Strategy and Environment for SOC Mixed-Signal Validation: A Case Study

Erik A McShane
Intel Corporation
5000 W Chandler Blvd
Chandler, AZ 85226
Telephone: 1 480 552 0425
erik.a.mcshane@intel.com

Srini Iyengar
Intel Corporation
3600 Juliette Lane
Santa Clara, CA 95054
Telephone: 1 408 653 8571
srinivasan.s.iyengar@intel.com

ABSTRACT
In this paper, we introduce the concept of full-chip mixed-signal validation (FCMSV) for SOCs, in which any top-level ports of an analog discipline are represented as real-valued signals for validation simulations. Secondly, it exploits SystemVerilog syntax, especially "real" ports, to create one testbench suite that spans a project from top-down development to final bottom-up implementation. Thirdly, it presents a global strategy to improve simulation speed by targeting specific analog blocks for replacement by BMODs. This methodology is a generic strategy independent of EDA tool. The methodology and environment outlined in this paper have been successfully applied to validate three successive projects.

Categories and Subject Descriptors
B.7.2 **[Integrated Circuits]:** Design Aids – *simulation, verification, methodology.*

General Terms
Verification, Performance, Standardization

Keywords
Full Chip, Mixed Signal, SOC, Strategy, Methodology

1. INTRODUCTION
Pre-silicon validation of large digital ICs benefits from mature RTL methodologies and environments, including extensive support for formal verification (FV) and assertion-based verification. However in System on Chip (SOC) products, as the analog or mixed-signal content grows in quantity and complexity robust methods give way to disjointed validation approaches. Since transistor-level co-simulation of analog circuits and RTL is often impractical, behavioral models (BMODs) and bus functional models (BFMs) coded in an HDL are required to reduce simulation time. Such modeling is also common during top-down development. Standard HDLs—Verilog-A (VA), Verilog-AMS (VAMS), SystemVerilog (SV)—are not universally recognized by EDA tools, which may lead to compromises in coding flexibility and demand complex environments that link tools from multiple vendors. Introduction of BMODs also requires another level of equivalency checking to ensure they adequately emulate the block functionality.

Previous efforts have concentrated on building mixed-signal validation (MSV) environments that resemble but stand apart from traditional full-chip validation (FCV) flows. Some require knowledge of specialized tool languages, like Cadence SKILL or Ocean scripts, which have a limited population of developers [1-5]. Others represent RTL as VCD files which prohibits validation of dynamic interaction or closed-loop behavior [6-7]. Finally, by implementing testbenches/behavioral models in VAMS (or similar) some projects lack a direct reuse of FCV collateral, which typically relies on Verilog and SystemVerilog HDL [8-10].

Moreover, although suitable for embedded or transparent analog functions—those which are invisible or hidden from chip or cluster I/O—most MSV and FCV methodologies do not comprehend so-called opaque analog functions. In this growing class of components, significant analog functionality is present at chip ports where it has strong real-time dynamic interaction with other components, either on the package or platform. The common practice of modeling an analog port, such as an I/O lane, as an equivalent 4-state digital signal becomes insufficient to rigorously validate its behavior.

The methodology in this paper makes several significant departures. Firstly, it introduces the concept of full-chip mixed-signal validation (FCMSV), in which any top-level ports of an analog discipline are optionally represented as real-valued signals for validation simulations. Secondly, it exploits SystemVerilog syntax, especially "real" ports, to create one testbench suite that spans a project from top-down development to final bottom-up implementation. Thirdly, it presents a global strategy to improve simulation speed by targeting specific analog blocks for replacement by BMODs. The methodology and environment outlined in this paper have been applied to three successive Intel SOC projects, each with significant mixed-signal content comprising both open- and closed-loop networks.

Section 2 reviews the challenges encountered in MSV and describes the proposed method. In Section 3 the methodology is discussed in detail, particularly its impact on a project's development cycle and milestones. The enhanced environment—scripting, structure, and coding styles—is described in Section 4. Based on the proposed approach, Section 5 presents results from the aforementioned projects, and Section 6 suggests future work to further enhance the capabilities. Section 7 summarizes the contributions.

2. FULL-CHIP MIXED-SIGNAL VALIDATION

2.1 Proposed Solution

Pre-silicon validation often assumes two phases: MSV of embedded analog functions and their mixed-signal interfaces, followed by FCV in which the analog behavior is either ignored, black-boxed, or replaced by pseudo-BMODs which capture approximate digital behavior. For example, an I/O lane may be emulated as a simple 4-state digital signal. However, an emerging class of analog functions has critical real-time interactions with external components on the package or on the platform. This includes such familiar examples as high-speed serial I/O and link training, PLL lock and clock references, and voltage regulators, as well as second-order effects like supply decoupling and package resonance. Additionally, on-package or on-die power supplies—e.g. switching regulators, linear and LDO regulators, and charge pumps—are another class of analog functions that are usually simplistically "assumed to be present" at all times in large scale cluster or half-system simulation environments. Often these functions are modeled with nearly ideal characteristics that lack any feedback or loading effects.

In all these cases, significant analog or power functionality is present at SOC pins—it is no longer sufficient to black-box this behavior: it must be represented at the pins as part of full-chip validation. Hence we propose a new terminology: full-chip mixed-signal validation (FCMSV). This approach is already common in analog and power regulation industries, but it represents a new paradigm for validating large SOCs.

Figure 1 illustrates the flowchart we developed for FCMSV [6]. Processes in orange are high-level tasks shared by the validation team, and RTL and circuit (CKT) designers. Green-shaded items are comprehended by a family of scripts with `simdrv` being the primary simulation tool, as described further in Section 4.1. A suite of testbenches and configurations is defined per a validation plan, which is oblivious of HDL choice and simulator engine. When executing a TB, `simdrv` parses the configuration. A build consisting of only Verilog and SV pulls in collateral from ordinary SVN or CVS repositories, and one of three engines is specified (Synopsys, ModelSim, or Cadence NC-Verilog). This flexibility accommodates various project preferences.

Results are available as logfiles for automated post-processing as well as graphically, which is useful during testbench definition.

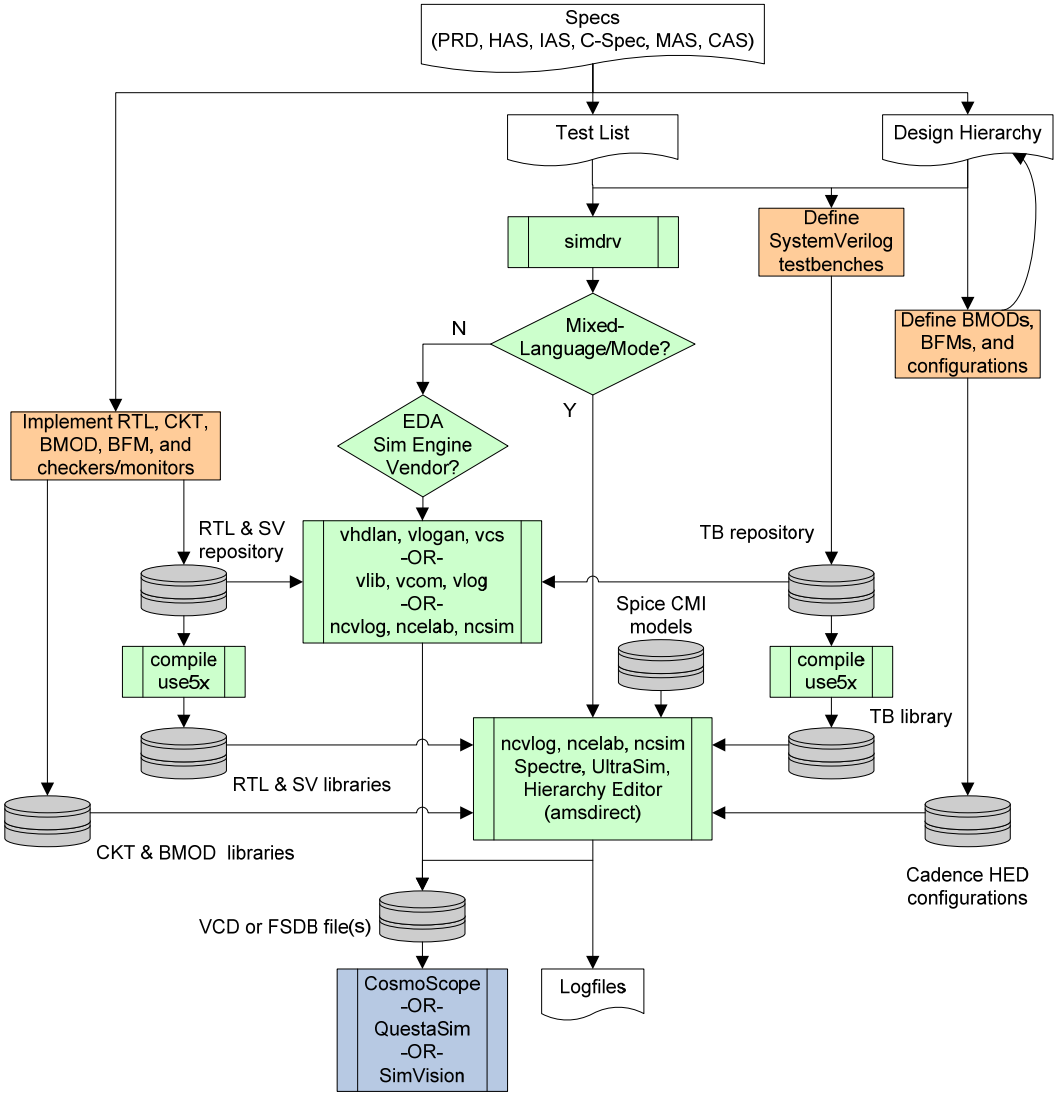


Figure 1. Full-chip (including mixed-signal validation) simulation flowchart.

For those testbench configurations that demand schematic or Verilog-A/AMS BMODs (a mixed-language or mixed-mode context), `simdrv` additionally compiles the Verilog and SV collateral into Cadence libraries (the compile `use5x` step). It then compiles, elaborates, and simulates a build, referring to the compiled Cadence libraries and those containing collateral from analog designers (their circuits and BMODs). The configuration of cellviews is controlled by a testbench config view, which permits instance- or cell-based selection of a netlist or BMOD view for every block in the design hierarchy.

Because this scripted flow recognizes multiple HDLs and engines, it is possible to execute MSV and traditional FCV using a single set of SV testbenches. And a single regression test list can provide digital and analog coverage since each test may stipulate a unique combination of view representations, and a specific simulator for optimal speed.

The proposed methodology achieves several key objectives:

1) Support common HDLs to enable maximum flexibility in coding BMODs for performance and portability,
2) Permit instance- or cell-based representation as a Spice netlist or one of several BMODs,
3) Enable validation during top-down development in addition to bottom-up implementation,
4) Be extensible to rigorous FV methods and assertion-based verification (PSL and SVA),
5) Simplify equivalency checking of BMODs and netlists through portable (shared) testbenches, and
6) Support multiple vendor engines to seamlessly transition between MSV and pure digital RTL FCV.

2.2 Applications to SOCs

Intel SOCs have a high degree of real-time interaction with platform components in both the analog and power domains. This impacts the validation strategy, since thorough coverage requires that the associated analog or mixed-signal blocks be stimulated (controlled and observed) from the testbench. Each SOC has substantially diverse analog circuit (CKT) content that operates in close coordination with RTL command and control logic. Small-signal circuits included bandgaps, current references, high GBW amplifiers, voltage and current sensors, thermal diodes and sensors, and LDOs; large-signal CKTs included high-speed I/O and equalization, ADC/DAC, DLL, PLL, transient detectors, and buck switching regulators. Therefore these SOCs are suitable examples to show the benefits of this flow.

3. METHODOLOGY

3.1 Top-Down Development

In our approach a single team owned FCV and MSV. This created opportunities to reduce resources and efforts by reusing testbenches and sharing BMODs. It also meant that a single validation team participated in early discussions with product architects to determine a comprehensive validation strategy. In particular, to facilitate top-down development (the activity during a project's architecture definition and leading up to its first major review milestone), we established two guidelines.

Guideline 1: The first element of this methodology was to adopt SystemVerilog as the only HDL for coding testbenches. Subsequently an RTL- or schematic-centric simulation could be performed just by customizing the hierarchy configuration. This greatly reduced the effort in coding and verifying the testbenches themselves. We made extensive use of SV real ports to control/observe analog behavior from the top level. This is distinct from other methodologies in which mixed-signal blocks are validated first, then black-boxed for FCV by encapsulating them in a wrapper to form a firewall that insulates FCV from "analog" behavior [4-5].

Guideline 2: The other element of our top-down methodology was to create cluster-level BMODs in SystemVerilog. This enabled the validation team to execute every testbench, prior to any design implementation, using ideal high-level representations of planned analog and mixed-signal blocks. (An approach which facilitates spec-driven top-down design.) As the hierarchy was developed and as underlying RTL and CKT collateral was implemented for subsequent project milestones, the cluster BMODs could be replaced with more detailed representations. Also, the cluster BMODs were essential in providing more flexibility in testbench configurations, especially in validating slow behavior and infrequent events; this is discussed further in Section 5.3.

3.2 Bottom-Up Implementation for RTL and Custom Circuits

As a design progresses through various milestones, the BMODs of analog circuits evolve to capture more accurate behavior. This is denoted by BMOD "Quality Factors," as shown in Table 1. Per a specification-driven design methodology, a preliminary model (BMOD0.5) captures only ideal input-output behavior. Timing and latency are approximated and digital control signals (e.g. enable or power-down) are modeled as simple on/off functions without comprehending charge and decay attributes. By its intermediate milestone a circuit is sufficiently mature to demand more detail in its corresponding BMOD. In particular, it models dependencies on supply and reference voltages, captures trim/calibration behavior, and exhibits accurate timing and slew. The final model (BMOD2.0) is updated to comprehend dependency on reference currents and other enhancements based on simulation profiling.

Table 1. Definition of BMOD quality factor.

BMOD development to Quality Levels	BMOD0.5 (Preliminary)	BMOD1.0 (Detailed)	BMOD2.0 (Final)
Basic behavior	YES	YES	YES
Detailed behavior	no	YES	YES
Dependence on trim code(s)	no	YES	YES
Dependence on supply voltage(s)	no	YES	YES
Dependence on reference voltage(s)	no	YES	YES
Dependence on reference current(s)	no	no	YES
Modeling of PVT corners	no	no	no

The BMODs in general do not comprehend PVT corners—ensuring adequate circuit performance margin is the responsibility of CKT designers using conventional analog simulators; FCV and MSV are too cumbersome to verify analog design corners.

Another significant activity during bottom-up implementation is maintaining the equivalency of a design block and its corresponding BMOD. This is especially challenging since no formal verification methods exist for analog circuits. We identify five possible checks, shown in Table 2, in which "1" is closest to a transistor-level representation and "5" is the highest level of abstraction. In general, the validation team owns equivalency checks for abstraction level 5; CKT designers are responsible for checking and maintaining all lower abstractions.

Equivalency “1” is comparing transistor-level and Verilog-A behavior in an analog simulator (most rigorous comparison of actual and modeled behavior); equivalency “3” similarly compares a VAMS BMOD in a Verilog-AMS environment. Some Intel devices include proprietary simulation models: to port a schematic for Cadence Spectre or UltraSim simulation, equivalency “2” verifies the same netlist using CMI and the proprietary models. At the cluster level, equivalency “4” is between an SV BMOD and an alternate representation, such as VA or VAMS. Finally, to enable multiple simulation engines, equivalency “5” verifies the same SV model in Cadence and Synopsys (or ModelSim) to isolate any issues with vendor-specific HDL interpretation.

Table 2. Equivalency checks for mixed-signal data representation.

Sim Engine / Data Type	Custom Spice (custom)	Spectre / Ultrasim (CMI)	Virtuoso -AMS (CMI)	Synopsys -VCS (BMOD)
Transistor	1, 2		3	
Verilog-A / AMS	Verilog-A	Verilog-A	4	
SystemVerilog			5	

4. ENVIRONMENT

4.1 Scripting

The validation and design flow includes a unified directory structure (with revision control in SVN or CVS). Synthesis, simulation, linting, and tagging are all controlled by the same underlying set of Perl scripts, Makefiles, and shell scripts in CSH, Awk, and Sed. The scripts had the capability:

1) To recognize various HDLs such as Verilog-A/AMS and SystemVerilog, and
2) To comprehend the Cadence design environment in addition to Synopsys and ModelSim.

The validation environment is initialized in two steps. First, a Cadence analog design environment is invoked and the schematic design libraries are identified. Then additional EDA tools are configured for RTL design, verification, synthesis, etc. We combined these actions in a single `setup_fcmsv` script. Afterwards there are three categories of scripts to enhance productivity and provide automation. Most rely on gmake which processes relationships in a global sim_make.rules file.

Script set 1: Automated Code Generation from Templates

`mkmodule and mktest` – These Perl scripts use parameterized templates to create the necessary directory structure for a new Verilog module or testbench, respectively, populate it with file templates, and perform a check-in to the project repository.

Script set 2: Compilation Scripts

`gmake` – The Makefile utility may be invoked directly to compile, elaborate, and simulate individual modules or testbenches. The simulator engine (vcs, mti, or ncv) may be specified on the command line. For the special case of NC-Verilog, the `-use5x` flag is added, such that the compiled result is also a valid Cadence library, which conforms to the library:cell:view (LCV) or 5X format. See also Section 4.2.

`buildmodel` – This Perl script builds all modules, from either a release area or user area, based on a revision symbolic tag. By default it then compiles the model for either Synopsys or ModelSim.

Script set 3: Simulation

`simdrv` – A Perl script to run a testbench in batch mode. Optionally it may perform a full regression test if a list is provided in the `regression_test.list` command file.

```
> simdrv -list regression_test.list -build
```

`simdata` – A Perl script to parse the results of a regression run, which stores a result as simdata.results. A sample result of four regression tests:

```
------------------------------------------------------------------------
Test                    Status                Cycles  Cyc/Sec Errors  Host
------------------------------------------------------------------------
clkrst_unit_s000_cfg.nbl PASS                 0       0       0       chlsivro01
enaflt_unit_s000_cfg.nbl FAIL-SM              0       0       35      chlsivro01
smb_unit_s000_cfg.nbl    STARTCMP-NOT-FND     0       0       0       chlsivro01
jtag_unit_s000_cfg.nbl   PASS                 0       0       0       chlsivro01
******************** SIMDATA LOG INFORMATION ********************
TESTS RUN: 4   PASSED: 2   FAILED: 2   RUNNING: 0  DEAD: 0   PASSING CYCLES: 0
ERRORS : 35
WARNINGS : 39
USER TIME : 301.05          SYSTEM TIME : 1.08
AVG CYC SEC : 0             PERIOD : 17.86
```

4.2 Directory Structure

A skeleton of the environment directory structure is shown in Figure 2. Generic templates, tools, and models are indicated, as well as the location of project design collateral, with placeholders denoted by italics.

To enable MSV using Cadence tools (Virtuoso-AMS and UltraSim) the RTL and SV testbench collateral must be identified as valid libraries. Assuming the collateral has been compiled-in-place in 5X format (see Section 4.1), the gray fields indicate the additional files—`cds.lib` and `hdl.var`—to declare the collateral to the Cadence library manager. Effectively, each unit, each BFM, the core, and the system appear as distinct libraries; likewise for each unit- or system-level test.

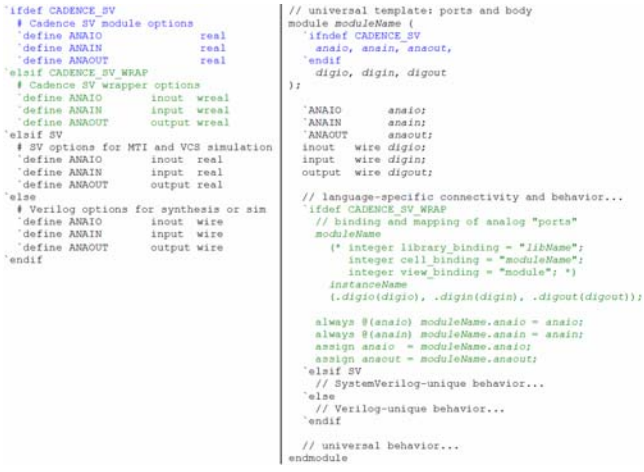
```
core/
    src/
        Makefile
        cds.lib, hdl.var
        core.v
    syn/
system/
    src/
        Makefile
        cds.lib, hdl.var
        my_tb.sv
    tests/my1_unit/my1_unit_s001/
            Makefile
            cds.lib, hdl.var
            cds_globals.vams
            my1_unit_s001.sv
            tst_tb.sv
units/my1_unit/
        src/
            Makefile
            cds.lib, hdl.var
            unit.v
        tests/my1_unit_u001/
            Makefile
            cds.lib, hdl.var
            cds_globals.vams
            my1_unit_u001.sv
            tst_tb.sv, unit_tb.sv
```

```
common/templates/
    templates/
        {model|module|systest|unittest}_cds.lib
        {model|module|systest|unittest}_hdl.var
        {module|systest|unittest}_Makefile
        newmodel.sv
        newmodule.v
        systest.sv
        tst_tb.sv
        unit_tst_tb.sv
        unit_unit_tb.sv
        unittest.sv
models/my1_bfm/src/
            Makefile
            cds.lib, hdl.var
            my1_bfm.v
release/
tools/bin/
    bin/
        buildmodel
        mkmodule
        mktest
        setup_fcmsv
        sim_make.rules
        simdata
        simdrv
```

Figure 2. Directory structure. (Grayed fields are enhancements to enable Virtuoso-AMS recognition.)

4.3 Coding Styles

We developed a single Verilog template which supports Verilog and SystemVerilog syntax, and can be correctly parsed by Synopsys, ModelSim, and Cadence, by defining a compact set of macros. All RTL modules were coded using this template, including the chip top-level, such that the declaration of any analog ports could be toggled between a scalar wire (for synthesis) and a scalar real (for MSV).



```
`ifdef CADENCE_SV
  # Cadence SV module options
  `define ANAIO               real
  `define ANAIN               real
  `define ANAOUT              real
`elsif CADENCE_SV_WRAP
  # Cadence SV wrapper options
  `define ANAIO        inout  wreal
  `define ANAIN        input  wreal
  `define ANAOUT       output wreal
`elsif SV
  # SV options for MTI and VCS simulation
  `define ANAIO        inout  real
  `define ANAIN        input  real
  `define ANAOUT       output real
`else
  # Verilog options for synthesis or sim
  `define ANAIO        inout  wire
  `define ANAIN        input  wire
  `define ANAOUT       output wire
`endif
```

```
// universal template: ports and body
module moduleName (
  `ifndef CADENCE_SV
    anaio, anain, anaout,
  `endif
    digio, digin, digout
);

  `ANAIO       anaio;
  `ANAIN       anain;
  `ANAOUT      anaout;
  inout  wire digio;
  input  wire digin;
  output wire digout;

  // language-specific connectivity and behavior...
  `ifdef CADENCE_SV_WRAP
    // binding and mapping of analog "ports"
    moduleName
      (* integer library_binding = "libName";
         integer cell_binding = "moduleName";
         integer view_binding = "module"; *)
      instanceName
      (.digio(digio), .digin(digin), .digout(digout));

    always @(anaio) moduleName.anaio = anaio;
    always @(anain) moduleName.anain = anain;
    assign anaio  = moduleName.anaio;
    assign anaout = moduleName.anaout;
  `elsif SV
    // SystemVerilog-unique behavior...
  `else
    // Verilog-unique behavior...
  `endif

  // universal behavior...
endmodule
```

Figure 3. Universal template for Verilog and SystemVerilog that comprehends usage by Virtuoso-AMS.

Figure 3 shows an example template (values in italics are placeholders). If no macro is defined, the interpretation defaults to Verilog syntax suitable for simulation in ModelSim or Synopsys, or for synthesis. Adding an "SV" flag causes the module to be interpreted as SystemVerilog, and ports with an analog context are redefined as reals. In this way a scalar port can be used to exchange real-valued data, which is consistent with its analog context (e.g. an analog value of 1 mA is passed as 0.001). The "SV" flag is sufficient for ModelSim and Synopsys, since both engines are fully compliant to IEEE P1800 SystemVerilog.

Cadence, however, is not P1800-compliant and requires a two-level approach in which the original SV module is wrapped in Verilog and its ports are mapped using hierarchical notation. By specifying both "SV" and "CADENCE_SV" flags, first the module is compiled to redefine the "analog" ports as internal real signals (see blue text of Figure 4). Then a separate Verilog wrapper must be compiled. Using the same template with the "CADENCE_SV_WRAP" flag (green text of Figure 4), the module is compiled as Verilog and it includes explicit bindings to its embedded SystemVerilog-like instance.

This discussion is based on an older Cadence release. In a more recent release, Cadence has been advocating a strategy called Real-Value Modeling (RVM), in which the wreal type is recognized as a valid port type and the separate Verilog wrapper is no longer required.

The resulting hierarchy of a Cadence SV-like module is illustrated in Figure 4, an example of a bandgap reference testbench shown in Hierarchy Editor (HED). It is evident that multiple HDLs may be supported in a single testbench configuration (this example includes schematic, spectre netlist, Verilog-AMS, Verilog, and SystemVerilog).

To represent the bandgap block in SystemVerilog, the cell:view (LCV) binding of its fundamental BMOD is:

```
ssiyenga_sv  : sim_bgref_tb : module
```

and the LCV of its Verilog wrapper is:

```
ssiyenga_ams : sim_bgref_tb : module
```

Note that since the BMOD and its wrapper have the same cell and view names, they must be compiled into different libraries. In this example, the top-level testbench is represented as a schematic but any abstraction, including Verilog or SystemVerilog, is permitted.

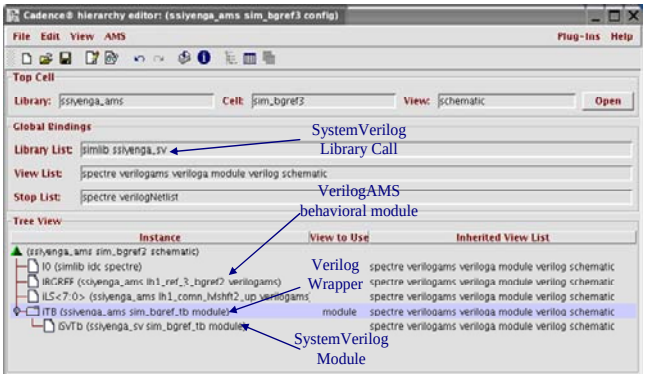


Figure 4. Example of instantiating SystemVerilog modules in Virtuoso-AMS HED.

Table 3 itemizes recommended cellviews within Cadence-AMS. In addition to typical schematic and symbol views, Verilog-A/AMS have unique cellviews for models and stubs. The "module" view denotes Verilog syntax, although as shown above explicit binding rules may be used to emulate SystemVerilog.

Table 3. Recommended Virtuoso-AMS cellview labels.

View Type	Description
schematic	view showing transistor level circuit representation of a given unit
symbol	view showing IOs/ports of a given unit
veriloga	BMOD view created using Verilog-A tool type
verilogams	BMOD view created using Verilog-AMS tool type
module	BMOD view created using SystemVerilog/Verilog option/tool type
stub_va	Null BMOD view created using Verilog-A tool type
stub_vams	Null BMOD view created using Verilog-AMS tool type
stub	Null BMOD view created using Verilog tool type

5. RESULTS

5.1 Analog Regression Results

We applied this flow to perform analog regression simulations, including parametric sensitivity. This was extremely useful as the design progressed toward complete schematics since violations due to unexpected sensitivity could be detected as early as possible. Table 4 shows an example of parametric regression coverage for a voltage supply rail. Columns 10-17 list the independent variables; note that these include digital and analog qualities, all controlled from an SV testbench. The simulated results, listed in columns 1-9, include common voltage regulator metrics like setpoint voltage (Vnom) and efficiency (Eff).

Table 4. Analog regression simulation example with parametric sensitivity.

Vnom (V)	Eff (%)	Vrpl+ (%)	Vrpl- (%)	Vovr (%)	VsftsDev (V)	PhaseSW	HSovrFlt	Shoots	disbal	ibal_pd	novip	softsw	loadMinMax	ENAI mvp4	Vimvp (V)	Rcpuvss (ohm)	Last Run Time
1.053	84.2	3.6	3.2	2.0	0.071	23	3	0	0	0	0	1	0	0	0.7875	0.000001	Nov 8 2007 20:12:48
1.053	84.6	3.6	3.3	1.2	0.097	26	3	0	0	0	0	0	0	1	0.6168	0.000001	Nov 8 2007 23:24:18
1.054	79.8	4.4	3.0	2.1	0.050	23	3	0	0	0	0	0	0	1	0.6168	0.05	Nov 9 2007 02:37:57
1.054	79.6	4.5	2.9	3.0	0.041	23	3	0	0	0	0	0	0	0	0.7875	0.05	Nov 9 2007 05:51:26
1.052	84.6	7.5	6.2	2.2	0.078	12	3	0	0	0	0	0	1	0	0.7875	0.000001	Nov 8 2007 20:13:19
1.052	84.2	7.6	6.0	2.0	0.069	12	3	0	0	0	0	1	1	0	0.7875	0.000001	Nov 8 2007 22:08:18
1.052	84.1	6.0	6.3	2.7	0.128	7	3	0	0	1	0	1	1	0	0.7875	0.000001	Nov 9 2007 00:00:39
1.052	84.5	8.0	6.5	2.1	0.079	12	3	19	0	0	3	0	1	0	0.7875	0.000001	Nov 9 2007 01:41:40
1.052	84.1	8.4	5.7	2.1	0.076	12	3	0	0	0	3	1	1	0	0.7875	0.000001	Nov 9 2007 03:38:08

This example combined analog and power functions with significant closed-loop digital control. Initially the testbench configuration consisted of RTL and high-level BMODs, which were gradually replaced by more accurate models. Ultimately, the configuration shifted to RTL and schematics with relatively few BMODs. The evolutionary process provided weekly checkpoints to detect bugs in interfaces or implementation.

5.2 Effective BMOD Strategy

Given the complexity and quantity of analog blocks in SOCs, a validation strategy must identify a minimal set of BMODs necessary for effective simulation coverage. Figure 5(a) summarizes the unique analog block counts of all three projects. "Leaf" blocks are those at the lowest level of hierarchy, comprised only of transistors and other schematic primitives. "Structural" blocks establish the hierarchy by instantiating and connecting child blocks. Projects A and B had similar proportions (~0.66) of leaf and structural blocks; Project C had a relatively "deeper" hierarchy reflected in its smaller 0.55 ratio.

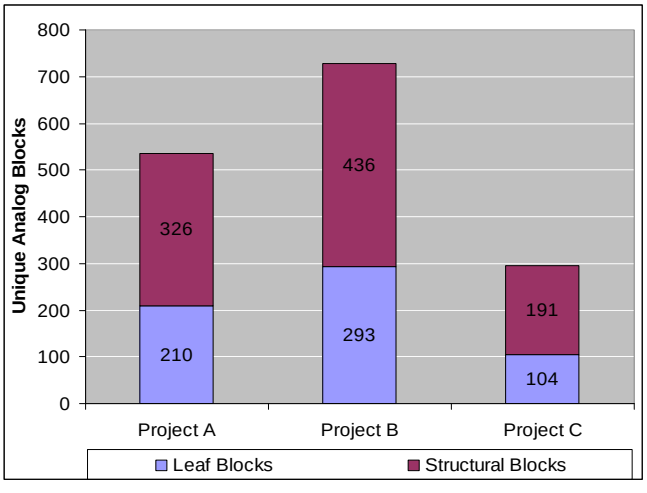


Figure 5(a). Circuit block count.

Figure 5(b) presents the BMOD statistics for the same projects. It shows a generational improvement in validation strategy and planning. In all three projects nearly 50% of leaf blocks had corresponding BMODs as dictated by their validation plans; however some redundant models were coded too. (Redundancy implies a single block has multiple BMOD representations, such as Verilog-A and Verilog-AMS, or Verilog-A and stub, etc.) In Project A, 10% of leaf cells required alternative models, which was reduced to 0% redundancy by Project C. The prevalence of BMODs at the structural level has no clear trend. This is expected considering that the projects had very different functions and design hierarchies. But the trend in modeling improvement is even more pronounced: 24% of Project A's structural blocks had redundant models, which was reduced to just 9% in Project C.

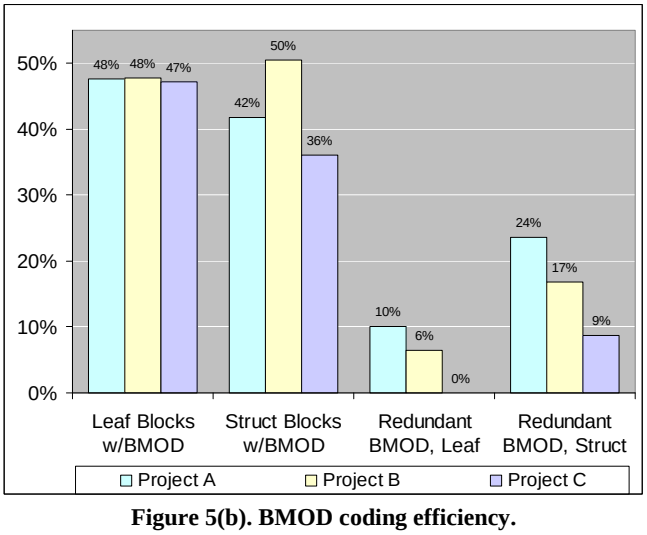


Figure 5(b). BMOD coding efficiency.

This improvement in modeling efficiency is due to generational learning about which blocks critically impact simulation speed and how to define a single behavioral representation to satisfy all testbenches.

5.3 Correlation between Hierarchy, Partitioning, and Simulation Speed

Proper partitioning of analog design hierarchy can facilitate much faster simulations without sacrificing transistor-level accuracy. Many analog leaf blocks, especially those which include passive networks or trim functions, have a very high node count if captured as a flat schematic. Examples include opamps, references, comparators, and data converters. By collecting each distributed network into a single element instead, each element may be modeled by a lumped equivalent; there is no impact to layout artwork or LVS, only one more level of design hierarchy.

Figure 6 shows the impact on the simulation executable of two projects as a function of design partitioning. All results are normalized to simulations consisting of RTL and a complete transistor netlist (Spice netlist column), which is the baseline for simulation accuracy and actual simulation time. To reduce simulation time, a common practice is to replace all schematics with BMODs. As shown (in column BMODs w/lumped RC) the speedup is significant, with CPU usage reducing by a factor of 9 and 25 for Projects B and C, respectively. However, creating a complete set of BMODs may demand a substantial coding effort, and it potentially masks bugs due to inadequate model fidelity.

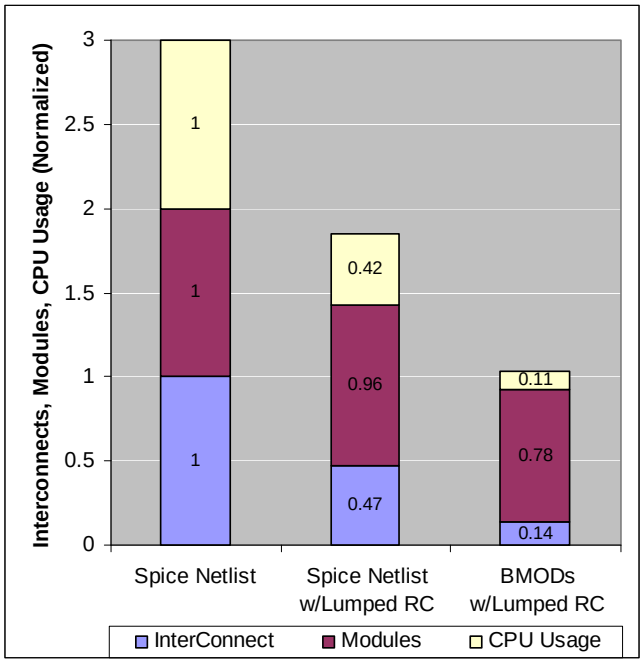
We found that simply replacing resistor and capacitor networks with lumped elements, and leaving the analog netlist otherwise unchanged, reduced simulation time by 30-60% in both examples (column Spice Netlist w/lumped RC). The module count relative to the Spice Netlist baseline was almost unchanged as expected, but interconnect count was nearly halved. This suggests a strategy for design hierarchy partitioning:

1) If a leaf cell contains a high node-count network (e.g. resistor matching), capture it as a single symbol and push that schematic complexity down the hierarchy as a new leaf cell.
2) Create an equivalent lumped-element schematic or BMOD.
3) For a faster simulation use those equivalent cellviews along with selective BMODs of those blocks less essential to the testbench coverage goal.

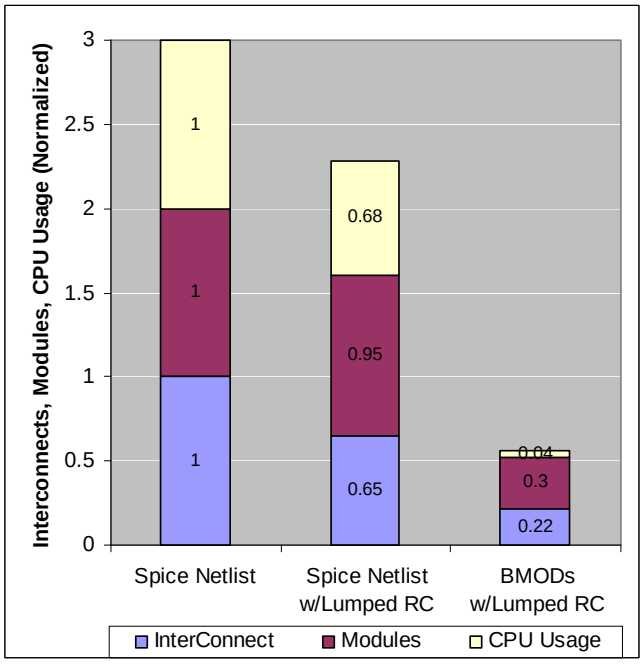
This strategy of selective BMOD usage is more effective for MSV than a BMOD-only approach:

1) Greatly reduces the number of BMODs which must be coded,
2) Retains as much transistor-level accuracy as possible, and
3) Can achieve nearly the same simulation speed.

Another example of partitioning is the tradeoff between high-level (cluster) BMODs and a combination of detailed block BMODs and schematics in terms of simulation fidelity and speed. For Project B, when simulating for 1 millisecond, a configuration comprised of schematics and detailed block-level BMODs required 80 minutes to complete. The same simulation based on relatively few SV BMODs of entire analog clusters completed in less than 4 minutes, a 20× improvement. Our validation plan leveraged both configurations for different testbench goals. The slower, high-quality representation was used to evaluate critical datasheet specs, whereas the faster representation was selected to validate low-bandwidth behavior, such as control algorithms and platform interaction.



(a): Project B.



(b): Project C.

Figure 6. Effect of optimized partitioning and modeling of mixed-signal circuits in (a) Project B and (b) Project C on three metrics of simulation resources: number of interconnects, number of modules, and CPU usage.

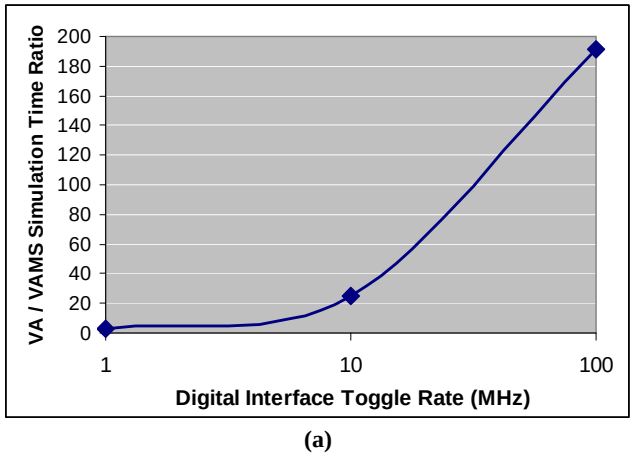
5.4 Selective Usage of Digital Constructs

In coding BMODs, particularly for analog leaf blocks, we recommended Verilog-A unless mixed-signal functionality required Verilog-AMS mainly because most CKT designers are more fluent in Verilog-A and this policy made it possible for the designer to own and maintain the schematic and BMOD views, which reduced the burden on the validation team. Secondly, Verilog-A is usually recognized by analog simulators, however Verilog-AMS is not. Therefore the validation team supported only the relatively small population of CKT designers who required training on Verilog-AMS usage and environments.

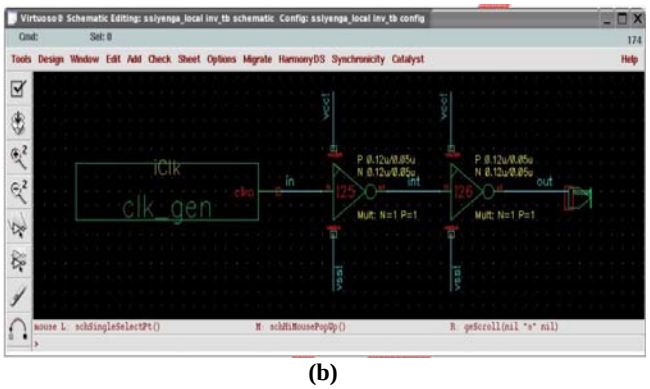
To determine which types of mixed-signal blocks require VAMS representation, we benchmarked a simple clock buffer, shown in Figure 7. For toggle rates approaching DC there is little benefit in VAMS. But as toggle rates exceeded 10 MHz, simulation time of VA increased exponentially compared to VAMS. This inflection point can guide the selection of VA or VAMS to model a mixed-signal block. Certainly anything with high-speed digital interfaces (PLL, DLL, fast data converters, etc) should be modeled in VAMS. Note that a VAMS leaf cell requires all parent cells up the hierarchy also be represented in VAMS to retain the logic discipline.

6. FUTURE WORK

There is no analog equivalent of logical assertion-based verification (ABV), as found in PSL and SVA, although considerable research is being done in the standards community on extensions to system modeling. One example is the activity by Accellera to align Verilog-AMS and SystemVerilog into a SystemVerilog-AMS standard. As an intermediate solution, we are actively investigating methods to develop macro-based analog self-checkers, initially based on Verilog-A, which may be embedded in analog circuits and/or their behavioral models.



(a)



(b)

Figure 7. Impact of properly modeled digital nodes on (a) actual simulation time using a clock buffer (b) as an example.

Simple checkers have been coded to monitor basic input-output relationships (like block enable/disable and signal voltage or current levels). More sophisticated checkers create a parallel representation of a block as a parameterized transfer function. By applying the block's input also to its checker, it is possible to dynamically compare results of the DUT and its checker; a violation is reported if their difference exceeds a preset tolerance. These checkers recognize their position in the design hierarchy such that the top-level testbench may universally enable/disable checkers above or below a certain depth. This permits fine-grained control of validation coverage and simulation speed without requiring edits of the actual blocks.

7. SUMMARY

Considering the growing complexity of SOCs, we introduced the concept of full-chip mixed-signal validation (FCMSV). A key component of this approach is the use of SystemVerilog to propagate analog behavior through scalar real ports; SV was also proposed for many high-level cluster BMODs. To facilitate FCMSV we then presented an environment, based on a common Intel RTL flow, in which an SV testbench suite comprehends multiple HDLs and simulators. The environment was reviewed in detail, including its application to combined digital and analog regression testing. It greatly reduced testbench coding, and enabled testbench and BMOD sharing between FCV and MSV.

A methodology for top-down development and bottom-up implementation was presented with particular emphasis on BMOD evolution and equivalency checking. Guidelines were discussed for the incremental improvement of analog BMODs as part of each design milestone.

Finally validation results were presented from three different projects. The capability of this flow to perform rapid analog regression testing was demonstrated. More significantly the performance benefits of various validation strategies were quantified. Firstly, intelligent hierarchy partitioning and BMOD configuration provided substantial simulation speedup without sacrificing any transistor-level accuracy. Next, it was shown that a careful validation plan can achieve very high efficiency in BMOD coding: minimal BMODs and little redundancy. Lastly, we benchmarked a circuit to guide selection of Verilog-A versus Verilog-AMS in modeling a block.

The union of FCV and MSV by a single team had great opportunities for resource sharing. A team of four full-time staff were able to validate each project by following this approach.

8. ACKNOWLEDGEMENTS

The authors acknowledge the support of DAs (Vikram Muttineni, Liza Jones, and Jason Salvaggio) for implementing changes to the original tool environments. And we acknowledge the technical contributions of Prashant Choudhari and Roque Thuo. Significant support was also provided by Cadence applications engineers, particularly Vasant Pai.

9. REFERENCES

[1] Virtuoso AMS Designer Simulator User Guide, Version 8.x, Cadence, Inc.
[2] Miller, I. and Fitzpatrick, D. Analog Behavioral Modeling with Verilog-A Language.
[3] Kundert, K. S. and Zinke, O. A Designer's Guide to Verilog-AMS.
[4] Accellera, SystemVerilog 3.1a Language Reference Manual.
[5] Miller, I., Fitzpatrick, D., and Aisola, R. "Analog design with Verilog-A." In Proceedings of Verilog HDL Conference. 1997.
[6] Mayes, M and Chin M. S. "All Verilog mixed signal simulator with analog behavioral and noise models." In VLSI Circuits. 1996.
[7] Wong, W., Gao, X., Wang, Y., and Vishwanathan, S. "Overview of mixed-signal methodology for digital full-chip design/verification." In Proceedings of International Conference on Solid-State and Integrated Circuits Technology. 2004.
[8] Joeres, S., Groh, H.-W., and Heinen, S., "Event driven analog modeling of RF frontends." In Behavioral Modeling and Simulation Workshop. 2007.
[9] Daglio, P., "A complete and fully qualified design flow for verification of mixed-signal SoC with embedded Flash memories." In Design, Automation and Test in Europe. 2006.
[10] Pennell, M., Forni, B., and Evans, R. "Full chip mixed-signal simulation of a disk drive read/write channel." In Proceedings of Bipolar/BiCMOS Circuits and Technology Meeting. 1996.