Fast Validation of Mixed-Signal SoCs

Daniel Stanley, *Graduate Student Member, IEEE,* Can Wang, *Graduate Student Member, IEEE,* Sung-jin Kim, *Member, IEEE,* Steven Herbst, Jaeha Kim, *Senior Member, IEEE,* and Mark Horowitz, *Fellow, IEEE*

Today's mixed-signal SoCs are challenging to validate. Running enough test vectors often requires the use of event-driven simulation and hardware emulation, which in turn necessitates the creation of analog behavioral models. This paper reviews different approaches proposed to address that modeling challenge, and shows how they can be divided by the methods used to solve for analog circuit values, represent analog waveforms, and validate analog functional models. We illustrate the power of these techniques as applied to a 16 Gb/s PHY, demonstrating a 10,000× speedup vs. SPICE simulation using event-driven models in Verilog simulation, and a further 5,000× speedup using synthesizable analog models in FPGA emulation.

***Index Terms*—analog validation, behavioral modeling, event-driven simulation, equivalence checking, linear abstraction, mixed-signal circuits, validation**

I. Introduction

FOR many reasons, including the increasing performance requirements and low cost of digital hardware, an increasing number of analog building blocks have rich connections with digital logic that "controls" their operation. These blocks depend on sophisticated calibration/adaptation algorithms for proper operation. This coupling of the analog and digital sections of an SoC makes it necessary to validate their combined operation, which is challenging because analog and digital circuits fail for different reasons, and thus traditionally use different tools and testing approaches for validation.

For digital systems, validation is all about test coverage. This need to explore the entire state space arises because the output result surface of a digital circuit isn't smooth; an incremental change in an input could completely change the output, so one must look at **all** possible inputs to ensure correct behavior. Thus, advanced testing strategies employ formal methods to guide the exploration [1], try to find specific inputs that violate assertions [2], use randomized testing, and generally require complex validation frameworks.

Interestingly, analog circuits are nearly the opposite. Their analog nature implies that their output surface is smooth, since that is what makes them analog.[1] In fact, for most analog circuits, that surface is not only smooth but also nearly linear with respect to primary analog inputs.[2] The smooth result surface means that the measured output of one set of inputs provides a lot of data about what the output will be for a different set of inputs. This means that generating a set of test vectors to cover a traditional analog circuit is not difficult. Instead, the complexity in analog testing comes from the non-discrete nature of the outputs. One needs to consider many more circuit inputs (supply voltage/noise, temperature, process, bias signals, etc.) and extract their effect on the circuit. As a result, analog circuit validation has focused more on creating the right high-fidelity model of the circuit, and characterizing different "noise" sources.

Validating modern mixed-signal building blocks, which use tightly-coupled analog and digital logic, is challenging because analog behavior must be modeled in a way that is compatible with the digital validation tools used to run test vectors and exercise calibration/adaptation loops. This paper reviews different approaches to address that challenge, using signal flow or functional models of analog blocks. To set up this discussion, the next section explores the validation concerns for analog designs and shows how they can be partitioned into issues handled by traditional analog simulation/validation flows and issues addressed using fast analog functional models.

These functional models are often called real number models [4] but differ widely in how they model the underlying circuits. Section III-A presents a framework that characterizes the different approaches and uses this structure to discuss their advantages and limitations. Section III-B focuses specifically on methods to automatically generate functional models, and Section III-C describes how to validate that a functional model matches the circuit it represents. As the complexity of SoCs increases, final validation often needs to run on an emulator, which requires synthesizable analog models. While that is a hurdle by itself, it turns out that the modeling strategies used in emulation also have to be chosen carefully to avoid performance impacts. These issues, and methods for addressing them, are described in Section IV. Finally, we apply

This work is supported by Stanford SystemX, DARPA POSH and DSSoC programs, a Hertz Fellowship, and a Stanford Graduate Fellowship.

D. Stanley, C. Wang, S. Kim, S. Herbst, and M. Horowitz are with Stanford University, Stanford, CA 94305, USA (emails: dstanley@stanford.edu, canswang@stanford.edu, sunginau@gmail.com sherbst@alumni.stanford.edu, horowitz@ee.stanford.edu).

J. Kim is with Seoul National University, Seoul 08826, Korea and Scientific Analog, Inc., Palo Alto CA 94301, USA (e-mail:jaeha@snu.ac.kr).

[1]As Kim showed in [3] this smooth response curve may not be apparent when looking at the voltage (or current) vs. time waveforms. For these analog circuits, one first needs to transform the domain of the input and/or output of the system. For example, a VCO with a digital output doesn't seem linear, but if one looks at the phase of the generated output, it is the integral of a (piecewise) linear function of the input voltage.

[2]As will be described later, digital inputs can cause large changes in the response and must be enumerated.

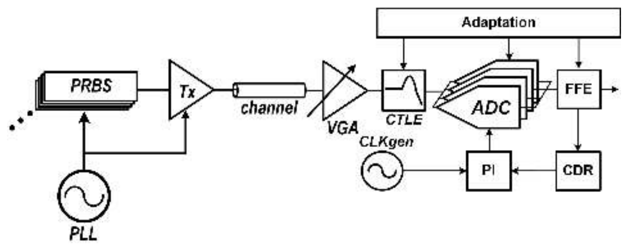

Fig. 1. High-speed link architecture used as an example throughout this paper.

these methods to a high-speed link and discuss the results in Section V.

II. Analog Design

Like their digital counterparts, analog circuits are designed and validated hierarchically. For digital designs, low-level circuit checking is handled in the construction/validation of standard cells and ERC checks that are performed as part of the design flow; the designer only deals with functional models extracted from that flow. For custom analog cells, however, this circuit-level checking must be done for each design.

A. Circuit Simulation

To estimate the behavior of circuits, one typically uses a circuit simulation program that is a variant of *SPICE* [5], such as *Spectre* [6] or *HSPICE* [7]. These simulators contain sophisticated device models and solve for nodal voltages using numerical integration with guaranteed error control to provide accurate results. To validate the function of a circuit, the designer creates a set of testbenches to measure the key properties of that circuit, for comparison with its design specification. As we will see later, these testbenches can also be reused for functional model extraction and validation.

As one starts to aggregate more analog (and possibly digital) blocks together, the speed of SPICE simulation can become an issue. There are two approaches to reduce simulation time: simplify the circuit model and simplify the computation used. As we will see in the next section, simpler circuit models, called macromodels, can enable much simpler computational methods. Attempts at both types of simplification started soon after SPICE was created [8], [9]. Over the past few decades, a number of *FastSPICE* products [10], [11], [12], [13] and *parallel SPICE* products [14], [15], [16], [17] have been introduced. These products provide significantly improved throughput and capacity with a small decrease in the overall accuracy, allowing large circuits to be simulated.

While modern simulators can easily handle the complexity of individual analog blocks or a large collection of blocks, they still run into issues for systems with slow digital correction loops. In these systems, such as the high-speed link described next, analog blocks are validated individually with SPICE simulation and then replaced with simplified models for higher-level system validation.

B. High-Speed Link Example

Fig. 1 shows the block diagram of a high-speed link that we will use as an example throughout this paper. It consists of a pseudo-random bit sequence (PRBS) generator that provides data to a transmitter, a lossy physical channel, a variable gain amplifier (VGA), a continuous-time linear equalizer (CTLE), and a high-speed time-interleaved analog-to-digital converter (ADC). The ADC is driven from 4 phases of a 4 GHz clock, yielding a 16 GS/s conversion rate, and each phase of the clock is generated through a phase interpolator (PI), which allows each of the clocks to be placed anywhere in the clock cycle.

This link uses a relatively small number of analog blocks: the phase-locked loop (PLL), transmitter, channel, VGA, CTLE, ADC, sampling network for the ADC, and PI. Each of these circuits is simple enough that it can easily be simulated at SPICE-level with modern tools. For each block, a set of testbenches is created to extract the critical parameters of that block. For example, for the CTLE, one would characterize the differential and common-mode gain of the input (vs. frequency), how that gain depends on other inputs (e.g., gain adjustment inputs or process/voltage/temperature variation), and the “gain” of other inputs (e.g., supply voltages or bias lines) to the output. In some systems, the CTLE can be nonlinear for large inputs, so one might also have testbenches to extract parameters describing that behavior. Running the testbenches for a given block allows designers to determine whether the circuit matches its specification.

This link design uses extensive calibration and adaptation loops that correct for gain/offset mismatch between the ADCs, equalize the channel, set the parameters of the VGA and CTLE, and make the sampling clocks track the incoming data. These feedback loops take 100,000s of cycles to settle, which is extremely slow to simulate, even when using *FastSPICE* simulators. To evaluate the system at this level requires higher-level functional models of the analog blocks. Methods for creating those models are described next.

III. Functional Models

To speed up the validation of mixed-signal SoCs, one needs to move from solving for voltages and currents that satisfy device constraints (circuit simulation) to functional models where input changes trigger output changes. This means all functional models must address two challenges:

1) In a circuit, inputs and outputs have continuous, real-valued waveforms, but in functional simulations, they are values that change only at discrete times.
2) Devices/circuits are fundamentally bidirectional, but functional model execution is unidirectional from inputs to outputs. Circuit simulation finds the voltages that make the circuit consistent, but in a functional model, outputs must be a function of the inputs.

These challenges provide a framework for categorizing different approaches used to create functional models.

A. Functional Model Classification

Since functional models only receive inputs and generate outputs at discrete times, one must represent the shape of analog waveforms in between those updates. To that end, a variety of different *feature vectors* have been used to represent analog signals in functional models. The simplest feature vector is a single value, and the simplest interpolation function is to hold that value constant until the next value arrives (i.e., representing a piecewise-constant waveform).

This simple feature vector is the most widely used, usually in conjunction with a discrete-time model, as described later in this section. Piecewise linear models [18] have been used to better approximate waveforms, especially when using non-uniform time steps between events. A piecewise linear waveform representation is a two-element feature vector with a value and slope passed at each event. As we describe in Section IV-B, for emulation we have found that using a feature vector consisting of spline points can be effective [19]. Another approach is to express waveforms using sums of complex exponential functions, which is a general form for a linear system response [20]. In that case, the feature vector is variable-width, containing a set of eigenvalues (i.e., pole frequencies and multiplicities) and their eigenvectors (i.e., magnitudes). This representation is adopted by Scientific Analog's *XMODEL* [21].

Given a representation of analog waveforms, we next need a way to compute model outputs without requiring circuit simulation. Almost all functional models use one of two approaches: discrete-time modeling or piecewise linear modeling. In the discrete-time approach, the differential equations used for circuit simulation are converted into a set of finite difference equations, resulting in a discrete-time model. By Nyquist's Theorem, this conversion will yield an equivalent model if the chosen time step is small enough to sample signals faster than their Nyquist rates. Since a high sampling rate is needed in these systems, they are often referred to as *oversampled* models. The resulting set of difference equations can be easily represented as a discrete-time FIR/IIR filter. While this approach is widely used, it can be inefficient when modeling circuits with high bandwidth, which need fine time resolution. In the high-speed link example, this means we would need to have a number of samples (6-10) during each bit period to achieve good fidelity, and even more samples to model the effects of jitter. Thus, while these models are popular, they cannot be used for all situations. In particular, they can dramatically limit the speedup that is possible through emulation [19].

The main alternative to oversampling is to make functional models piecewise linear. Using this approach, until a model is re-evaluated, its outputs (feature vectors) represent the response of a linear dynamical system to its inputs (also described by feature vectors), which has a closed-form solution. How these outputs are computed and how often the model needs to be re-evaluated depend on the model and the feature vector used. For example, with the complex-exponential representation of *XMODEL*, the output response can be computed in the Laplace domain and expressed exactly using the same complex exponential form [20]. Here the model would only need to re-evaluate if its inputs changed, or if the model transitioned to a different linear operating mode. For a model with a simpler, piecewise linear waveform representation, more evaluations would be required, since the response of a linear system to piecewise linear input is not generally piecewise linear. To solve that problem, the model may project a piecewise linear segment that approximates the true response, scheduling a re-evaluation event at the end of the segment [22].

The piecewise linear modeling technique can be used on a wide variety of circuits that may not seem linear at first [3]. As an example, consider the phase interpolator in the high-speed link example, where the input and output voltages are digital clock signals. If we think of those signals as being in the phase domain, rather than the voltage domain, the linear relationship between them becomes clear: the output is a weighted average of the inputs. This same technique can be used to model a more complex circuit like a PLL, which is usually broken down into a phase detector (PD), filter, voltage-controlled oscillator (VCO), and possible clock dividers. The PD converts the input phase difference into a voltage or current (which maybe pulsed). This signal then passes through a linear filter and drives the VCO. In the phase domain a VCO is simply a phase integrator, where the time constant of the integrator is set by the control voltage. Notice that all of these operations can be modeled by "linear" systems. The output clock is generated by switching the output value each time the phase of the VCO increases by π, or by a more sophisticated scheme if the following circuit depends on intermediate VCO output voltages. The same technique can be used to create simple, linear models for duty cycle adjusters and voltage-to-time converters, among many others.

In addition to the waveform representation and choice of discrete-time vs. piecewise linear system model, functional models can also be classified by the granularity of the circuits being modeled. In general, higher-level models are more efficient because they avoid the computation of internal signals that are not observable from a system perspective, but these models require more specialized effort to create and validate. As we will see in the next section, lower-level models can be easier to generate automatically.

B. Automatic Model Generation

To auto-generate a functional model, a tool needs to have three components: templates for functional models, a method to tear a large circuit into smaller pieces that fit into those templates,[3] and a way of extracting template parameters from each torn circuit piece such that its model will match its behavior. To understand how this process works, we start by describing the flow used by tools that generate functional models from netlists.

Since functional models are unidirectional, we tear the initial netlist into element clusters that have strong couplings within them and must be solved simultaneously [21], [23]. Fig. 2 illustrates the circuit clusters identified from an example circuit. Each of the clusters can then be replaced with an appropriate functional model. If no other information is given about the circuit's functionality, the tool replaces each of the

[3]This component might be optional if it can be guaranteed that the template can model the behavior of the entire circuit.

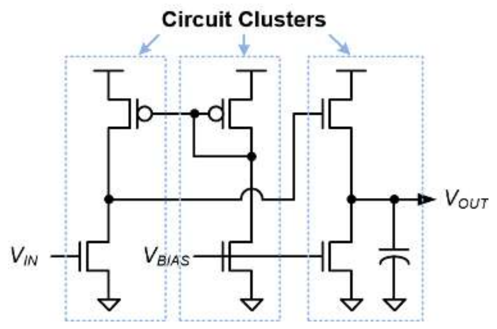

Fig. 2. Identification of circuit clusters to convert a transistor-level circuit into a signal-flow system.

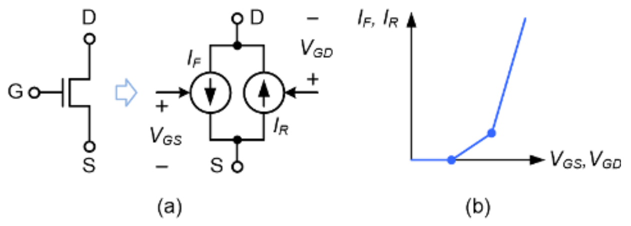

Fig. 3. Piecewise linear modeling of a nonlinear transistor element.

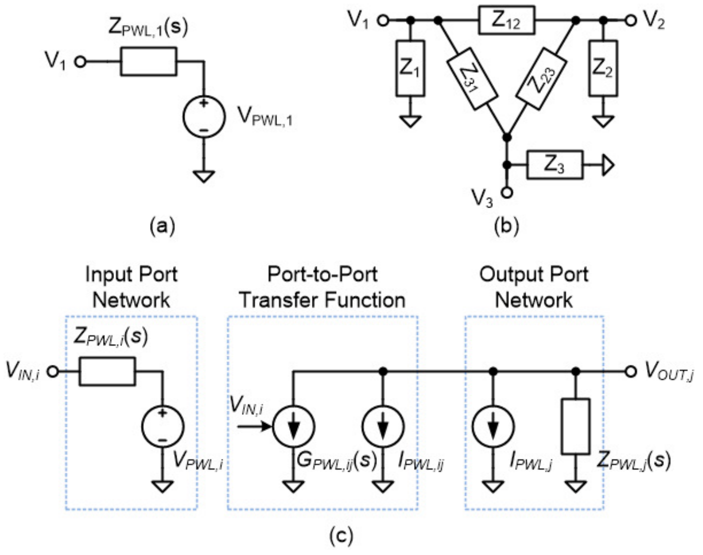

Fig. 4. Examples of analog macromodel templates: (a) a single-port circuit, (b) a multi-port network with passive impedances, and (c) an active circuit with multiple inputs and outputs.

transistors with a macromodel, builds the circuit equations for each cluster via modified nodal analysis [24], and derives the Laplace-domain transfer functions from its inputs to outputs [21], [25]. This transfer function is a parameter of the functional model template and is used to create a functional model for this subcircuit.

Different tools use different transistor macromodels and model templates. For example, *XMODEL* models nonlinear elements such as transistors and diodes as piecewise linear, meaning that a circuit cluster may take different input-to-output transfer functions depending on which operating region it is in. Fig. 3 illustrates the basic piecewise linear model of a transistor used by *XMODEL*. The transistor is modeled with a pair of forward and backward nonlinear voltage-controlled current source (VCCS) elements, of which V_{GS}-to-I_F and V_{GD}-to-I_R characteristics are modeled using a piecewise linear function. Using this model and pre-characterized parameters, *XMODEL* is able to compute the transfer functions between the inputs and outputs of a given cluster during runtime.

While these functional models extracted at the device level provide performance gains, larger gains are possible with higher-level analog macromodels. We found that a small number of macromodel templates can cover a wide variety of analog circuits commonly used in practice. Fig. 4 illustrates a few of them; each can be regarded as a special case of a multi-port network model.

The first template, in Fig. 4(a), is for circuits with a single port, such as those that generate a voltage or current output or serve as a load to others. It is basically a Thevenin-equivalent model of the circuit (a Norton-equivalent counterpart is also possible). The equivalent open-circuit voltage V_{PWL} and series impedance $Z_{PWL}(s)$ can take different values depending on the present value of the port voltage V_1 to model nonlinearities in a piecewise linear fashion (e.g. a current source falling into a linear region when the output voltage becomes too low). The series impedance $Z_{PWL}(s)$ described in the s-domain can express any small-signal AC characteristics. The values of V_{PWL} and $Z_{PWL}(s)$ may also change depending on the values of digital mode bits (e.g., digital inputs for power-down, reset, or parameter trimming). For example, this template can be extended to model a digital-to-analog converter (DAC), varying its output voltage, current, resistance, or capacitance as a function of its digital inputs.

The second template, in Fig. 4(b), is for circuits with multiple ports that can be described using a set of passive impedances between the ports, such as an RC interconnect, a power grid, the LC tank of an oscillator, etc. As with the first template, each port-to-port impedance can change as a function of digital mode bits, in which case the template can also model a network of switches, such as analog multiplexers/demultiplexers.

The third template, in Fig. 4(c), models active circuits with inputs and outputs such as amplifiers, filters, current mirrors, etc. The circuit may have an arbitrary number of input and output ports; the first template is used to model the DC nonlinear and AC linear characteristics of each input/output port. For each pair of input port $V_{IN,i}$ and output port $V_{OUT,j}$, a VCCS element with a transfer function $G_{PWL,ij}(s)$ and a DC current source with a value $I_{PWL,ij}$ model the transfer function from $V_{IN,i}$ to $V_{OUT,j}$ as a piecewise linear system. Again, all model parameters can change as a function of digital mode bits.

These macromodels are used to simplify the circuits which the nodal analysis needs to analyze. Raising the abstraction level even higher, to functional blocks, yields even faster models. The problem is that the best modeling approach often depends on the circuit function. In the prior section we described how to use the transfer function of a piecewise linear system to generate an output feature vector from input feature vectors. While the complete link can be modeled this way (with a functional model for each component) there are better modeling approaches.

For example, in a link, the transmitter, channel, VGA, and CTLE are often modeled as a linear time invariant (LTI) system. LTI systems can be represented completely by a single step response. Since the input into this system is binary (but with jitter the transitions occur at different times) and the output is sampled, the output sample can be quickly computed by simply adding several time-shifted step responses, where each is shifted according to the time difference between the transmitter transitions and the receiver sample point. Outputs are calculated on-demand when required by the digital clock at the output. This model is only activated when either the transmit or receive clock fires and is extremely efficient. This method requires no analog-only timesteps, and the accuracy of the output voltage is completely independent of the digital timestep.

To allow users to leverage these efficient functional models, we need a system that encapsulates these modeling approaches into flexible templates that can handle a wide variety of circuits with the same function. For example, while an amplifier is a very basic building block, its actual implementations vary widely because of the different possible sets of inputs adjusting the offset, common-mode level, gain, bandwidth, etc. The templates illustrated in Fig. 4 address this need to some degree by supporting a variable number of input/output ports and changing the parameter values with the digital mode bits, but they may not be sufficient to model how a circuit's characteristics, such as gain, bandwidth, and nonlinearity, vary as continuous functions of analog control inputs.

To deal with these issues, Lim in [26] proposed creating a library of model templates, each representing a basic analog function. For each of those functions, he realized that the only effect of additional pins would be to alter its behavior. Thus, he created templates that can extract the dependence of analog functions on user-added pins. In this approach, the user breaks a circuit into blocks (which usually are subcircuits in the schematic), picks which template to use for each block, and provides the tool the acceptable input ranges. From this information and the circuit implementation, the tool generates a functional model.

Returning to the amplifier example, a template was created to model its characteristics including gain, offset, nonlinearity, dynamics, noise, etc. If there are additional input pins added to the amplifier, they may affect one or more of these characteristics. The template contains a set of testbenches that can measure these amplifier characteristics and their dependence on the user-added pins. Fig. 5 shows how this amplifier template applies to a differential amplifier with a calibration mode, separate calibration inputs, digital differential offset correction, and gain-controlling current bias. The next section describes these templates' testbenches in more detail.

Since functional models are valid only within the conditions from which their parameters were extracted, it is important for the models to include assertions that check whether these conditions are being met [25]. These assertions verify whether the input ranges assumed during the parameter characterization are respected and can prevent the models from being used in situations in which they are not validated.

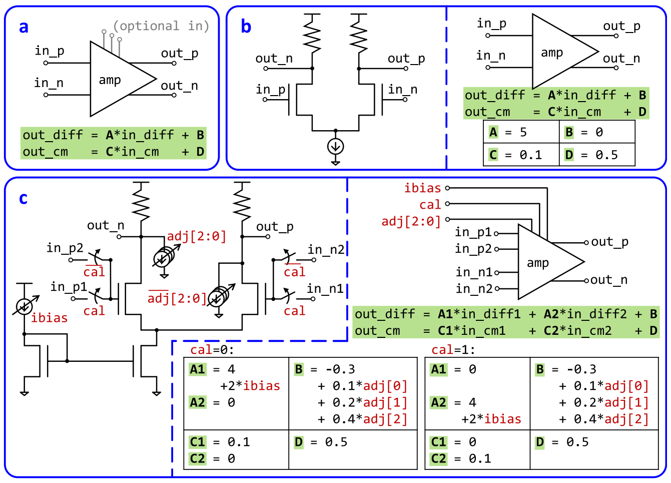

Fig. 5. (a) Top-down functional model of an ideal differential amplifier, with 4 adjustable parameters: A, B, C, D. (b) Simple differential amplifier circuit, and the corresponding model with parameter values that might be extracted. (c) Differential amplifier circuit with adjustable current bias, 3-bit offset cancellation, and calibration mode. To the right is the model and set of parameters that might be extracted; note their dependence on optional pins.

C. Validation of Functional Models

The goal of validation is to check that the functions of a model correctly match the operation of the circuit it represents. There are two approaches for this check. One is to use self-checking testbenches, which ensure that both a circuit and its model match the circuit's specifications [27]. The other is to use testbenches that extract model parameters and ensure that the parameters from the circuit and model match. In the latter approach, the parameters extracted from the circuit can be passed to a template to generate a functional model. Thus, testbenches associated with auto-generation of model templates are able to extract everything needed to describe the model's behavior, and therefore are able to do a complete circuit/model comparison. In this section, we will discuss what a testbench is, and how to write an effective set of testbenches for any model.

A testbench consists of three parts: stimulus generation, model/circuit simulation, and response analysis. Stimulus generation creates the test vectors that drive the circuit, which may be as simple as an explicit input waveform, but is generally a piece of code that generates a set of input waveforms and denotes which measurements should be made. Simulation is the job of a SPICE, SystemVerilog, or VerilogAMS simulator, taking explicit input vectors and returning the corresponding output vectors. Finally, response analysis converts the raw measured results to model parameters, and may be as simple as plotting the result vectors, but generally is a piece of code that extracts high-level features from the simulation results.

As we argued in [28], stimulus generation for analog circuits is easy because, in the correct domain, an analog circuit's outputs are smooth functions of its inputs. Thus, to generate a set of test vectors spanning the circuit's inputs, the designer or testbench generator can simply choose points distributed throughout each dimension of the inputs. Using Latin hypercube sampling and orthogonal sampling ensures that the input vectors are evenly distributed throughout each dimension and represent the entire multi-dimensional space. For a differential amplifier, the dimensions might be differential and common-mode voltages, while for a phase interpolator, the inputs might be the input frequency, phase difference, and interpolating ratio. When testing the model, it is also important to include some test vectors outside the acceptable input range to ensure that the model reports an issue. Once the input vectors have been chosen, they must be converted from the original domain to the voltage/current vs. time domain to be passed to the simulator.

Some additional complexity is added to the stimulus generation by control inputs. If the controls are analog, such as a bias voltage, they can simply be added to the list of input dimensions for the circuit. For some digital controls such as *sleep* or *calibrate* pins, the entire circuit behavior must be re-extracted from scratch for each of the 2^N combinations of N such input pins. But for other digital inputs, such as the bits in a DAC's input code, the effect of one pin adds to others in a linear fashion. For N such inputs, it is only necessary to check on the order of N+1 combinations, not 2^N. Examples of these two types of digital inputs can be seen in the *cal* and *adj* inputs, respectively, in Fig. 5. For validation, the designer specifies the type of each digital pin, but the testbench creates enough vectors to validate that their specification is correct.

Typically, the circuit and the functional model run on different simulators, which means that the testbench needs to create inputs for and read outputs from both. One solution is to find an analog/mixed signal (AMS) simulator that can run both models. An example is to run a transistor-level circuit and its SystemVerilog analog model on a Verilog-AMS simulator. Otherwise, to prevent writing the testbench twice, Kundert suggested writing the testbench in a high-level language and compiling it to run on various supported simulators [29]. We have been using the open-source test compiler *fault* [30] in our work. In *fault*, a user can write input stimuli in Python, which are translated to one of the supported simulator decks.

One more advantage of using a compiler from a high-level language, rather than a universal simulator, is the ability to write all parts of the testbench in a single language. For example, *fixture* [31] is a model generation tool based on the work of Lim [32] and uses *fault* to translate the input stimuli to the target simulation languages. This allows the stimulus generation and the analysis code to be written in the same Python file and allows for easy reuse of helper code, such as domain translators, between testbenches.

Once a simulation has been run on both the circuit and model, the template extracts parameters that describe the circuit's behaviors, as was done for modeling in Section III-B. With this method, two separate checks must be made. First, the extractor must ensure that the model matches the specified behavior to within some error tolerance. For example, a tool to extract the gain of an amplifier may work by finding a best-fit line for the input vs. output data. This tool must be sure to check the RMS error of the fit to catch errors such as a circuit with unintended gain compression. Second, for comparison checks, the parameter extracted from the circuit and extracted from the model are compared to each other. For self-checking templates, both results are compared to a specified value/range. In both the extraction and comparison, the designer must specify a reasonable error margin (e.g., as a percentage error), that is considered valid for that particular feature of the model. The comparison is only slightly more complex when control inputs are present. In this case, rather than extracting a single gain parameter, a gain equation that is a function of the control parameter (as shown in Fig. 5(c)) is extracted for both circuits and compared.

Sometimes the designer does not have a parameterizable representation of all the important circuit behaviors, and therefore cannot extract parameters for comparison. In this case, another option is to bound the output waveform within acceptable regions. These regions can be handwritten if the user is writing a specification for both the circuit and model to follow. For comparison checks, the user can check that the model's waveform matches the circuit's waveform within some specified bounds (ΔV and ΔT). For example, Cadence *amsDmv* implements this function [33].

Of these approaches, we strongly recommend comparing extracted model parameters for circuit vs. model validation. Two circuits meeting the same specifications don't necessarily have the same behavior, and ensuring complete coverage is hard with waveform comparisons.

IV. Emulation

Computer simulation is the workhorse for block-level functional verification but is often too slow for top-level verification tests and pre-silicon firmware/software development. Hardware emulation is an effective strategy to address that limitation. For example, IBM found that FPGA emulation provided a five-order-of-magnitude speedup in verifying Linux boot-up on a custom processor, thus reducing a multi-year simulation to a few minutes [34].

Emulation can be performed directly on FPGAs, or on commercial emulation platforms such as Cadence Palladium [35], Synopsys ZeBu [36], and Mentor Veloce [37]. In all cases, the design under test (DUT) must be synthesizable in order to be mapped to the emulator resources. This is problematic for many SoCs because, unlike digital blocks described by register transfer logic (RTL), AMS blocks are not synthesizable.

Today, the emulation of AMS blocks is often accomplished using basic logical models that are sufficient only for verifying high-level protocols and top-level connectivity. Since AMS blocks often interact with digital blocks and firmware through complex feedback loops, this rudimentary modeling approach severely limits both verification coverage and firmware feature validation. Several approaches have been proposed to address that limitation.

The most comprehensive AMS emulation model is a test chip containing all AMS blocks, which interacts with the SoC digital functions emulated on an FPGA. When such a test chip is available, it can be used to build a very fast and accurate emulator. R. Sanchez [38], for example, emulated an ADC-based high-speed link design using a TI-ADC test chip, where the digital parts of the design were implemented on an FPGA. The result was a 1 Gb/s high-speed link emulator – five orders of magnitude faster than the fastest reported RTL simulation of high-speed link (B. Lim [22]).

Unfortunately, that approach is often rendered impractical by two main factors: (i) rapid development cycles dictate that analog circuits are developed in parallel with the digital circuits that interface with them, and (ii) bandwidth limitations between the AMS test-chip and the emulator FPGA limit emulation speed. Hence, the rest of this section focuses on fully virtual AMS emulation, where the entire chip design is modeled within an emulator. That requires analog blocks to be replaced by synthesizable models, as we describe next.

A. Oversampling

Oversampling, described in Section III, is the modeling approach most commonly used for emulation. In the context of emulation, oversampling is implemented by having each "tick" of an emulator clock correspond to a fixed-size timestep. Analog circuits are then implemented as fixed-coefficient discrete-time filters, which map efficiently to emulator resources.

An example of oversampled emulation is work by R. Bhattacharya [39], who created a synthesizable model for a DC-DC buck converter on an FPGA. The switching frequency of the buck converter was 200 kHz and the emulation timestep was 50 ns, corresponding to 100 timesteps per switching cycle. With the intent to operate in real-time, Bhattacharya set the emulator clock frequency to the inverse of the timestep, i.e. 20 MHz. However, in general, emulators can operate faster or slower than real-time, depending on the relationship between the timestep and the emulator clock frequency.

Once a timestep is selected, analog dynamics are discretized to that timestep, resulting in a system of discrete-time equations. A common method is to convert transfer functions from the s to the z domain using Euler's method, as Bhattacharya did; alternatives include the bilinear transform and the zero-order hold approximation. These approaches can also be used when analog dynamics are more readily described using a system of differential equations.

A subset of nonlinear dynamical systems can be discretized with a simple extension: if a system's dynamics switches between various linear operating modes, then each of those modes can be discretized individually at compile-time (formalized in later work by Bhattacharya [40]). When the emulator is running, the appropriate mode is chosen in each emulation cycle, using the same underlying set of state variables. For example, a buck converter topology can be modeled with four linear operating modes, corresponding to all on/off combinations of the high- and low-side switches, and two state variables, corresponding to the capacitor voltage and inductor current.

When using oversampling for emulation, different sampling intervals can be used for different parts of a design. For example, A. Fernandez-Alvarez [41] built an emulator in which analog parts of a design were implemented by an on-FPGA processing system (PS), while digital parts were implemented in the FPGA's programmable logic (PL). Since the PS couldn't keep up with the PL on a cycle-by-cycle basis, the analog models had to be updated less frequently, and therefore with a larger timestep, so that the two domains would stay synchronized.

B. Variable Timestep Methods

Oversampling in emulation, as in simulation, typically requires multiple samples between digital events to achieve reasonable accuracy. However, the performance impact of these "analog-only" timesteps is generally worse in an emulator. When emulating an SoC design that is mostly digital, as many are, emulator resources are mostly committed to representing digital circuits, yet those resources sit idle during analog timesteps. Hence, a single oversampled analog model requiring fine time resolution can decimate an emulator's capacity to accelerate digital logic simulation, because most of its resources will end up sitting idle for most of the time.

A solution is to avoid analog timesteps, instead having the emulator step directly from one digital event (e.g., clock edge) to the next. For the link example, we would start with the efficient step response model of as much of the system as possible, since this model only evaluates on digital clock edges. However, if part of the system were nonlinear we would need another approach. We proposed a scheme to handle these nonlinear cases, coupling an event-driven emulation engine with a method to project the shape of analog waveforms between digital events.

As discussed in Section III-A, there are many options for feature vectors to represent a complex waveform between events; we found that a spline representation works well for emulation. Within one analog timestep we explicitly calculate a small number of evenly-spaced sample points and implicitly connect them using a spline interpolation. The value of the spline only needs to be calculated explicitly when the signal is sampled by a digital clock. Fig. 6 shows an overlay of the spline model of a waveform on top of a spice simulation of the same waveform. Although the frequency of analog timesteps is slower than the system's dynamics, the flexibility of the spline allows it to capture high-frequency behavior. Calculating the update from one set of spline points to the next requires more computation than taking several smaller time steps sequentially; however, with the spline interpolation the calculation is parallelizable so the FPGA's cycle time can stay short and analog-only time steps are not needed [19].

In addition to reducing or eliminating analog timesteps, it is also important that analog computation does not increase the FPGA cycle time. Floating-point multiplication is generally slow on an FPGA, so we use a tool called *svreal* to map real values in a fixed-point format [42]. *svreal* automatically keeps track of signal ranges and maps real values appropriately in order to take full advantage of the limited bit width of FPGA digital signal processing (DSP) blocks. By using the DSP blocks we are able to perform analog computation with few sequential multiplies, such as a multiply-accumulate operation, with FPGA cycle time similar to the digital emulation.

This spline-based approach yielded a $100\times$ reduction in the number of timesteps required to emulate analog parts of a

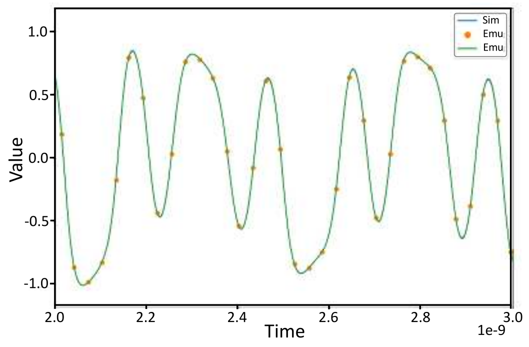

Fig. 6. Comparison between spice waveform and implicit spline representation from the model. Note that the spice waveform is nearly hidden because the two overlap. Dots show emulator timesteps; between each dot the emulator explicitly calculates 4 values in parallel. The smooth emulator line is the implicit spline interpolation through those 4 values, which is evaluated only on-demand [19].

high-speed link, including a lossy channel and multiple CTLEs with saturation nonlinearities. Thus, it sped up the FPGA emulation greatly. We discuss more examples and results using this approach in [43].

C. AMS Emulation Tools

Various tools have been used in constructing oversampled AMS models. Bhattacharya, for example, used Xilinx System Generator [44] to implement manually-derived discrete-time filters representing analog blocks. For AMS-specific abstractions, others (e.g. B. K. Mishra [45]) used Simscape Electrical [46], which converts schematics of linear and “switched linear” devices into oversampled synthesizable models.

Other AMS emulation tools have been described in publications, but have not been released. For example, Tertel and Hedrich [47] built a library of oversampled models including RC filters, amplifier stages, rectifiers, and a sample-and-hold. These models were integrated using a netlist format. W. Wu [48] took a generator-based approach, building a tool called WaveACE to construct wave digital filters for transistor-level circuits.

F. Nothaft [49] tackled the problem of re-using AMS simulation models (behavioral Verilog) in emulation. He developed a tool that converted floating-point numbers in existing models to fixed-point, using annotated range/resolution information. The tool was successfully applied to emulate a commercial cellular modem IC on Cadence Palladium, yielding a 120× speedup as compared to RTL simulation.

Unfortunately, of the tools described, only Xilinx System Generator and Simscape Electrical are publicly available, and they do not represent a complete emulation flow. This motivated us to develop and release what we believe is the first publicly available, complete framework for AMS emulation. The free, open-source framework consists of *msdsl* [50], a Python tool for generating synthesizable AMS models, and *anasymod* [51], a simulator-like abstraction of FPGA boards. A standalone synthesizable fixed- and floating-point library, *svreal* [42], discussed in Section IV-B, is used in the model generation process. The flow of creating models for FPGA emulation [43] is depicted in Fig. 7. *msdsl* provides a set of

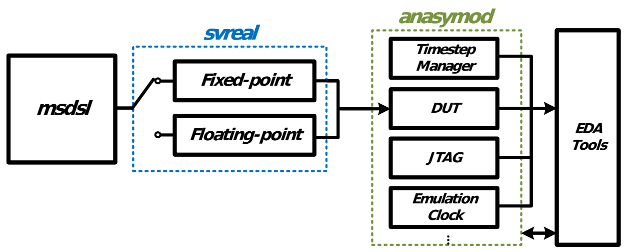

Fig. 7. Framework of FPGA AMS emulation model generation [43].

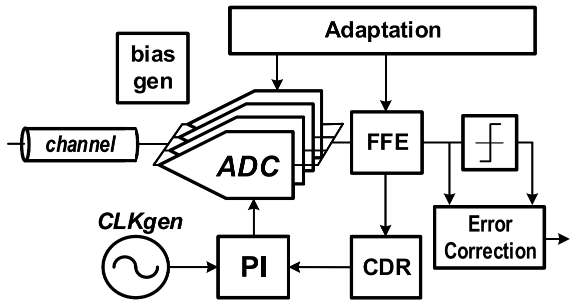

Fig. 8. Architecture of DragonPHY, our synthesizable high-speed link receiver, which served as a test case for the modeling techniques described in this paper.

functions that allows users to describe AMS blocks as *differential equations*, *netlists*, *transfer functions*, or *switched systems*, etc. The generated synthesizable HDL then leverages *svreal* to switch between fixed-point and floating-point representation to allow the flexible trade-off between accuracy and emulation throughput. *anasymod* provides emulation infrastructure that manages the emulation timestep, emulation clock speed, and test interfaces; it also communicates with EDA tools to generate the FPGA emulation bitstream for a given FPGA board. Details and examples of how to use the emulation framework are described in [43].

This framework supports event-driven emulation with a spline-based analog waveform representation, as described in Section IV-B. It has been successfully applied to six commercial designs, including an automotive magnetic sensor [52], and one academic design, a high-speed link receiver [53]. Speedups across these applications were typically two to three orders of magnitude as compared to existing simulations.

V. DragonPHY Example

To demonstrate the capability of the modeling techniques described in this paper, we present our results using them to validate DragonPHY [53], a synthesizable high-speed link we recently designed in TSMC’s 16nm technology. DragonPHY[4] is similar to the link described in Section II-B, except it doesn’t contain the CTLE/AGC, and feeds the channel output directly to the interleaved ADC.

As shown in Fig. 8, DragonPHY uses 16 time-interleaved ADCs organized into four banks, with each bank driven by

[4]Both the design and the emulator variants described in this section are available as open-source on GitHub (https://git.io/dragonphy).

one phase interpolator (PI). The PIs are essentially digitally-controlled phase rotators and are adjusted by changing the PI control codes so that they produce four equally-spaced clock phases. The ADCs in each bank are activated sequentially, with one sample taken on each rising edge of the PI clock. The ADC consists of a voltage-to-time converter (V2T) and a stochastic time-to-digital converter (STDC). The PI consists of an open-loop delay chain and a phase selection network followed by a phase blender. The DragonPHY architecture is split between an analog core, which contains the ADCs, PIs, and a bias generator, and a digital core, which contains DSP circuits to recover the transmitted data from the ADC samples. It is an open-source circuit generator, so we built a design verification suite that contains automatic regression tests to check block performance, and an FPGA emulator to check the full link performance.

The automatic regression tests are realized by using functional model templates. The regression tests are triggered every time a user makes changes to any part of the design. For example, if the SPICE netlist of the voltage-to-time converter changes, the new circuits are then extracted to the analog functional models. The functional models with updated parameters will be used for system-level simulations to validate the design change automatically.

Several analog functional models have been created for DragonPHY. For functional simulation we created five functional models: 1) V2T; 2) sample-and-hold (SnH); 3) phase blender; 4) bias generator;and 5) channel model. The V2T, phase blender, and bias generator use an amplifier template. For the V2T the bias voltage changes the voltage-in to time-out gain, and the PI works in phase space with the digital input controlling gain. The sample-and-hold uses a sampler template which has a filter before a sampler. The channel model uses the channel template to take advantage of its digital drive.

Only two analog models are used for the emulator. The entire analog signal chain from digital data into the transmitter to the sampled digital values were combined into one augmented channel model, and PIs were used to create the receiver's digitally controlled oscillator.

To create these models, we followed the general flow described throughout this paper for converting a spice netlist into a functional model:

1) Choose the type of model template to use; examples of model types are described in Sections III-A and III-B in addition to the emulation-specific models in Section IV.
2) Run a simulation of the spice circuit to extract parameters for the model. Sections III-B and III-C discuss testbenches for the characterization of spice netlists.
3) Validate that the extracted model matches the spice netlist behavior. Validation is discussed in Section III-C.

Fig. 9 shows the Verilog simulation results of the proposed ADC slice as an example of the proposed flow. Note that using functional models reduced simulation time by more than 10,000× as compared to the conventional SPICE simulation while providing sufficient accuracy.

While the functional models allowed the validation of the analog core, they started to run into speed issues when we tried to simulate the clock recovery and channel equalization loops for the full link. As described earlier, this required hundreds of thousands of cycles to settle. To enable validation of these loops, and test for bit error rates (BER), we created emulation models using the tools and flow described in Section IV-C.

One would immediately realize that there are different possible levels of abstraction when creating the emulation models for DragonPHY. We experimented with AMS emulation tools and created two emulation implementations for DragonPHY: a "low-level" emulator that swapped in synthesizable models for the ADCs, PIs, and channel, and a "high-level" emulator that lumped all of those behaviors together into a single model. The "low-level" architecture used variable timesteps that were determined at runtime, and its AMS models had a one-to-one mapping to the models that would be used in a conventional CPU-based simulation. The "high-level" architecture conceptually computed all 16 inputs and sampled outputs in parallel (once it knew the timing of the transmit and receive clocks), which made better use of parallelism and also had simpler event handling.In bit error rate (BER) measurements, both emulation models matched the CPU model within 7.5% over multiple jitter and noise conditions (Fig. 10) [43]. The low-level model ran at 5 Mb/s, and the high-level model ran at 80 Mb/s, allowing the full link training to complete within a second. At 80 Mb/s, this performance was more than 5,000× faster than our functional models [19] [43].

VI. Conclusion

The continued intertwining of analog and digital systems makes functional modeling of analog blocks more critical than ever, not only in the context of simulation but for emulation as well. Two main approaches have been proposed to create such models: fixed-timestep oversampling and event-driven modeling with analog waveforms represented by feature vectors. Oversampling is the traditional approach but suffers from poor performance because it generates many additional simulation/emulation events. Although the event-driven approach can be more complex, it has the potential to achieve high modeling fidelity while reducing the number of required events. This is especially important in emulation, where analog events have a stronger impact on performance as compared to simulation.

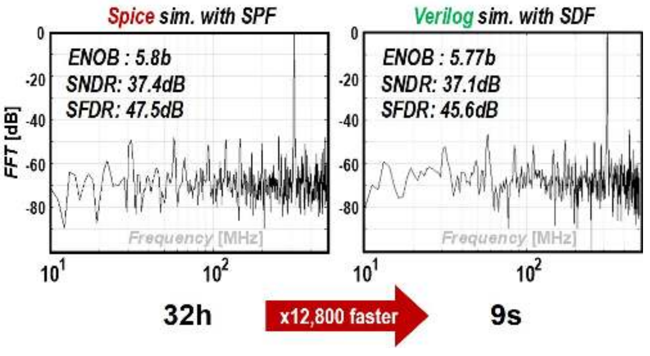

Fig. 9. AC simulation results of the DragonPHY ADC. (a) SPICE simulation with extracted SPF. (b) Verilog simulation with extracted SDF.

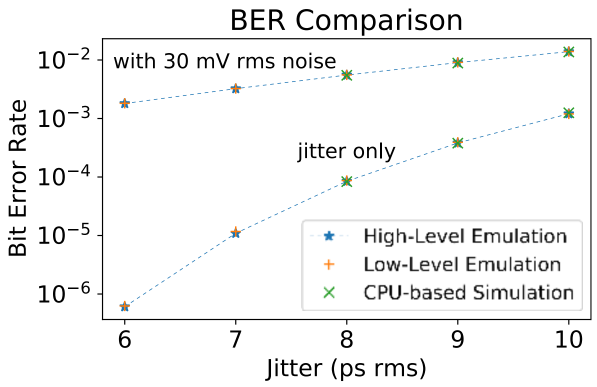

Fig. 10. The BERs predicted by both emulation models closely match those of the CPU simulation, despite running orders of magnitude faster. We originally presented these results, along with more details, in [43].

It is critical to ensure that analog functional models closely track the behavior of the circuits they represent. The first step towards that goal is to create a testbench that extracts model parameters from a circuit of interest. Those parameters can then be passed into a flexible model template in order to generate a functional model. Finally, the generated model can be validated by extracting its model parameters with the same testbench and comparing them to those extracted from the circuit itself.

Applying these techniques to a high-speed link design has enabled us to run a full link validation, using both simulation and emulation, in a fraction of the time needed by more conventional approaches.

Acknowledgment

We would like to thank Stefanos Sidiropoulos and Byong Chan Lim for their contributions to this paper.

References

[1] C.-J. Seger, R. Jones, J. O'Leary, T. Melham, M. Aagaard, C. Barrett, and D. Syme, "An industrially effective environment for formal hardware verification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 9, pp. 1381–1405, 2005.
[2] D. Lin, E. Singh, C. Barrett, and S. Mitra, "A structured approach to post-silicon validation and debug using symbolic quick error detection," in *2015 IEEE International Test Conference (ITC)*, 2015, pp. 1–10.
[3] J. Kim, K. D. Jones, and M. A. Horowitz, "Variable domain transformation for linear PAC analysis of mixed-signal systems," in *2007 IEEE/ACM International Conference on Computer-Aided Design*, 2007, pp. 887–894.
[4] Cadence. Solutions for Mixed-Signal SoC Verification Using Real Number Models. [Online]. Available: https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/solutions/mixed-signal-verification-wp.pdf
[5] L. W. Nagel and D. Pederson. (1973) SPICE (Simulation Program with Integrated Circuit Emphasis). [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/1973/22871.html
[6] Cadence Design Systems. Spectre. [Online]. Available: https://www.cadence.com/en_US/home/tools/custom-ic-analog-rf-design/circuit-simulation/spectre-simulation-platform.html
[7] Synopsys. HSPICE. [Online]. Available: https://www.synopsys.com/implementation-and-signoff/ams-simulation/primesim-hspice.html
[8] P. Weil and L. P. McNamee, "A nonlinear macromodel for operational amplifiers," *International Journal of Circuit Theory and Applications*, vol. 6, no. 1, pp. 57–64, 1978. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cta.4490060108
[9] A. Newton and A. Sangiovanni-Vincentelli, "Relaxation-based electrical simulation," *IEEE Transactions on Electron Devices*, vol. 30, no. 9, pp. 1184–1207, 1983.
[10] Cadence Design Systems. Spectre FX. [Online]. Available: https://www.cadence.com/en_US/home/tools/custom-ic-analog-rf-design/circuit-simulation/spectre-fx-simulator.html
[11] Cadence Design Systems. Spectre XPS. [Online]. Available: https://www.cadence.com/en_US/home/tools/custom-ic-analog-rf-design/circuit-simulation/spectre-extensive-partitioning-simulator-xps.html
[12] Synopsys. PrimeSim Pro. [Online]. Available: https://www.synopsys.com/implementation-and-signoff/ams-simulation/primesim-pro.html
[13] Synopsys. PrimeSim XA. [Online]. Available: https://www.synopsys.com/implementation-and-signoff/ams-simulation/primesim-xa.html
[14] Cadence Design Systems. Spectre APS. [Online]. Available: https://www.cadence.com/en_US/home/tools/custom-ic-analog-rf-design/library-characterization/spectre-accelerated-parallel-simulator.html
[15] Cadence Design Systems. Spectre X. [Online]. Available: https://www.cadence.com/en_US/home/tools/custom-ic-analog-rf-design/circuit-simulation/spectre-x-simulator.html
[16] Siemens. AFS (Analog FastSPICE). [Online]. Available: https://eda.sw.siemens.com/en-US/ic/analog-fastspice/
[17] Primarius. NanoSpice. [Online]. Available: https://www.primarius-tech.com/en/products/NanoSpice
[18] R. Cottrell, "Event-driven behavioural simulation of analogue transfer functions," in *Proceedings of the European Design Automation Conference*. Los Alamitos, CA, USA: IEEE Computer Society, mar 1990, pp. 240,241,242,243. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/EDAC.1990.136652
[19] S. Herbst, "An Open-Source Framework for FPGA Emulation of Analog/Mixed-Signal Integrated Circuit Designs," Ph.D. dissertation, Stanford University, 2021. [Online]. Available: http://purl.stanford.edu/gj828vr5382
[20] J.-E. Jang, M.-J. Park, D. Lee, and J. Kim, "True Event-Driven Simulation of Analog/Mixed-Signal Behaviors in SystemVerilog: A Decision-Feedback Equalizing (DFE) Receiver Example," in *Proceedings of the IEEE 2012 Custom Integrated Circuits Conference*, 2012, pp. 1–4.
[21] Scientific Analog, Inc. XMODEL. [Online]. Available: https://www.scianalog.com/xmodel/
[22] B. C. Lim and M. Horowitz, "Error Control and Limit Cycle Elimination in Event-Driven Piecewise Linear Analog Functional Models," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 1, pp. 23–33, 2016.
[23] L. Yang and C.-J. Shi, "FROSTY: a fast hierarchy extractor for industrial CMOS circuits," in *ICCAD-2003. International Conference on Computer Aided Design (IEEE Cat. No.03CH37486)*, 2003, pp. 741–746.
[24] C.-W. Ho, A. Ruehli, and P. Brennan, "The modified nodal approach to network analysis," *IEEE Transactions on Circuits and Systems*, vol. 22, no. 6, pp. 504–509, 1975.
[25] C.-J. R. Shi, "Mixed-signal system-on-chip verification using a recursively-verifying-modeling (RVM) methodology," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, 2010, pp. 1432–1435.
[26] B. C. Lim and M. Horowitz, "An analog model template library: Simplifying chip-level, mixed-signal design verification," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 1, pp. 193–204, 2019.
[27] K. Kundert and H. Chang, "Model-based functional verification," in *Design Automation Conference*, 2010, pp. 421–424.
[28] B. C. Lim, J. Kim, and M. A. Horowitz, "An efficient test vector generation for checking analog/mixed-signal functional models," in *Design Automation Conference*, 2010, pp. 767–772.
[29] H. Chang and K. Kundert, "Verification of Complex Analog and RF IC Designs," *Proceedings of the IEEE*, vol. 95, no. 3, pp. 622–639, 2007.
[30] L. Truong, S. Herbst, R. Setaluri, M. Mann, R. Daly, K. Zhang, C. Donovick, D. Stanley, M. Horowitz, C. Barrett, and P. Hanrahan, "fault: A Python Embedded Domain-Specific Language for Metaprogramming Portable Hardware Verification Components," in *Computer Aided Verification*, S. K. Lahiri and C. Wang, Eds. Cham: Springer International Publishing, 2020, pp. 403–414.
[31] D. Stanley. (2021) fixture: A library of templates for analog blocks and strategies to model them in a digital environment. [Online]. Available: https://github.com/standanley/fixture
[32] B. C. Lim, J.-E. Jang, J. Mao, J. Kim, and M. Horowitz, "Digital analog design: Enabling mixed-signal system validation," *IEEE Design Test*, vol. 32, no. 1, pp. 44–52, 2015.
[33] Cadence, "AMS Design and Model Validation User Guide."

[34] S. Asaad, R. Bellofatto, B. Brezzo, C. Haymes, M. Kapur, B. Parker, T. Roewer, P. Saha, T. Takken, and J. Tierno, "A Cycle-Accurate, Cycle-Reproducible Multi-FPGA System for Accelerating Multi-Core Processor Simulation," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 153–162. [Online]. Available: https://doi.org/10.1145/2145694.2145720

[35] Cadence, "Cadence Palladium." [Online]. Available: https://www.cadence.com/en_US/home/tools/system-design-and-verification/acceleration-and-emulation/palladium-z1.html

[36] Synopsys, "Synopsys ZeBu." [Online]. Available: https://www.synopsys.com/verification/emulation.html

[37] M. Graphics, "Mentor Veloce Emulation Platform." [Online]. Available: https://www.mentor.com/products/fv/emulation-systems/

[38] R. M. Sanchez, B. T. Reyes, A. L. Pola, and M. R. Hueda, "An FPGA-based emulation platform for evaluation of time-interleaved ADC calibration systems," in *2016 IEEE 7th Latin American Symposium on Circuits Systems (LASCAS)*, 2016, pp. 187–190.

[39] R. Bhattacharya, S. Biswas, and S. Mukhopadhyay, "FPGA based chip emulation system for test development of analog and mixed signal circuits: A case study of DC–DC buck converter," *Measurement*, vol. 45, no. 8, pp. 1997 – 2020, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0263224112001923

[40] R. Bhattacharya, S. Kumar, and S. Biswas, "Fault diagnosis in switched-linear systems by emulation of behavioral models on FPGA: A case study of current-mode buck converter," *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, vol. 31, no. 5, p. e2314, 2018, e2314 JNM-17-0113.R1. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/jnm.2314

[41] A. Fernandez-Alvarez, M. Portela-Garcia, and M. Garcia-Valderas, "FPGA-based HW/SW co-simulation system for mixed-signal circuits," in *2016 Conference on Design of Circuits and Integrated Systems (DCIS)*, 2016, pp. 1–6.

[42] S. Herbst. (2021) svreal. [Online]. Available: https://git.io/svreal

[43] S. Herbst, G. Rutsch, W. Ecker, and M. Horowitz, "An Open-Source Framework for FPGA Emulation of Analog/Mixed-Signal Integrated Circuit Designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Aug 2021.

[44] Xilinx, "Model-Based DSP Design using System Generator," 2020. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_1/ug897-vivado-sysgen-user.pdf

[45] B. K. Mishra, S. Save, and R. Mane, "A Frame Work for Model Based Designing of Analog Circuits Using Simulink," in *Proceedings of the International Conference & Workshop on Emerging Trends in Technology*, ser. ICWET '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 1225–1228. [Online]. Available: https://doi.org/10.1145/1980022.1980290

[46] MathWorks. (2021) Simscape Electrical User's Guide. [Online]. Available: https://www.mathworks.com/help/pdf_doc/physmod/sps/sps_ug.pdf

[47] P. Tertel and L. Hedrich, "Real-time emulation of block-based analog circuits on an FPGA," in *2017 14th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, 2017, pp. 1–4.

[48] W. Wu, Y. Chen, Y. Ma, C. J. Liu, J. Jou, S. Pamarti, and L. He, "Wave digital filter based analog circuit emulation on FPGA," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2016, pp. 1286–1289.

[49] F. A. Nothaft, L. Fernandez, S. Cefali, N. Shah, J. Rael, and L. Darnell, "Pragma-Based Floating-to-Fixed Point Conversion for the Emulation of Analog Behavioral Models," in *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '14. IEEE Press, 2014, p. 633–640.

[50] S. Herbst. (2021) msdsl. [Online]. Available: https://git.io/msdsl

[51] G. Rutsch, S. Herbst, and S. Saravanan. (2021) anasymod. [Online]. Available: https://git.io/anasymod

[52] G. Rutsch, S. Fontanesi, S. Herbst, S. Tan Hee Yeng, A. Possemato, G. Formato, M. Horowitz, and W. Ecker, "Boosting mixed-signal design productivity with FPGA-based methods throughout the chip design process," in *Design and Verification Conference in Europe*, 2020. [Online]. Available: https://dvcon-europe.org

[53] S.-J. Kim, Z. Myers, S. Herbst, B. Lim, and M. Horowitz, "20-GS/s 8b Analog-to-Digital Converter and 5-GHz Phase Interpolator for Open-Source Synthesizable High-Speed Link Applications," *IEEE Solid-State Circuits Letters*, vol. 3, no. 11, pp. 518–521, 2020.

Daniel Stanley is pursuing a Ph.D. in Electrical Engineering at Stanford focusing on tools for analog and mixed-signal circuit design. He received his Bachelor's degree in Electrical Engineering from Princeton University in 2018 and his Master's in Electrical Engineering from Stanford University in 2020.

Can Wang is pursuing his Ph.D. in Electrical Engineering at Stanford University with a research focus on mixed-signal circuits design, design automation etc. Prior to the Ph.D. program, he received his B.Eng. from the Chongqing University of Posts and Telecommunications, China, and Master of Philosophy degree from The Hong Kong University of Science and Technology, Hong Kong, in 2016 and 2020, respectively.

Sung-Jin Kim received his B.S. and M.S. in Electrical Engineering from KAIST, South Korea in 2008 and 2010 respectively, and he recently completed a Ph.D. in Prof. Mark Horowitz's group at Stanford University (2021). Prior to starting the Ph.D. program, he worked for Samsung Electronics as a mixed-signal circuit designer from 2010 to 2016. He is interested in synthesizable analog circuits and automated design flow for high-speed serial links (SerDes).

Steven Herbst recently completed a Ph.D. in Prof. Mark Horowitz's group at Stanford University (2021). His research aimed to make the chip design process more accessible through open-source tools for mixed-signal circuit design. Prior to the Ph.D. program, he spent five years in industry as an analog designer for PCBs and ICs (2011-2016). Steven holds S.B. and M.Eng. degrees in electrical engineering from MIT (2010, 2011).

Jaeha Kim Jaeha Kim is a professor at Seoul National University. In 2015, he founded Scientific Analog Inc., Palo Alto, CA, USA, an EDA company focusing on analog/mixed-signal verification. His research interest includes low-power mixed-signal systems and their design methodologies. He was cited as the Top 100 Technology Leader of Korea by the National Academy of Engineering of Korea (NAEK) in 2020. He was a recipient of the Takuo Sugano Award for Outstanding Far-East Paper at the 2005 International Solid-State Circuits Conference (ISSCC) and the Low Power Design Contest Award at the 2001 International Symposium on Low Power Electronics and Design (ISLPED).

Mark Horowitz (F'00) is the Yahoo! Founders Professor at Stanford University and was chair of the Electrical Engineering Department from 2008 to 2012. He co-founded Rambus, Inc. in 1990 and is a fellow of the IEEE and the ACM and a member of the National Academy of Engineering and the American Academy of Arts and Science. Dr. Horowitz's research interests are quite broad and span using EE and CS analysis methods to problems in molecular biology to creating new design methodologies for analog and digital VLSI circuits.