CHAPTER 3

AMS Behavioral Modeling

Ronald S. Vogelsong, Ph.D.

Overview

Analog designers have for many decades developed their design using a “Bottom-Up” design flow. First, they would gain the necessary understanding of the desired functionality of their subsystem, and then they would build the transistor-level design from the basic blocks. When that subsystem was complete, it would be tested as a unit, and given to another group for integration into the system. Only when the simulations took too long at the transistor level would they seriously consider the development of behavioral models for analog and mixed-signal subsystems. They would use those behavioral models to replace less interesting portions of the system so that simulations could be performed more rapidly for the parts of the system that were of interest.

The trouble with this approach is that it is usually reactionary: modeling is considered because the simulations are getting so large and complicated that they can’t run simulations fast enough and/or are suffering from convergence problems. The project is falling behind schedule, so a modeler is asked to speed up the simulation throughput. This approach to modeling as a last-minute Band-Aid is problematic: the amount of time needed for modeling pushes the project farther behind, model quality can be questionable, and significant simulation speedup is not assured. What’s often learned from this process is that planning is needed earlier in the process so that the modeling and simulation requirements are anticipated and available in a timely fashion.

In order to effectively integrate modeling into the system development process, the system should be analyzed ahead of time and a plan developed that integrates behavioral model development with the expected simulations that will be needed to verify the system operation. A “Top-Down” design flow can be defined. Requirements are developed for each block within each sub-system, and those requirements converted into behavioral models. These sub-systems, composed of behavioral models, can then be fully exercised with verification testbenches to see whether they
meet the requirements of the system. If not, subsystem characteristics can be adjusted until satisfactory results can be obtained. At that point, the transistor-level design for that subsystem can begin.

![Figure 1: Typical Top-Down Design Flow](image)

In a top-down flow there is more valuable content that the analog designer can use to develop the implementation. In the past, only a specification, which estimated what might be needed at a high level, and textual descriptions of what the connectivity between blocks should be, was available. Now they have a fully operational model of the blocks that need to be developed along with testbenches that are used to verify these blocks independently, as connected into the subsystem, and in the system. While they are working on the design of these well-described subsystems, each model (which can now be considered an “executable spec”) will continue to be used to fully verify that all controls operate as expected, passing through all levels of interconnect and interfaces from the system through the analog and mixed-signal subsystems and back out to the system.

This approach allows much of the critical functional verifications to be performed efficiently prior to the completion of all transistor-level implementations. Incremental verification can be performed during the block-level design. This reduces the verification bottleneck that would otherwise occur at the end of the design phase. With proper modeling of the subsystems, functional verification can start much earlier and can complete hundreds or thousands of times faster than would have been possible if pure transistor-level circuits were required to describe the analog functionality.
Modeling Classifications

System verification is easier if models are available for all of the blocks within the system. Unlike in the digital domain, analog and mixed-signal modeling has a huge range of possible features to model, and few tools exist to help automate the model development process. It can take from minutes to months to develop a model (depending on complexity), and the model can simulate at a rate from slower than the transistor-level design to a million times faster (depending on model type, features, and complexity). Efficient model development requires that decisions are made about the types of models that are needed and the characteristics that must be modeled in order to bound the model development and verification time. Without careful planning, models may not function properly when connected to other models, or may operate very inefficiently, resulting in significant delays in the verification process.

There are several dimensions that need to be understood in model development:

- the target simulator has particular language requirements depending on whether it’s analog, digital, or mixed-signal;
- the model itself could be of small, medium, or large size depending on the level in the hierarchy that the modeling is performed; and
- the level of detail in the model can vary from very basic first-order effects, to reasonably functional descriptions, to fully characterized implementations including the detailed nonlinear dependencies existing in the system.

There is no single correct modeling approach, but with many choices available there are places where poor selection of modeling dimensions can result in models that are not well-suited for the tasks intended.

The point of reference for model development is its intended purpose, or in other words: what is the model going to be used for? The answer to that question defines the features needed in the model. There is a trade-off between the time required to develop the model and the time saved using the model. If the only purpose for putting a feature into a model is for one particular simulation, then it may be more productive to run a few longer, slower simulations using the transistor-level circuitry for those few tests rather than spending time and effort developing features that will only be used for a few simulations.

Model Development

When a system is broken down into smaller subsystems, each with their own block-level descriptions, there are often several levels at which models could be developed. There could be a single model for an entire subsystem, or a model for the primary blocks within that subsystem, or models for each of the lower-level blocks inside of one of those primary blocks. The efficiency of modeling at any given level depends on a number of factors. There may be more than one level where modeling is feasible, but the corner cases where the model is “too big” or “too small” can be readily identified:

- If a block has an excessive number of pins, a highly complex or abstract operational description, or it includes complex subsystems that each need their own description, it is likely that it would
be beneficial to decompose the block and model those subsystems that occur inside of it, which typically would have more reasonable pin counts and specifications.

- If a particular level in a design has a large number of blocks each with a few pins, and specifications are defined at a higher level (not on the small blocks themselves), this is generally too low a level to define behavioral models. It is probably too close to the transistor level, there may be too many models to develop, and the block specifications for each may not be well-defined. This is a good level for transistor-level implementation, but not for behavioral modeling.

The optimal level for modeling is for blocks that have a moderate number of pins and a good specification of the functionality that is expected of that block. This simplifies both model implementation and verification.

**Design Topological Considerations**

When attempting to behaviorally model a system, the division of the system into a structure of sub-modules for the purpose of design, modeling, and verification is important. In a well-defined system, there will be a clear description of the input/output characteristics of each cell. Proper construction will break the design down into functional operations, and each function will be broken down into more basic operational blocks. In such a system, it should be clear how to choose the correct structure where modules are of the proper level of complexity and functionality.

However, common design structuring approaches can greatly complicate this process. For example, an analog block may require digital control, so a simple module, from a high-level perspective, may be broken into two pieces by a designer: an analog portion that requires a large number of digital controls to drive its lower-level implementation, and a digital block that generates all those low-level signals. As two independent blocks, modeling the behavioral description becomes very difficult – low level controls have been promoted to a higher level, and so the proper operation can only be described and verified with the two modules working together. In such cases, the proper approach is to create a single cell for the overall function, and inside of that cell place the separate digital and analog blocks. Such a block can then be specified, modeled, and verified at the level of the complete function, while the designer can still separate the analog and digital portions as lower-level implementation details of that subsystem.

Another common problem occurs when a system is described by combining complex blocks with raw transistor-level circuitry that is used to connect those blocks together. Any schematic that includes directly placed transistors, resistors, capacitors, etc., cannot be used in a behavioral model of the system because these cannot be converted to a different format. The only place that transistors and other circuit elements should show up in a well-defined block diagram is inside of the transistor-level implementations. Switches or resistor dividers should always be implemented as part of the functionality of the blocks that are being developed, not added between the blocks as an afterthought, since that will complicate the development of a behavioral model-based verification plan for the system.
Types of Modeling

During the verification of a large integrated circuit, it is common to use a variety of modeling formats in the verification process. Common formats can include:

- **Device based design** (*Spectre, SPICE*) – schematics built using process-specific devices is the standard transistor-level design technique. A macromodeling approach can also be used that uses generic elements and dependent sources to define simple block operations.

- **Analog modeling** (*Verilog-A*) – defines an analog description of relationships as current/voltage equations to be solved by the analog solver.

- **Mixed-signal modeling** (*Verilog-AMS [7], VHDL-AMS [8]*) – allows linked descriptions of analog operation of electrical portions and digital operation of logical portions of the block’s operation.

- **Discrete real number modeling** (*Verilog-AMS, VHDL [6], SystemVerilog [11]*) – uses a discrete solver for the model definition. This replaces electrical operations with the computation of real output values in terms of real input values to define primary input-to-output functionality. It typically ignores impedance effects.

- **Logic modeling** (*Verilog [5], VHDL, SystemVerilog*) – model defines discrete logic data flow, ignores analog operations.

![Figure 2: Modeling Accuracy versus Performance Gain compared to Transistor-level Simulation for Various Modeling Styles](image)

It is important to track the different models, or views, available for a block throughout the design and verification cycle. Blocks will typically have at least one device-based view, as well as at least one behavioral model (pure analog or pure digital if needed for specific applications, as well as
Mixed-Signal Methodology Guide

a mixed-signal and/or discrete real model for functional verification). In order to fully verify system operation, it should be kept in mind that all views of a model should be incorporated into block-level testbenches to verify that their operation stay in sync with each other, so developing too many different models can become a significant verification problem. To choose which types of models need to be developed, it is important to understand the purpose, capabilities and limitations of each style of modeling.

**Discrete Digital Modeling**

Pure digital solvers can be used to model the digital input/output characteristics of a system. Languages available include Verilog, VHDL, and SystemVerilog. This approach is extremely efficient at handling logic and timing relationships using a discrete event simulation kernel, but does not handle analog signals. It is commonly used for pure digital modeling (including synthesis), as well as with black-boxed analog subsystems where only the digital operations are modeled. This may be used for verification before any analog or AMS constructs are available. The discrete event simulation approach can be extended to model analog signals as discrete real values (see 'Real Number Modeling' below).

**Continuous Analog Modeling**

Pure analog (SPICE-like) languages can be used to model the electrical nature of a system. The standard analog behavioral modeling language is Verilog-A. The language is used to describe the interrelationships between voltages and currents in the system, in much the same way as resistors, capacitors, and transistors provide the description in a traditional circuit. Impedance characteristics along with integral and derivative dependencies can be written directly. The Verilog-A model is converted into a set of simultaneous equations (nonlinear, ordinary differential equations) suitable for a simulator. The built-in models for transistors and other components are also defined internally to create sets of simultaneous equations in a similar format. During transient analysis, matrix-based numerical analysis techniques are used to solve the complete set of voltage and current equations at each analog timestep.

The use of well-defined analog behavioral models can result in a reasonable speedup, typically in the range of 10x to 50x, compared to transistor-level models. Simulation speed is based on the size and complexity of the equations to be solved (which determines the size of the equation matrix), in addition to the timestep size used to step through the transient simulation. The speedup from using behavioral models is primarily based on the reduction in the number of nodes and equations in the system and the ability to take larger timesteps due to fewer lower-level nonlinearities in the system.

If logic signals are included in the analog model, they must be converted to electrical signals that swing between defined voltage levels with specified rise & fall times. Analog simulations of very active logic networks often simulate relatively slowly due to the small timesteps required during each logic transition. A simulation performed using separate analog and digital simulators using Inter-Process Communication (IPC) between the simulators can suffer from the same speed problem, since all logic signals must be converted to analog waveforms before usage on the analog side of the co-simulation environment, resulting in similar small timestep issues.
Mixed-Signal Modeling

Mixed-Signal simulation combines the discrete digital and continuous time analog solvers within a single simulator. Modeling languages Verilog-AMS [7] or VHDL-AMS [8] can be used to describe these models. Mixed-signal languages allow the most natural modeling of mixed-signal systems where the digital portions can be modeled using standard discrete modeling techniques, while the analog portions can be modeled with the standard electrical modeling approach. Data and events are transparently passed between the two simulation algorithms. This is also the language of choice for defining mixed-signal testbenches. Verilog-AMS code can be used to write procedures that both read and write digital and analog quantities, making this an optimal environment for mixed-signal verification testbench development. Real Number Modeling techniques (described below) can also be used in these languages, so they can handle all of the flavors of models being described here.

Simulation speedup when using well-defined AMS models (compared to all-transistor level) depends on the amount of transistor-level circuitry being replaced. Employing AMS techniques removes the digital circuitry from the analog simulation engine, and the replacement of the remaining analog circuits with AMS operations can speed up the remaining analog portion of the simulation by a factor of 10x to 50x.

Real Number Modeling

Real Number Modeling (RNM) is a special technique used to model electrical signals by representing them as real values. Provided that the modules are at a sufficiently high level of abstraction, the interfaces can be described by passing single real numbers between blocks to represent the voltage (or current) signal being transferred. This is a powerful way to rapidly simulate complex systems. RNM is available in the Verilog-AMS, SystemVerilog [11], and VHDL [6] languages. This technique uses a discrete solver, with no analog solver, and can be used to simulate systems at incredible speeds (many thousands to millions of times faster than transistor-level speeds for large simulations). It is primarily limited to modeling at a high enough level of abstraction that bi-directional analog interactions between blocks are not significant. In other words, typical real modeling defines blocks in terms of input/output transfer characteristics with no strong direct feedback present between the blocks (traditionally an analog solver would be required to iterate to a solution involving feedback). Logic can be modeled naturally in these languages, so this is also a very good choice for systems with only a small amount of analog content.

Combined Approaches

It is common when working in an AMS environment to develop models that use a combination of techniques. For example, an RF receiver could be modeled using RNM techniques for the RF signal path; electrical for the baseband signals, biasing and power supplies; and discrete logic for control signals. This has the benefit of reasonable simulation times due to the high-speed signal processing and digital control performed in the discrete environment, along with easy interface to transistor-level subsystems from the analog baseband and bias connections.
Putting together complex higher-level testbenches will often include combining portions that include several different modeling styles. For example logic subsystems may be designed in RTL (synthesizable Verilog code), while a few analog subsystems may be investigated using transistor-level designs. The remaining analog portions could use a combination of analog, mixed-signal, and real number behavioral models.

Basic Modeling Formats

Significant time and effort is required to learn the syntax and proper coding techniques for each of the four types of modeling approaches, and that is not the purpose of this book. Each approach would require a book of its own. To gain some insight into the types of capabilities and limitations of each approach, it is useful to look at a few examples of each. This section provides an introduction to each of the languages by working through a model of a fairly basic mixed-signal device – a programmable gain amplifier. It will be described in each of the four language formats: mixed-signal, analog, digital, and real number modeling. The Verilog-AMS language and its subsets will be used for the models defined here.

Model Operational Description

The primary operation of a Programmable-Gain Amplifier (PGA) is to provide a gain from a differential analog input to a differential output, where the value of the gain is controlled by a digital bus input. A few other pins are required on a real PGA as well: power supplies, a bias input, and a digital control input to allow the output to be enabled or disabled.

The operation of the block is defined by the dependence of the outputs on all of the inputs, not just the gain path. To build a good model, it should only provide the proper response when properly connected. For example, it should not provide gain if the device is disabled or if the power supply is not connected. Here is the specification for the model to be developed:

- Analog input \((\text{INP, INN})\) is a pure differential signal
- Gain in dB is controlled by a 3-bit bus \((\text{GAIN}[2:0])\), which selects the gain between \(\text{dBmin}\) for GAIN=000 and \(\text{dBmax}\) for GAIN=111
- Differential output signal is centered halfway between the supplies \((\text{VDD, VSS})\), and saturates to the supply levels if overdriven
- Each output pin has a nominal output resistance \(\text{Rout}\) when active
- When enable input \((\text{EN})\) is not high, output is changed to high impedance
- If bias input \((\text{VB})\) or supplies are not at proper levels, output will be set to zero

Note that this list defines the qualities that the model must have, rather than specifically defining numeric values of the effects. It is good modeling practice to define parameters or variables for the controlling characteristics, and then specify the initial estimated values for those characteristics as the default values of the parameters. Using that format, rather than hard-coding constants into