrated in Figure 1. In this methodology, the assertions are written and attached to design elements in a given development platform such that they reside with the design through the life cycle of the design. Such a concept enables what is called the "self-checking" designs. Such designs attached with all required assertion based checks enable automatic verification of the design context and functionality both at independent module level or upon integration at higher levels of integration, at SoC level and may be extended to board level or system level verification as well.

This methodology enables an intuitive and user friendly platform for verification. It also frees the designer from the familiarity and expertise with variety of languages, their syntactic nuances. Thus it allows him/her to spend value time in developing right test cases and checkers.

3. ARCHITECTURE AND IMPLEMENTATION

With the self-checking concept in focus, and given a design environment like Cadence Virtuoso Schematic Editor or any other similar platform, the assertions may be linked to a definite design element like the cell symbol or forms, and hence can reside in the design itself. Such design elements can be linked to various design cell views to enable generation of assertions in any language of interest like PSL, OVM, VMM, Specman e-language, VHDL, Verilog, VHDL AMS, Verilog AMS, Spice native formats, Verilog-A or any other vendor specific formats. Such a platform enables easy portability, interoperability across various design validation and verification platforms.

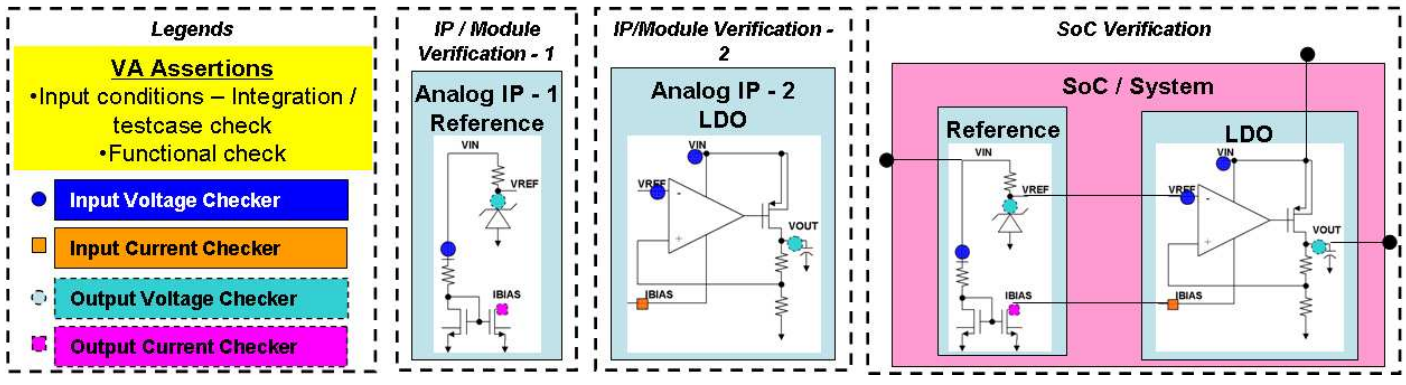

Figure 1. Concept of assertion based self-checking design

Though this method is not limited to any specific design environment, due to popularity among analog and mixed-signal SoC design community, the Cadence Virtuoso schematic editor is chosen as the platform for implementation. Spice and AMS co-simulation environments are the focus of this implementation.

3.1. Library of Assertions

This methodology mandates a pre-developed and validated library of assertions covering comprehensive set of basic checks. This library of assertions can be used to build any complex checkers for functional, electrical, reliability and manufacturability specification or requirement compliance.

Among the various languages that may be used Verilog-A based assertion modules can directly be interfaced with SPICE, without explicitly setting up a co-simulation environment, thereby saving on simulation run time for analog module level verification. Verilog-A is supported in most AMS verification tools and environments. While Verilog-A is chosen for assertions for analog transistor level design, VHDL/Verilog is used for digital portion of the design in RTL or higher levels of abstraction and VHDL-RN used for analog behavioural models. A basic set of assertions to enable following measurements are built in Verilog-A & VHDL-RN respectively and linked to unique "symbols":

1. Value of node potential (voltage) at any given time
2. Value of branch current at any given time
3. Signal transition information between specific threshold levels like the following:
    a. Rise time
    b. Fall time
    c. Number of transitions
    d. Occurrence of a specific type of signal transition in a specified time window
4. Delay between two identified events
5. Average, RMS value of voltage or current in a given time window
6. Peak / trough or maximum / minimum value of the signal level in a given time window.

The above listed measurements are in no way comprehensive enough, but form a set of representative, most often used functions. They can also be used to build other more complex functions.

Using right netlisting attributes and options, and features like the Cadence hierarchy editor, one can dynamically choose the language in which to netlist any given design module in an SoC.

4. PROOF OF CONCEPT IMPLEMENTATION

The proposed method was applied for the verification and behavioural model (BMOD) validation of a circuit used in the power management system of a complex mixed-signal SoC. In addition, this methodology has been applied as a means of checking over-voltage conditions for voltage dependent physical design rules such as metal spacing requirements.

4.1. Symbol Based Implementation

Figure 2 illustrates an example of how the symbol based assertions are used to build a specific checker of interest. In this particular instance the checker is built to check if, after power-up, the output of the test circuit is within the functional specification limits and if the input control signals are asserted / de-asserted at the required sequence meeting the necessary timing requirements. The scenario is illustrated in Figure 3. Also illustrated below are the VA codes for each of the functions mapped to unique symbols.

4.1.1. VA code examples

4.1.1.1. Digitizer module

This module checks for any event on the input signal *"x"* based on the parameterized threshold values *"VTH"* and *"VTL"*, sets the status outputs accordingly for further processing in addition to reporting a predefined message represented by *"msg"*. The status outputs *"rising_edge"* and *"falling_edge"* are defined as below.

rising_edge = 1, if V(x) >= VTH
0, if V(x) < VTH

falling_edge = 1, if V(x) >= VTL
0, if V(x) < VTL

The corresponding VA code segment is given below:

```
module digitizer (x, rising_edge, falling_edge);
input x;
electrical x;
output rising_edge, falling_edge;
electrical rising_edge, falling_edge;
parameter real VTH = 1.3;
parameter real VTL = 0.6;
real v_rising_edge, v_falling_edge;

analog
 begin
  V(rising_edge) <+ v_rising_edge;
  V(falling_edge) <+ v_falling_edge;

  @(initial_step) begin
   v_rising_edge = -1;
   v_falling_edge = -1;
  end

  @(cross(V(x) - VTH, +1)) begin
   v_rising_edge = 1.0;
   v_falling_edge = -1;
  end

  @(cross(V(x) - VTL, -1)) begin
   v_rising_edge = -1;
   v_falling_edge = 1.0;
  end
 end
endmodule
```

4.1.1.2. Minimum time checker module

This module checks for the delay between two events, in this case time elapsed since the occurrence of logic '1' in the input *"edge_1"* till the occurrence of logic '1' in the input *"edge_2"*, is greater than the predefined value represented by the parameter *"min_time"*. Any violation of the condition is reported as a predefined message represented by *"msg"*. The corresponding VA code segment is given below:

```
module min_time_diff (edge_1, edge_2);
input edge_1, edge_2;
electrical edge_1, edge_2;
parameter string msg = "message";
parameter real min_time = 30.0e-06;
real time_1;
```

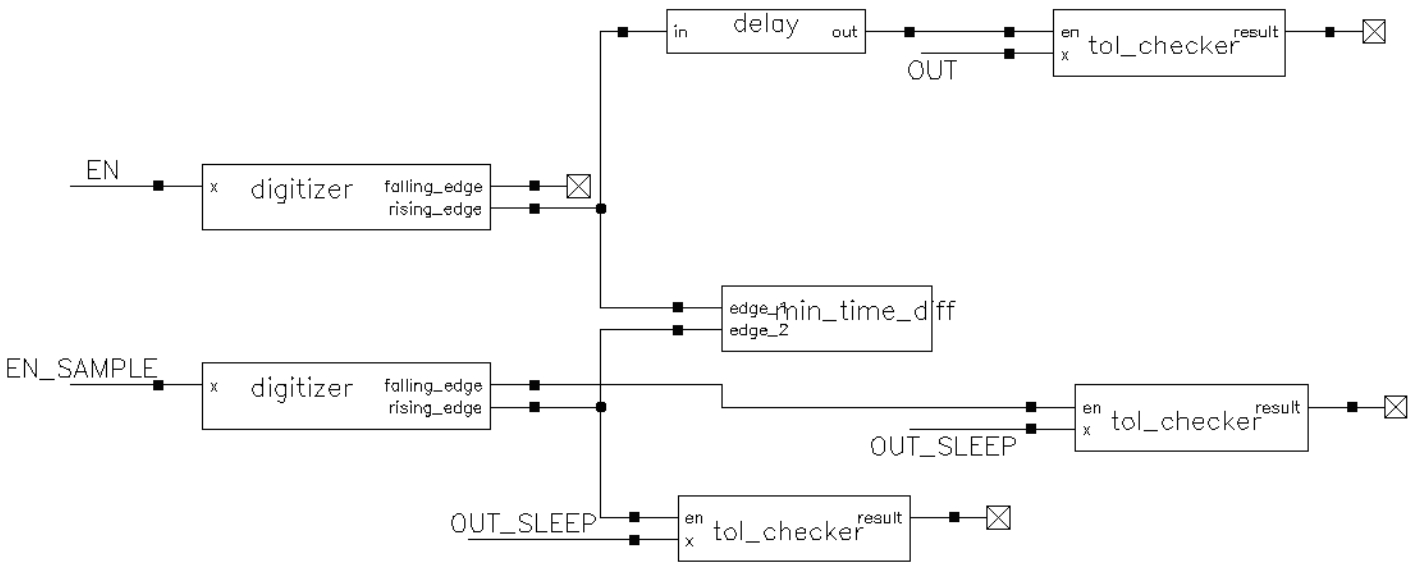

Figure 2. Symbol based checker for power-up and sleep functionality

```
time[160]  = 75506.786  911        4
EN rising edge at 8.22222e-05
GE_EN rising edge at 8.22222e-05
Rising edge on EN_SAMPLE at 8.22222e-05
Ready to check amt of sleep discharge..
Message: Actual 0.472403V @ 0.000082 Vs Expected LO:0.576000V,
HI:0.642000V
ERROR: minimum time diff check violated @ 0.000000s, Expected: 0.000030s
time[170]  = 82224.577  1.12       2
…
```

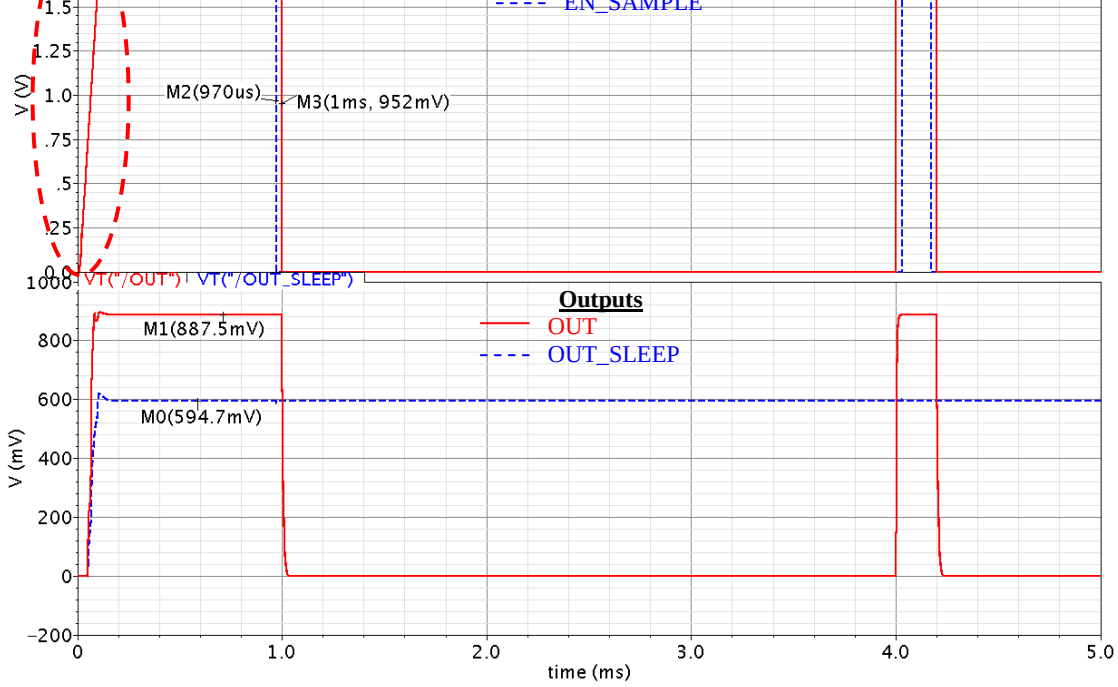

Figure 3. ABV scenario under test and results

```
analog
  begin
    @(cross(V(edge_1), +1))
      time_1 = $realtime;

    @(cross(V(edge_2), +1)) begin
      if($realtime - time_1 < min_time)
        $strobe(mg);
        $strobe("ERROR: minimum time diff check violated @ %fs,
Expected: %fs", $realtime-time_1, min_time);
    end
  end
endmodule
```

4.1.1.3. Voltage tolerance checker module

This module checks if enabled (*"en" = 1*) for the voltage level tolerance of the input signal *"x"* between predefined values represented by the parameters *"LO"* and *"HI"*. Any violation of the condition is reported as a predefined message represented by *"msg"*.

Message msg is reported if LO <= V(x) <= HI and if V(en)=1

The corresponding VA code segment is given below:

```
module within_limits (x, en, result);
input x, en;
electrical x, en;
output result;
electrical result;
parameter string msg = "message";
parameter real LO = 0.6;
parameter real HI = 0.6;
real v_result;

analog
  begin
    V(result) <+ v_result;

    @(initial_step) begin
      v_result = -1;
    end

    @(cross(V(en), +1)) begin
      if(V(x) > HI || V(x) < LO) begin
        $strobe(msg);
        $strobe("Actual %fV @ %f Vs Expected LO:%fV, HI:%fV\n",
V(x), $realtime, LO, HI);
        v_result = -1.0;
        end
      else
        v_result = 1.0;
    end
  end
endmodule
```

4.1.1.4. Time delay checker module

This module passes the value of input signal *"in"* to the output signal *"out"* after a predefined delay represented by parameterized variable *"del"*. The corresponding VA code segment is given below:

```
module time_delay (in, out);
input in;
output out;
electrical in, out;
parameter string msg = "message";
parameter real del = 100.0e-06;

analog
  begin
    V(out) <+ absdelay(V(in), del);
  end
endmodule
```

4.1.2. Monitoring Over-voltage for Physical Design Rule Requirements

While the core digital supply voltage is reduced at every technology node with respect to previous nodes, the I/O voltage requirements remain the same in order to support industry standards and legacy systems. Integration of analog and power management contents into SoC, direct battery interface (2V to 5V range or even higher) to portable / mobile applications and necessity for low power operation are all various other reasons for varied and multiple voltage and power domains in an SoC. Such requirements in addition to manufacturing constraints in UDSM technologies have necessitated voltage dependent physical design rules. For instance, the spacing requirement for two metals at 3.3V potential difference is higher than that of two metals that are within 1V of each other.

An approach has been developed to assign voltage properties to nets in the physical design for design rule checks using a voltage label on a pseudo layer in the layout. Since the layout data itself does not contain any real electrical information, a corresponding device, termed a DVR (Device for Voltage Recognition), is placed in the schematic. DVR enables checking for over-voltage conditions during simulation using the ABV methodology proposed in this paper. Equivalent connective placement of the assertion in the schematic and the voltage label in the layout is checked during LVS by extracting both as a pseudo device. A graphical overview of the full voltage dependent physical verification methodology is illustrated in Figure 4.

The DVR devices appear in the netlist as a Verilog-A assertion that compares the voltage during operating point, DC, and transient simulations to the property assigned to the device in the schematic. If the simulated voltage exceeds the assigned property, an error summary is printed at the end of the simulation. While Spice models do provide the capability to check for over-voltages at the device level, there was previously no low effort method for checking over-voltages on individual nets. Furthermore, the checks found in Spice that use voltage limits of devices tend to be overly verbose in the error reporting. Since the assertions are written in Verilog-A, the ABV methodology gives the flexibility to provide a simple error summary, and to tailor the checks and the error reporting to the needs of the business. This methodology ensures the correct use of compatible devices in a given voltage domain with reliable interconnectivity. It may be further extended for comprehensive checks for proper handling of multiple voltage and power domains and signal crossings.

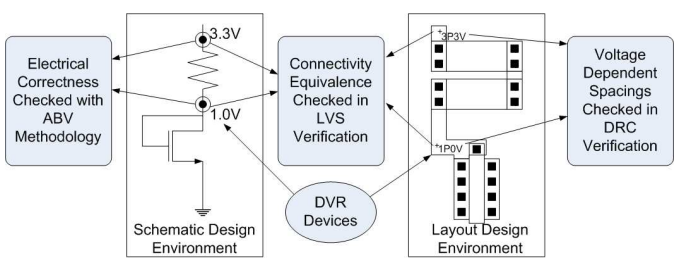

Figure 4. Full Verification of Voltage Dependent Physical Design Rules

4.2. Form Based Implementation

As is seen in Figure 2, the symbol based methodology has a limitation of building the asserting in the design itself by making physical connections to its ports. This can cause errors due to connecting the DUT or its components to various checkers. Manual errors in such connections can cause inadvertent shorts between different nets or signals of the design causing circuit malfunction. While the purpose of the assertions is to check the circuit functionality and correctness of context with minimal non-recurring effort, it should not cause additional issues to the design under verification (DUV) itself. To overcome this shortcoming, a connectionless, form based, interactive system is proposed. In this case an independent form that can be invoked from the schematic window GUI or from a terminal less symbol that can be instantiated in the design. Such a system is illustrated in Figure 5.

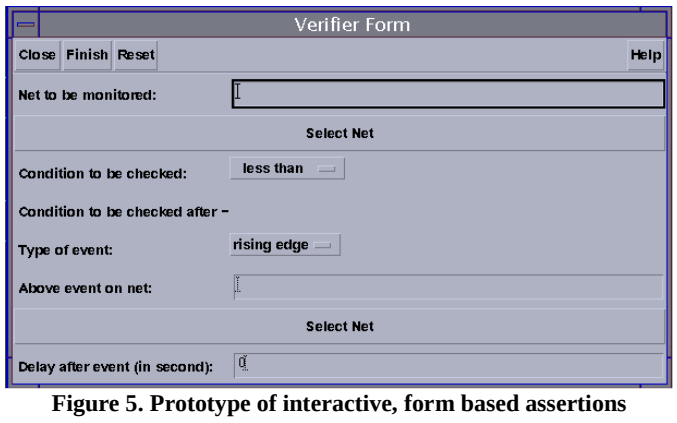

Figure 5. Prototype of interactive, form based assertions

This can be implemented in two different ways. The first one uses the same symbol based method discussed in section 4.1 above, but with no terminals attached to the symbol and with form based selection of nets for operation / observation. Such a form is attached as a property of the symbol. This can be called a symbol-linked-with-form based approach.

An alternate implementation style that is completely form based uses an automated, dynamic generation of checkers and node connectivity in the DUT enabled by appropriate scripts. In addition this also needs a proper database structure compatible with the design platform. In case of Cadence schematic based design platform, additional directory structure for each design unit or cell is to be created where in the necessary scripts, VA assertions and connectivity information are placed in addition to the existing schematic, symbol, layout and other views. In case of new design this new directory structure will be automatically created upon invoking the ABV form. Upon reinvoking, the forms for ABV of each design hierarchy will be preloaded with the details available under the respective directory structure. Skill scripts are used to build such an infrastructure.

4.3. Comparison of Symbol and Form Based Implementations

A symbol based implementation discussed in chapter 4.1 or symbol-linked-with-form based implementation discussed above enable platform independent truly self-checking design with out any infrastructural complexities. In contrast, the fully form based implementation discussed above needs proper infrastructure support like the database and automation scripts that are platform dependent and needs to be redeveloped or trimmed for different design platforms. Since form based implementation requires parallel data structure with minimal interference to existing designs it can easily lend itself to comprehensive automation and building of assertion based checkers by independent verification team. This is in compliance with contemporary and more popular organization of verification effort by industrial SoC design teams.

4.4. Circuit Verification & BMV

The implementation and application details in sections above discussed the circuit verification using the proposed methodology. This methodology can also be applied to BMV by choosing right netlisting language options for the assertions, namely VA for the transistor level circuit of the DUV and VHDL for the equivalent VHDL BMOD and running the simulations using Spice simulator and VHDL simulator respectively. The embedded assertions automatically will take care of checking both abstraction levels equivalently as long as the test bench equivalence is taken care of. Of course, necessary design guidelines have to be followed to allow seamless VHDL and spice netlisting. Some of the key guidelines are listed below, while they are not comprehensive as it is out of the scope of this paper:

1. No transistors or any technology components to be used at the level where BMOD netlisting is to be used.
2. Ensure the pin direction definition of all the designs consistently to have exactly identical comprehension in both Spice and BMOD contexts. Limit the use of INOUT direction to those pins, including power supplies and grounds, that really functionally behave as a bidirectional and not just because of the current flow direction as is usually comprehended by most analog designers.

5. RESULTS

The proposed methodology has a distinctive advantage of avoiding errors due to various existing manual process steps in VnV of AMS SoC, extensive reuse of library of assertions and reuse of assertions at module and SoC levels. Due to the use of VA assertions which are natively supported in many Spice based simulators and AMS co-simulators, it also has a specific simulation runtime advantage by avoiding use of co-simulation and insertion of several connect modules just to enable ABV. The proposed methodology ensures optimal simulation run times by mindfully choosing native assertion support for various levels of abstraction wherever possible.

Table 1. Simulation run time summary

Simulator	Speed-Accuracy settings	Medium of Assertions	Run time (minutes)
TISpice	Default	No	1
HSIM	Optimal for analog operation	No	4
TISpice	Default	VHDL[1]	∞
HSIM+VCS co-sim.	Optimal for analog operation	VHDL[1]	52
HSIM+VCS co-sim.	Optimal for analog operation & analog-to-digital interface	VHDL[1]	8
TISpice	Default	Verilog-A[2]	1.1
HSIM	Optimal for analog operation & analog-to-digital interface	Verilog-A[2]	4

[1] VHDL requires co-simulation feature
[2] Verilog-A is native to most Spice simulators

A comparative study of simulation run-times with verification using co-simulation under different settings versus verification using Verilog-A checkers is presented in the Table 1. All results have been tabulated for the same test condition of the test circuit, measuring its transient analysis for 5 ms. The results clearly show that native assertion support namely Verilog-A in this case, improves the run time by 2 to 50 times based on necessary simulator and accuracy settings.

6. CONCLUSION

Simulation based verification and behavioural model validation of mixed-signal designs and SoC would greatly benefit from assertion based VnV to overcome manual, iterative, error-prone waveform inspection. A novel assertion based self-checking concept is introduced in this paper. A proof-of-concept of implementation using a small set of often useful functions built using familiar design infrastructures that allows a set of checkers to be coded once, linked to a design element like a symbol or form. This infrastructure is used to build any complex checkers to enable circuit verification, behavioural model validation of AMS DUV. This automatically allows functional and context checks at SoC level.

One of the primary challenges is the creation of an adequately rich library, which requires a thorough study of typical checks needed during verification. It is especially useful in developing complex systems like the DC-DC switching regulator, and has the scope to be applied in diverse environments in the IC design flow. The proposed methodology has the following key advantages

1. Obviates the need for learning and familiarity with specific HDL or any other higher level language for coding the assertions
2. Enables building of assertions based checks for all specification requirements at module level and high order of reuse
3. Can work seamlessly with both Spice and co-simulation environments
4. Enables automated verification at module level and all higher level of SoC integration
5. Though an implementation using Verilog-A based assertion & uses Cadence design environment is illustrated, the concept can easily be extended for most other languages (HDL and high level languages) and design environments.

7. ACKNOWLEDGMENTS

The authors would like to thank Sandeep Tare and Dr. M. K. Srivas or Texas Instruments for initiation into the area of assertion based verification; Ranjit Kumar Dash, Sameer Dabadghav, Mohamad Kassem, Daniel Pickens and Keith Kunz of Texas Instruments for motivation, support and providing opportunity to apply the proposed concepts on real designs; acknowledge with thanks the support provided by Badrinarayanan Zanwar of Cadence Design Systems and Guha Lakshmanan of Texas Instruments for sharing several technical know-how and literature in the field of assertion based verification.

8. REFERENCES

[1] Buss, D.D., Chatterjee, A., Efland, T.R., Evans, B., Goodpaster, H.D., Haroun, B.S., Hellums, J.R., Krenik, W.R., Morton, A., Schichijo, H., Tsai, C.-Y., Vrotsos, T.R. 2000. DSP & analog SOC integration in the Internet era. IEEE Emerging Tech. Symp.: Broadband, Wireless Internet Access. Apr. 2000, 1-5. DOI=http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=916520

[2] Virtuoso Schematic Editor. Cadence Design Systems. DOI=http://www.cadence.com/products/rf/schematic_editor/pages/default.aspx.

[3] Hspice. Synopsys. DOI=http://www.synopsys.com/Tools/Verification/AMSVerification/CircuitSimulation/HSPICE/Pages/default.aspx.

[4] HSIM. Synopsys. DOI=http://www.synopsys.com/Tools/Verification/AMSVerification/CircuitSimulation/HSIM/Pages/default.aspxc.

[5] Nanosim. Synopsys. DOI=http://www.synopsys.com/Tools/Verification/AMSVerification/CircuitSimulation/Pages/NanoSim.aspx.

[6] Ultrasim. Cadence Design Systems. DOI=http://www.cadence.com/products/cic/UltraSim_fullchip/pages/default.aspx.

[7] Sharma, V., Lakshmanan, G., Tare, S., Dhamankar, S. 2008. Predicting the Correlation between Analog Behavioral Models and SPICE Circuits for robust SoC Verification. IEEE BMAS Workshop. Sep. 2008, 130 – 135. DOI=http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4751254

[8] IEEE Standard for the Functional Verification Language E. Standard IEEE 1647. Design Automation Standards Committee. Aug. 2008. DOI=http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4586409.

[9] PSL: IEEE Standard for Property Specification Language (PSL). Standard IEEE 1850, Design Automation Standards Committee. Apr. 2010. DOI=http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5446004.

[10] Open Verification Methodology (OVM). Cadence & Mentor Graphics. DOI=http://www.ovmworld.org/datasheets/OVM_Datasheet_12-17-07.pdf.

[11] VMM. Synposys & Accelera. DOI=http://www.vmmcentral.org/grg.html.

[12] Dammers, D., Domingues, C., Schollan, D., Vosskamper, L.M. 2009. Mixed signal system design verification accelerated with detector-based diagnostic method. IEEE BMAS Workshop. Sept. 2009, 66-72. DOI=http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5338887.

[13] Lammermann, S., Ruf, J., Kropf, T., Rosenstiel, W., Viehl, A., Jesser, A., Hedrich, L. 2010. Towards assertion-based verification of heterogeneous system designs. DATE. Mar. 2010, 1171-1176. DOI=http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5456985.

[14] Tutorial on analog assertions and analog value fetch. Version 0.8, June 2010. Cadence Design Systems, Inc. 2010.

[15] IEEE Standard for SystemVerilog-Unified Hardware Design, Specification, and Verification Language. Standard IEEE 1800, Design Automation Standards Committee, Feb. 2009. DOI=http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4796920.