VERILOG PIECEWISE LINEAR BEHAVIORAL MODELING FOR MIXED-SIGNAL VALIDATION

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING

AND THE COMMITTEE ON GRADUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

Sabrina Liao

May 2014

This dissertation is online at: http://purl.stanford.edu/pb381vh2919

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Mark Horowitz, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Boris Murmann

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Bruce Wooley

Approved for the Stanford University Committee on Graduate Studies.

Patricia J. Gumport, Vice Provost for Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.

Abstract

Today's mixed-signal systems-on-chip (SoC's) are complex entities that contain very tightly coupled analog and digital circuits. Validating these complex systems requires simulating the entire design through a large number of test vectors, and each test vector might in turn lead to a long simulation. Due to traditionally differing views of validation as well as design tools for analog and digital, validation of an SoC is not easily accomplished. Behavioral modeling is an attractive approach that tries to address this issue by replacing analog circuits with high-level functional models to speed up simulation while retaining some of the analog behavior. This dissertation proposes a method for creating these models in an event-driven, digital modeling language.

The model-writing strategy is three-pronged. First, to fit into a digital simulator that simulates mostly unidirectional designs, the analog circuits must be broken into sub-blocks with unidirectional ports. Second, continuous time analog signals need to have a suitable representation in a discrete-time simulator. A piecewise linear format in which value-slope pairs are updated at discrete intervals is employed as an example. Third, time integration must be avoided in the computation of model outputs. By leveraging the linear (possibly with small nonlinearity) intent of analog circuits, a companion method of efficiently calculating output in piecewise linear format is developed. The overall result is analog behavioral models that are pin-accurate, fast to simulate and capture the key dynamics in analog circuits.

Models of various types of circuits (a simple RC filter, a phase interpolator, a comparator and a current DAC) are composed to illustrate the wide applicability of the proposed modeling method. A 250MS/s track and hold, a 2.5-1.8V switching regulator, and a 1GHz PLL are also modeled to verify the preservation of important circuit behaviors as well as to gauge these models' computation complexity.

Acknowledgements

William Arthur Ward once said: “Feeling gratitude and not expressing it is like wrapping a present and not giving it.” I am grateful for this opportunity to give thanks to the people who have helped to bring this dissertation to fruition.

First and foremost, I would like to thank Professor Mark Horowitz for his valuable mentorship and tutelage. His dedication to his students despite his many other academic duties is awe-inspiring. He has the amazing ability to explain the most confusing concepts in the clearest of terms, and somehow can always find better words to express an idea. I have learned much from him over the years, and still have much to learn. It has truly been a privilege to work with him.

I would also like to thank Professor Boris Murmann and Professor Bruce Wooley for reading this thesis and for delivering enlightening lectures on circuits. These lectures have taught me a great deal and they have been one of the highlights at Stanford. I am honored to have Professor James Harris chair my orals committee and to have Professor Ada Poon serve on my orals committee. I appreciate the technical discussions and guidance from Professor Jaeha Kim when I first embarked on this journey of research.

My special thanks also go out to Sakshi Arora for providing the design of the dc-dc converter, and to John Brunhaver for selflessly organizing his digital design data so that this dissertation could benefit from it.

I have made great friends – Vaibhav Tripathi, Andrew Danowitz, Harendra Guturu, to name a few. Life as a graduate student wouldn't have been as pleasant without you. I will also cherish the memories I have made with everyone in the VLSI research group, including Teresa Lynn and Mary Jane Swenson who have tirelessly managed the administrative details, thus giving me freedom to stay focused on research.

To my wonderful parents: Thank you for always being there, showering me with love, encouragement and affirmation. You have done the impossible by giving me roots as well as wings.

This research is generously supported by the Stanford Graduate Fellowship, the Natural Sciences and Engineering Council of Canada Postgraduate Scholarships and the Stanford Rethink Analog Design initiative.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

There are two types of circuits in modern systems-on-chip (SoC's): analog and digital. Traditionally, different abstraction, design methods and tools are used for each type. For a long time, this approach worked well because the designs were on different chips, or weakly coupled, such that analog circuits can be designed and validated in isolation from the digital circuits. Over the past decade, however, the constant search for smaller, faster and less power hungry SoC's has brought analog and digital ever closer together. With aggressive scaling, designers saw benefit in the smaller form factor, flexibility and better noise sensitivity of digital circuits and began to implement a number of traditionally analog functions using digital circuits. For instance, continuous time equalization techniques in clock and data recovery circuits are supplemented with digital adaptive equalization algorithms [4]; in other cases, analog equalization techniques are complete replaced with an ADC frontend and extensive digital signal processing [5]. Phase-locked loops (PLL's) have mostly been analog entities, however there has been tremendous interest recently in achieving frequency synthesis using mostly digital PLL's [6] [7]. Since scaling degrades matching[1], a popular solution is to use digital circuits to reduce analog matching errors as seen in digitally assisted data converters [3] [2]. The

[1] Scaling improves analog matching per unit area [99], however minimum size devices often have the higher performance that designers seek and the overall matching of these devices degrades as technology scales.

demand for better power performance has also brought digital power management on chip. Wireless transceivers use profiled power management units to reduce active power consumption [8] and many digital control techniques such as digital peak voltage/peak current [84], digital pulsewidth modulation [85] and constant on/off time control [86] have been developed to rival traditional analog control loops in power converters. In all these cases, analog and digital circuits are no longer isolated entities, but rather two tightly coupled and constantly interacting components of a complex system.

Any complex system requires validation as a whole, but doing so is difficult because of the traditionally differing and incompatible design/validation methods of analog and digital circuits [80]. This thesis proposes using high-level analog functional models written in SystemVerilog to address this validation gap, and provides a method for creating these models.

1.1 Analog Validation

Analog circuits are validated at the device level, using a simulator (e.g. SPICE) that uses compact device models to evaluate the behavior of the circuit schematic. Designers are interested in measuring some characteristics of the analog circuit's output and if a set of required performance is achieved, then the circuit is considered to be validated [60]. For example, in a sample and hold, designers might drive the input with a sinusoid slightly below the Nyquist frequency and measure the circuit's spurious free dynamic range (SFDR), signal to noise ratio (SNR), power and output swing. For a regenerative latch, the set of test signals might be a series of constant voltages with varying amplitude. The performance specification might involve an upper limit on the input to output delay for different input voltages. Designers simulate the circuit using the various types of analyses available in SPICE (such as DC, transient, AC, PAC, PSS, PNOISE) and inspect that all simulation results meet spec. Often times, the simulation

space also extends to the entire gamut of process corners, temperatures and supply voltages [60].

Despite the large variety of characteristics that can be measured of analog responses, analog outputs are smooth functions of the circuit's input. Therefore, designers are usually not concerned with enumerating every possible test stimulus such that the input space is finely sampled. Typically, a small set of test inputs is sufficient to fully characterize the analog response surface, because the result from one input case yields a great deal of information on results from similar inputs. Using this small set of test cases, analog validation can be completed for each performance metric even if each analog test case (especially those that require transient simulations) might be time-consuming.

1.2 Digital Validation

The goals of digital testing are very different from those of analog test. Rather than trying to extract a number of parameters of the circuit, the digital designer wants to know whether a large collection of logic does the correct function. A digital design with N state variables has 2^N mutually independent states – that is, the correct function of one state yields no information on the correct function of any of the other states. In other words, the result surfaces of digital circuits are not smooth like analog circuits, and it is hard to predict what the machine will do in a specific situation without testing that situation. Validating a digital circuit, therefore, involves exercising every possible state that the design may go through. Commonly used digital simulators include VCS and ModelSim.

Even though the simulation time of a single test vector may be small, a digital design may contain millions of logic gates and the number of test vectors necessary to completely verify a design is very large – sometimes impractically so. As a result, coverage is oftentimes sacrificed by running only as large a set of test vectors as time will

allow. In addition, the verification is concerned with function only; timing information such as delay and skew are handled separately by static checking tools.

1.3 Mixed-signal Validation Challenge

Given the vastly different validation approach for analog and digital circuits, it is not surprising that the validation of a mixed-signal design is difficult. In order to run millions of test vectors with reasonable speed, the digital component of an SoC must be validated in a discrete-time, event-driven, digital simulator [87]. At the same time, active collaboration between analog and digital – for example, digital calibration – dictates that the digital component must be aware of the analog component's behavior in order to determine its own behavior. This means that the analog component must also be simulated for every test vector sent to the system. Difficulties arise in this case since first, there is a clear disconnect between analog and digital simulators and second, time-consuming analog transient simulation means that either the coverage of the validation suite will be very limited or the validation effort cannot be completed within any practical amount of time.

There have been a number of approaches that address mixed signal validation and these will be reviewed in Chapter 2. Among these approaches, behavioral modeling seems to be the only solution fast enough for mixed-signal validation. To summarize the contribution of this work – namely providing some formalism to behavioral modeling – Chapter 3 will lay down general guidelines developed to writing behavioral models. Next, to see how these guidelines can be applied, Chapter 4 will discuss models for different types of circuits. To be more specific and evaluate the models' performance, Chapter 5 will demonstrate the viability of the proposed model-writing strategies through several real circuit examples. An analysis of the models' efficiencies will also be presented. Concluding remarks are provided in Chapter 6.

Chapter 2

Mixed-Signal Validation Background

System-level validation of mixed-signal SoC is challenging [16], and not surprisingly, it is an active area of research. To better understand the issues that need to be addressed, this chapter will begin with a discussion of traditional analog and digital simulators (Sections 2.1 and 2.2 respectively). Given the difference in simulation tools, three different approaches to mixed-signal validation will be compared. One approach focuses on modifying the simulators to make the analog simulator run faster, and be able to connect to a digital simulator. The macromodeling approach tried to replace the device-level network that is solved by the analog simulator with a simpler one that has equivalent behavior. Lastly, behavioral modeling completely replaces the circuit schematics with a functional description. While these three methods have advantages and disadvantages, and behavioral modeling is the solution with the highest hopes of being fast enough for mixed-signal SoC validation.

2.1 Analog (SPICE) Simulator

Analog designers use SPICE-like simulators to help them predict and analyze the behavior of a circuit. Starting from a schematic and very accurate device models, SPICE

simulators compute the voltage and current values at each node for all instants of time. KVL and KCL nodal equations are generated first. Due to devices such as transistors in the design, these equations are usually a set of differential-algebraic equations (DAE's), whose general form is shown below [1]:

$$\frac{dq(\boldsymbol{v},t)}{dt} = i(\boldsymbol{v},\boldsymbol{u},t) \tag{1}$$

v = **v**(t) is a vector of nodal voltages and branch currents; **u** = **u**(t) is a vector of inputs; t is time; q(•) is an operator describing charge; and i(•) is an operator describing current. Numerical integration methods need to be invoked in order to solve these equations. First, time-discretization is applied using algorithms such as backward Euler, trapezoidal or second order Gear. For example, if backward Euler integration were used, the system of (1) will yield an algebraic equation

$$\boldsymbol{v}_{t+\Delta t} = \boldsymbol{v}_t + \Delta t \cdot i(\boldsymbol{v}_{t+\Delta t}, \boldsymbol{u}_{t+\Delta t}, t+\Delta t) \tag{2}$$

This equation needs to be solved for the unknown vector of node voltages and branch currents, $\mathbf{v}_{t+\Delta t}$, at time t+Δt where Δt is the time step. This is equivalent to finding zeros of a nonlinear operator:

$$F(\boldsymbol{v}) = \boldsymbol{v}_{t+\Delta t} - \boldsymbol{v}_t - \Delta t \cdot i(\boldsymbol{v}_{t+\Delta t}, \boldsymbol{u}_{t+\Delta t}, t+\Delta t) = 0 \tag{3}$$

The solution of (3) can be obtained with an iterative algorithm such as the Newton-Raphson method or successive chords method. Using, for example, the Newton-Raphson method, a single iteration involves solving the equation:

$$J_F\left(\boldsymbol{v}^{i+1}\right) - J_F\left(\boldsymbol{v}^{i}\right) = -F\left(\boldsymbol{v}^{i}\right) \tag{4}$$

J_F is the Jacobian matrix for operator F and i is the iteration number. The solution for (4)

can be found using direct matrix methods such as LU factorization. The iteration stops when the difference between v^i and v^{i+1} is smaller than some predefined value.

In a transient simulation, the construction of the system of equations, the discretization of this set of DAE's, and the iterative matrix solves are carried out at every time step. In accordance to analog designers' expectations, SPICE simulators prioritize high-accuracy above anything else [1]. This in turn means highly-accurate and detailed device models, creating a single system of equations for the entire circuit, using very fine time steps during discretization, and calling direct matrix solvers that are computationally expensive. All these traits as well as the purpose of the analog simulator are considerably different from those of the digital simulator, as the next section will show.

2.2 Digital Simulator

Digital designers work with circuits that have already been validated at the transistor level to design a larger system that performs a digital function. The purpose of the digital simulator, hence, is to evaluate the logical function that the design created. These designs are generally written in a hardware description language (HDL) like VHDL or Verilog, and simulated in a digital simulator such as VCS or ModelSim. The HDL is a functional model of the design and is later mapped to standard cells implementing different logical functions using digital synthesis tools.

In simulation, the distinct changes in Boolean values are tracked. As a result, digital simulators are discrete-time and only evaluate the functional models when inputs change. An internal time-wheel mechanism keeps track of when the input changes occur - these are called triggering events. When such an event is encountered, the simulator wakes up and computes the new outputs. If the outputs change, the changes are scheduled to take effect at a future time as determined by the delays of the gates. Then the time-wheel advances. Periods with no change in Boolean values (e.g. between clock cycles of

a flip-flop) are skipped in the simulation until the next triggering event – this is called the selective trace technique [16]. Thus, as time progresses the evolution of the system's state is determined and the logic function of the design is verified.

The fundamental difference between an analog and a digital simulator can be illustrated using an inverter with some clock input (see Figure 1). To the analog simulator, an inverter is two transistors. The SPICE algorithm will be concerned with solving the output waveform very accurately at each time instance, and will be doing computations at all times (although the figure shows constant time steps for computation, more advanced analog simulators have adopted variable time steps so that computation occurs less often when the signal is not fluctuation much). The digital simulator, on the other hand, only computes the output of the inverter when the input changes its Boolean value. The correct timing of the output will come from the delay of the inverter annotated in its functional model. Therefore, for the same amount of time simulated, an analog simulation is much more computationally intensive and time-consuming.

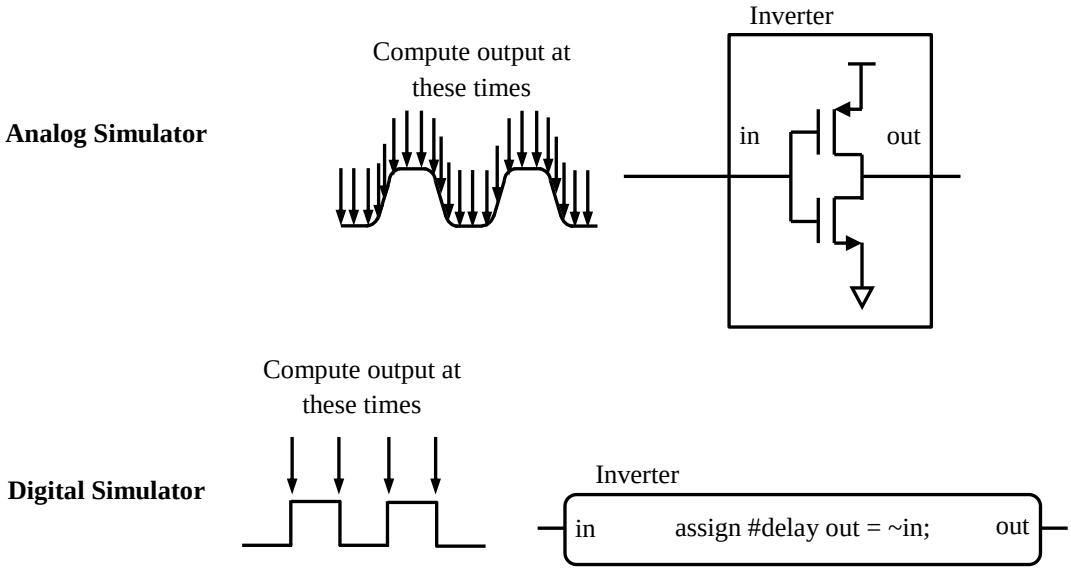

Figure 1 – Different treatment of an inverter by analog and digital simulators

2.3 Modified Simulators

The previous two sections have demonstrated that analog and digital simulators use different algorithms and have very different computational costs. This clear disconnect makes mixed-signal validation difficult. The following sections review research that concentrates on altering the simulators to make them more suitable for mixed-signal simulations. Fast SPICE involves a collection of techniques to accelerate the basic SPICE engine so that larger designs (possibly including digital as well) can be simulated. The piecewise-linear simulator speeds up simulation by using a piecewise-linear description of the circuit network instead of the full system of nonlinear DAE's. However, simulating digital circuits with analog simulators is difficult even with efficient algorithms and simplified nonlinear DAE's. Therefore, the mixed-mode simulators focus on creating an appropriate interface between an analog and a digital simulator so that the two different types of circuit can be simulated with their native engines.

2.3.1 Fast SPICE

The conventional SPICE simulator, whose mathematical engine was briefly outlined in Section 2.1, provides a means of modeling integrated circuits with the highest possible accuracy. Because of this constraint on accuracy, SPICE simulators must use full equation-based device models as well as high complexity direct linear solvers on a single large matrix that represents the entire design. All this results in a simulator with a complexity of $O(n^{1.5\text{-}2})$ where n is the number of nodes in the circuit and a limited capacity of about 100,000 active elements (MOSFETs) [1]. With modern SoC designs reaching an active element count of from 100 million to 1 billion MOSFET devices and a parasitic devices count in the millions, performing full-chip functional simulation is well beyond the capabilities of conventional SPICE. On the other hand, the accuracy requirement for full-chip functional validation (especially for the digital parts of an SoC) is not as stringent [1]. The accelerated transistor-level ("fast SPICE") simulator, therefore,

leverages this fact and applies three general methodologies to speed up simulation. These methodologies may be classified as matrix-based, graph-based and circuit-based.

The matrix-based approach is equally applicable to both analog and digital circuits. At its core, it fundamentally argues that after all other optimizations, there was going to be a system of equation that needed to be solved and hence developing faster matrix solution algorithms will lead to shorter simulation time. For instance, hMETIS [13] reorders nodes during matrix factorization and linear solves to produce fill-reduced sparse matrices that are easier to solve. Furthermore, matrix preconditioning techniques such as incomplete LU factorization and multi-grid preconditioning [1] can be used on sparse matrices to reduce solve time. There are also fill-in based matrix partitioning algorithms, such as [9], that review the fill-in patterns and break down the problem into smaller pieces such that the sub-problems are easier to solve than the single matrix.

The graph-based approach contains an eclectic selection of techniques that rely on treating the design as a graph and drawing inspiration for acceleration from hierarchical circuit data storage. It has been observed in [19] that simulating a very large circuit on a computer with a virtual memory leads to a large amount of paging due to the near random distribution of data in memory. By maintaining hierarchy in a design, and careful memory allocation, data structures for a given sub-circuit will be in a smaller number of pages, and hence will reduce overheads and improve the run-time of the simulator. Therefore, efficient methods for processing hierarchical netlist have shown to improve speed and capacity, most predominantly in the simulation setup phase [1]. Isomorphic matching techniques [14] [15] also fall into the graph-based category. Because the netlist data is stored hierarchically, sufficiently identical circuit blocks can be found during the transient simulation phase, and repeated simulation is avoided. The isomorphic matching technique is particularly well-developed for identifying highly regularized circuit blocks like memory arrays, leading to methodologies such as hierarchical array reduction [10] [11] that can improve memory simulation speeds by up to 50x.

The circuit-based viewpoint targets the acceleration of SPICE for digital circuits. The method is a two-step process. First separate the circuits, automatically or semi-automatically, with respect to their functional characteristics – analog or digital. Second, develop and apply appropriate optimization and simulation strategies for the two circuit types separately. The circuit partition algorithms have evolved over the years. Initially, simple methods such as grouping channel-connected MOSFETs proved to be effective in predominantly digital circuits connected to ideal voltage sources [1]. More recent advances in partition methods introduce digital rail detection as well as topological analysis [9]. Once the digital partitions are found, specialized simulation strategies can be applied to them. For example, approximate device models can be utilized instead of the full equation-based models. Some common simplified device models include lookup tables of empirical gate delays [18] [20], ideal current source in parallel with a conductance exhibiting piecewise-polynomial characteristics [21] and DC operation region dependent current models [20] [31]. Another strategy is to use different matrix-solving algorithms as well as integration time-points than the analog partition. For instance, relaxation-based algorithms in [17] discard coupling sub-matrices in the full matrix so that the decoupled matrices are faster to solve. In [22], the direct SPICE method with network separation is carried out so that relaxation is only applied when the coupling matrices are weak (i.e. close to 0). The coupling is also re-assessed dynamically for the duration of the simulation. In addition, this de-coupling allows latent sub-circuits (circuits whose node voltages and branch current don't change much on successive iterations) to use larger time-steps of integration than those of critical analog circuits and hence increase simulation speed. This is called the multirate integration technique because in the basic SPICE engine, the entire design uses the same integration time-points when solving the system of DAE's.

Though the three approaches (matrix, graph and circuit) have been presented separately, many fast SPICE simulators deploy all three simultaneously. Another technology that fast SPICE simulators benefit from is parallel computing. From the

matrix-based point of view, parallel and multi-threaded linear solvers can be employed to speed up simulation. From the circuit-based perspective, the partitioned sub-systems can be simulated in parallel on several processors. Some commercially available Fast SPICE simulators are listed in Table 1. Note that these simulators achieve an order of magnitude acceleration compared to conventional SPICE.

Table 1 - Some Commercially Available Fast SPICE Simulators

Tool	Company	Comments
Spectre APS	Cadence	10x faster than Spectre Proprietary full-matrix solver Multi-thread / multi-core
Spectre XPS	Cadence	Announced on October 9, 2013 Advanced partitioning/model reduction
Virtuoso UltraSim	Cadence	Hierarchical storage Isomorphic and adaptive partitioning Automatic parasitic reduction
Analog FastSPICE Platform	Berkeley Design Automation	5-10x faster than SPICE Proprietary full-matrix solver Multi-thread/multi-core
FineSim	Synopsys	3-10x faster than SPICE Advanced full-matrix solvers Multi-thread / multi-core
HSIM	Synopsys	Hierarchical storage Isomorphic matching Parasitic reduction algorithm
Eldo Premier	Mentor Graphics	2.5-20x faster than SPICE Proprietary full-matrix solver Multi-thread / multi-core

2.3.2 Piecewise-Linear Simulation

In Fast SPICE, the high accuracy of analog circuit simulation is maintained, while various relaxation techniques are applied to simulate digital circuits. The piecewise-linear (PL) simulator is an interesting class of tools, in that it represents an entire design, including logic gates, resistors, capacitors and transistors, in the form of a piecewise linear system rather than the full blown differential algebraic system. In other words, it allows both analog and digital circuits to have a particular form of relaxation.

Bokhoven in [88] derived that a continuous time piecewise linear dynamical system can be fully captured by (5).

$$\begin{bmatrix} 0 \\ \dot{u} \\ p \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} x \\ u \\ q \end{bmatrix} + \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \quad (5)$$

$$where\ p \geq 0, q \geq 0\ and\ p^t \cdot q = 0$$

Vectors x and u represent the system's inputs and state variables respectively. The matrix shown can describe a PL system of 2^n linear segments, with n being the dimension of matrix A_{33} [33]. The condition on p and q originates from the derivation of the matrix representation, which is based on the addition of ideal diodes as PL elements to linearize the system and ensure PL behavior is exhibited [88]. In this case, p would be the diode voltage, and q the diode current. As a result, equation (5) is sometimes called the ideal diode model.

An advantageous property of piecewise-linear equations is that the solution algorithms yield global convergence, instead of local convergence which sometimes occurs for SPICE [33]. The most commonly used numerical algorithm for PL solutions is the Katzenelson algorithm [88]. This algorithm is similar to the Newton-Raphson

algorithm used in SPICE. The major difference is that the solution of the DAE system for SPICE is continuous, while here the system is described by a set of linear segments and discontinuities occur at the boundary of the segments. Therefore, at any iteration of the algorithm, if the solution vector was found to have crossed the boundary of the current segment, the next iteration would have to take a smaller time-step and the set of PL equations need to be solved for the new segment before further integration can occur. Stiphout et al. in [33] presents a piecewise-linear simulator, PLATO, combining the above PL representation of a system with multirate integration techniques described in Section 2.3.1. This implementation suffers from a great deal of computational overhead. Since each sub-circuits has its own optimal integration rate and simulation results are only available after the determination of a new stepsize, recomputation would be necessary if the new stepsizes differs too much from the previous ones because the design is flattened and simulated as a whole. Yang et al. in [34] improves simulation speed as well as capacity by performing symbolic and hierarchical piecewise-linear analysis. Instead of the ideal diode formulation, parameterized matrices model a given linear segment of the system behavior. Not only does this allow for more compact data representation, symbolic solution can also be obtained so that each iteration of the Katzenelson algorithm does not require a separate LU factorization, hence accelerating simulation speed. The hierarchical scheme allows very large (actually without limit) designs to be analyzed symbolically since the size of the circuit matrix that needs to be solved is reduced to the size of small sub-blocks [34]. The symbolic and hierarchical PL simulator proposed in [34] reports simulation speed ups of up to 10x compared to conventional SPICE.

An interesting property of the piecewise-linear simulator is that macromodels of sub-circuits such as operational amplifiers can be simulated instead of the full transistor-level model [33]. Just as digital circuits are simulated in their Boolean abstraction in digital simulators, here, the analog circuits can be simulated in a high-level piecewise-linear abstraction. As long as a sub-circuit’s behavior is described by a set of PL matrices,

this sub-circuit can be inserted directly into the overall design's matrices and be naturally mixed with low-level components [33]. The idea of abstracting away transistors inside an analog sub-block is actually what macromodeling – another area of research geared toward mixed-signal SoC validation – attempts to achieve. The macromodeling solution will be presented in Section 2.4.

2.3.3 Mixed-Mode/Co- Simulation

Fast SPICE simulators have come a long way from conventional SPICE. Because of their efficient algorithms and in some cases, specialized technologies that target a specific circuit topology (e.g. memory arrays), they have been adopted by industry to aid in circuit design. Yet, they are not often employed in SoC validation. Although Fast SPICE simulators have an improved time complexity of $O(n^{1.1\text{-}1.5})$ compared to $O(n^{1.5\text{-}2})$ of basic SPICE, they are still not scalable to designs with up to billions of active devices requiring millions of test vectors [1]. The mixed-mode simulator research community recognizes the difficulties of simulating both analog and digital circuits using an analog (or simplified and optimized analog) simulator, and proposes to stitch together an analog simulator with a digital simulator, thus allowing each type of circuit to be analyzed by their respective engines.

Some work (for example [12] and [24]) focus on implementing only the interface and making it flexible so that numerous simulators can be coupled together without much modification to the interface itself. Others choose to integrate specific analog solvers and digital simulators into a single entity. Whichever the route taken, the key elements at play in the design of a mixed-mode simulator are sub-circuit type labeling, electrical interface between analog and digital, and global timing synchronization. First, circuit sub-blocks must be designated as digital or analog. Manual partitioning of a design is often done [20] as it is natural in a top-down flow for the designer to be aware ahead of time which sub-blocks are analog and which are digital. In this case, the partitions are inherent in the representation of each block: schematic for analog, RTL for digital.

Once the sub-circuits are marked for either the analog solver or the digital engine, the interface between analog and digital circuits needs to be defined and signal value conversion rules need to be set. In its simplest form (as seen used in [16]), an analog to digital signal conversion is a threshold given by (6):

$$v_{logic} = \begin{cases} HIGH, v > k_r VDD(rising) \\ LOW, v < k_f VDD(falling) \end{cases} \quad (6)$$

where VDD is the supply voltage, and k_r and k_f are technology dependent constants with $0<k_f<k_r<1$. A more elaborate conversion method is used in [20] in which a four-level threshold (Max-H, Max-L, Min-H, Min-L) is implemented, such that Max-H and Min-H determine the margin for the detection of logic high and Max-L and Min-L determine the margin for logic low. If the simulator does not detect two logic levels (high and low) or a digital input is non-monotonic, an error flag is sent to the user or if the schematic representation of the digital block is available, the simulator automatically switches this block to the analog solver.

For digital to analog interface, the typical method is to use rise and fall times on the digital signal since analog solvers do not handle well instantaneous transitions in voltage [16]. In [20], the user defines delay parameters including rise/fall time and driving capability in capacitance units. The total rise/fall delay time would equal rise/fall time + capability factor * load capacitance. The high impedance state in digital logic could be modeled as a user defined high impedance output resistance, which is ramped exponentially to its final value so that instantaneous changes in voltage can be avoided [16].

The last major piece to a mixed-mode simulator is a global timing synchronization mechanism. Recall from Section 2.1, the time-steps in the analog solver is controlled by the required simulation accuracy. For the digital simulator, the evaluation points occur when inputs change and is recorded on a time wheel. The time instances of

the analog time-steps are independent from the time instances of the digital events, but they need to move forward together in a mixed-mode simulator. The scheme presented in [16] and [23] is to preserve the digital event list and between the events, let the analog solver evaluate using numerical methods but adapt the integration step such that it stops at a digital event. At the same time, when an analog signal reaches a certain condition that causes a digital event, that event is scheduled on the digital time wheel to force an evaluation in the digital simulator.

Now that the conceptual mixed-mode / co- simulator is complete, parallel computing can be applied to accelerate simulation just as it had benefited Fast SPICE simulators. A design is first partitioned onto different processors, either randomly or by applying some heuristics such as grouping by nearness [25]. Communicating across partitions is a large overhead. Value synchronization across multiple processors is typically achieved with sending messages containing information on the last simulated time [24]. A few scheduling policies include deadlock avoidance [26], deadlock detection and recovery [27], the Timewarp algorithm [30] and the moving time window algorithm [28]. The latter algorithms try to reduce communication by blocking value udpates in time. Speedup from parallelism is typically sub-linear due to the communication overhead across multiple processors [24].

Mixed-mode or co- simulators bring together analog and digital simulation with the aim to benefit from the best of two worlds: analog circuits will be solved accurately by the analog simulation engine, and at the same time, the digital simulation will be maintained at its usual blazing speed [29]. When pitted against top-level mixed-signal SoC validation, however, analog simulation becomes the bottleneck and slows down the entire simulation [32]. Even if a Fast SPICE simulator were used as the analog solver here, the speedup would be about an order of magnitude (recall from Section 2.3.1 and 2.3.2), but performing mixed-signal SoC validation requires the analog circuits be simulated over millions of test vectors. For example, a test might involve a power-up sequence that needs to wait for the analog circuits to turn on and reach a stable operation

point, or it might run through a complex digital calibration loop. These simulations could take ms or even longer, while the analog simulator's integration step might be on the order of ps or fs. Therefore, another line of attack is needed.

2.4 Macromodeling

Modifying the analog and digital simulators brought about some (but not enough) simulation acceleration. Time is predominantly being spent in the analog simulator. With this in mind, macromodeling aims to alleviate the analog bottleneck. Given a circuit netlist, the complete behavior of the system is captured by a set of differential-algebraic equations (1). Loosely speaking, the macromodeling problem (sometimes called model order reduction problem) is defined as finding an alternate, possibly simpler, version of (1) which will then simulate faster, but preferably retain as much of the full SPICE model behavior as possible. To evaluate the effectiveness of macromodeling for the mixed-signal validation problem, this section will review two macro model generation methods. Both methods assume a model candidate and attempts to determine the model "coefficients" through an assortment of approximation techniques. The Linear Time Invariant (LTI) macromodeling method, as the name suggests, assumes the circuit is an LTI system. The nonlinear methods assume various degrees and types of nonlinearity exist in the circuit to be modeled.

2.4.1 LTI Circuit Macromodeling

A representation of an LTI system is its transfer function H(s). Since the analog circuit is assumed to be LTI, a straightforward modeling method is to fit a reduced-order transfer function $H_r(s)$ to the original full transfer function. A well-known method is the Padè approximation [53] in which $H_r(s)$ takes the form of

$$H_r(s) = \frac{a_0 + a_1 s + a_2 s^2 + \cdots + a_m s^m}{b_0 + b_1 s + b_2 s^2 + \cdots + b_n s^n} \tag{7}$$

which satisfies

$$H_r(0) = H(0) \tag{8}$$
$$\frac{d^k}{ds^k} H_r(0) = \frac{d^k}{ds^k} H(0), \qquad k = 1,2,\ldots,m+n$$

where $\frac{d^k}{ds^k} H(0)$ is also known as the k-th moment of the transfer function. Hence this is also called the moment-matching method. Asymptotic Waveform Evaluation (AWE) and Padè via Lanczos (PVL) are two macromodeling algorithms based on this idea. AWE [58] explicitly computes the moments to perform the order reduction; however, the reduced model often has numerical inaccuracies which then need to be remediated using heuristic methods. PVL [59] improves on the numerical instability and provides an error bound on the accuracy of the reduced model.

So far, moment-matching algorithms yield reduced models in the frequency domain; however, recall from Section 2.1 that an analog simulator solves a set of DAE's and therefore cannot simulate a transfer function. The transfer function could be transformed into the matrix format that an analog solver accepts, but it is difficult to control the properties of the resulting matrix (for example, it might be non-passive which then leads to instability [47]). Linear projection based algorithms start with the DAE representation of the circuit, and projects that system onto a smaller linear subspace that still roughly encloses the circuit's trajectories. The intuition is that by projecting the system onto such a subspace, the model order is reduced and at the same time, major dynamics of the original circuit is preserved since most of the system's state is reachable in the smaller subspace [39]. For LTI circuits, the DAEs have a specific form due to linear properties:

$$C\frac{d}{dt}x + Gx + Bu = 0 \qquad (9)$$
$$y = D^T x$$

The vector x represents the state variables, u the list of inputs and y the list of outputs. The reduced-order model is in the same form with new and smaller C, G, B, and D matrices that are defined by

$$C_r = W^T CV, \qquad G_r = W^T GV, \qquad B_r = W^T B, \qquad D_r = V^T D. \qquad (10)$$

V and W are projection matrices that define the smaller subspace onto which the original system is projected [39]. Since these models look just like any other matrix that an analog simulator is intended to solve, they can be directly inserted into SPICE/fast SPICE or a mixed-mode simulator. The selection of the V and W can be made to guarantee certain desirable linear circuit properties of the resulting macromodel, such as passivity [47], balanced controllability and observability [43], stability[44], etc.

The speed up achieved by simulating the reduced order macromodel instead of the full circuit DAE is directly correlated with the amount of order reduction performed. The smaller the resulting macromodel, the faster it will simulate. Wang et. al in [35] demonstrats LTI macromodeling of a two-stage op-amp using projection-based methods. The model order is reduced from 51 to 16, and a 60x improvement in runtime is observed using HSPICE.

2.4.2 Nonlinear Circuit Macromodeling

Nonlinear macromodeling has the same goal as LTI macromodeling, which is finding a lower order matrix representation of an analog circuit; however, it has the additional aim of capturing some nonlinear behavior of the original DAE's as well. Macromodeling algorithms for nonlinear systems are not as mature as those of LTI

systems [39]. In most cases, a particular modeling technique and model candidate are needed for each type of nonlinear behavior to be preserved in the reduced model. For example, if the behavior is linear time-varying (LTV), the Padè-like moment matching algorithm presented in [36] can be applied to preserve moments of such a system. The model candidate is a set of multirate partial differential equations. For a nonlinear behavior described by a Volterra series approximation, the NORM [41] algorithm retains the moments of the Volterra kernels by using polynomial differential equations as the model candidate. More nonlinear macromodeling algorithms are listed in Table 2.

Table 2 – Nonlinear macromodeling algorithms

Method	**Model Candidate**	**Behavior Retained**	**Application**
QLMOR [48]	quadratic-linear differential equations	moments of Volterra kernels	weakly nonlinear circuits
ISF [37]	time-varying phase sensitivity	oscillator phase sensitivity	oscillators
PPV [38]	scalar differential equation	oscillator phase sensitivity	oscillators
POD [39]	linear differential equations	deviation from training waveforms	nonlinear circuits
TPWL [48]	piecewise-linear differential equations	moments of transfer function of each linearized segment	nonlinear circuits
ManiMor [45]	piecewise-linear differential equations	DC and AC response	nonlinear circuits

The ad hoc nature of the above collection of algorithms is a good indication that nonlinear macromodeling is a difficult problem. In fact, many of these algorithms do not achieve much order reduction in their macromodels. For example, the POD method,

which is based on the idea of projecting the nonlinear state space onto a linear sub-space, may return a macromodel that's as large as the full model [39]. The ManiMor algorithm fairs a little better. Gu et. al [45] [46] proposes that the trajectories of a nonlinear circuit are restricted not to a linear subspace, but more likely to a nonlinear manifold. Intuitively speaking, since a manifold presents a more "snug" fit to the space that contains the circuit's trajectories, it is usually lower-dimensional compared to a linear sub-space. In [45], a 4-stage CML buffer with inductive peaking is shown to be order-reduced from 52 to 5 by the ManiMor algorithm.

LTI macromodeling provided nearly two orders of magnitude of analog runtime reduction; however, these order-reduced models are still not commonly used in high-level validation of mixed-signal SoC's with large digital circuits [39]. Nonlinear macromodeling, though more accurate in the sense that some nonlinearity of the original model is captured in the reduced model, is even less likely to provide the necessary simulation acceleration. Behavioral modeling is a form of macromodeling that relaxes the stringent requirement of accuracy and the next section will focus on this approach to mixed-signal validation.

2.5 Behavioral Modeling

The LTI and nonlinear macromodeling methods seek to ensure that the reduced-order model faithfully captures circuit characteristics in the transistor-level description [39]. Full-chip functional validation, however, has more moderate expectation on accuracy; designers were willing to pay a price for the ability to simulate a much larger system with greater speed [1]. Behavioral modeling is a method of fulfilling this need. Behavioral models are hand-crafted (sometimes automatically generated) models written in a high-level language that require expert knowledge of the analog circuit being modeled [39]. Because the models are not derived directly from low-level transistor models, there will be some loss in accuracy [52]. The attractiveness of behavioral

modeling, however, comes from the potential for 100-1000x simulation speedup [32] and at such speeds, a more complete validation of mixed-signal SoC seems more feasible. In addition, it is possible to include checking or assertions in the behavioral models to aid the validation effort. Many high-level modeling languages exist and the following discussion will be divided according to the language used.

2.5.1 Simulink/Matlab

Simulink/Matlab has been a popular tool for system-level functional simulation and verification [64]. The Simulink libraries have wide-ranging fundamental building blocks including continuous time integration, derivative, state space and transfer function blocks, discrete time quantizers, filters and delay elements, mathematical operators, signal routing blocks and data visualization scopes. The system-level designers only have to select the correct blocks from the libraries, connect them and configure the simulation settings [64]. If only discrete time components exist in the model, the solver is discrete; for continuous time states, several DAE solvers are available and the user may choose between fixed step size or variable step size numerical integration methods – the variable step size integration method is a means of potential simulation acceleration. In fact, the MATLAB toolbox has been widely used (for bang-bang PLL's [65], clock/data recovery circuits [66], DC-DC regulator's [67], etc.) and is the de facto tool for discrete-time delta-sigma modulator modeling [64]. A disadvantage of Simulink/Matlab is that these models must run in Matlab's simulator and cannot be easily connected to the digital system that needs to be validated in an SoC.

2.5.2 SystemC-AMS

SystemC-AMS is a more recently available modeling language that has garnered some attention. This is a C++ based language that extends SystemC (a digital circuits modeling language) and provides three Modes of Computation (MoCs) – Timed Data Flow model (TDF), Linear Signal Flow model (LSF) and Electrical Linear Networks

model (ELN) [69]. TDF is a discrete-time modeling style which treats data as signals sampled in time while carrying discrete or continuous values [70]. Different blocks are allowed to have different time steps [69]. The LSF MoC enables the modeling of continuous time systems as a set of linear algebraic equations [70] – this is similar to Simulink's continuous model elements. In the ELN model, electrical primitives can be instantiated along with the rest of the design. Available primitives include [70] current/voltage sources, resistors, capacitors, inductors, transmission lines, transformers, ideal switches, etc. In literature, models of PLL's [69] have been demonstrated. Heterogeneous systems (e.g. CMOS video sensor [71], wireless sensor networks [72]) have also found SystemC-AMS to be a useful modeling language. Unfortunately, again, SystemC-AMS has its own simulator and therefore is difficult to connect to a digital system written in Verilog.

2.5.3 Verilog-A/Verilog-AMS/VHDL-AMS

These languages, like MATLAB/Simulink and SystemC-AMS, are capable of handling continuous as well as discrete systems; but they have the added advantage of easier integration into circuit design tools such as Cadence, which has mixed-mode simulation capabilities [64], making the connection to a digital system relatively easy. Verilog-A is an extension of Verilog-HDL and introduces analog functions such as integration, differentiation, delay, transition, Laplace transform and Z transform [74]; and its solver for continuous systems is similar to DAE solvers found in SPICE simulators. Verilog-AMS and VHDL-AMS are the union of Verilog/VHDL and Verilog-A, and therefore inherit the ability to handle both digital and continuous time analog signals [73]. Models of diodes [75], gain amplifiers [74], PLL's [73], delta-sigma modulators [76], etc. have been prototyped.

Labeled Hybrid Petri Nets (LHPN) [60] [62] is an interesting algorithm that automatically generates finite state machine (FSM) based models in Verilog-AMS. It relies on the concept of system identification [50] [51]. The general idea is to view the

circuit to be modeled as a black box, measure the circuit's output waveforms corresponding to a set of input training waveforms, and fit a parameterized model that relates the input and output.

All the languages here have a SPICE-like analog solver. Even though it is most likely solving a much smaller problem than a full circuit DAE, once the solver is invoked, it must execute all the steps described in Section 2.1 in order to solve differential equations. Therefore, using the continuous time analysis capabilities of these languages will result in longer simulation times compared to pure digital simulation. For example, David in [68] presents a PLL Verilog-AMS model that simulates approximately 125x faster than its transistor-level representation.

2.5.4 Digital Verilog with Real

Lastly, Verilog simulators with "real" value interconnect wires and basic algebraic capability have existed for several years [80] and modeling using digital languages (VHDL/Verilog/SystemVerilog) with these extended functionality has been demonstrated. Digital Verilog is inherently discrete-time, hence sampled data systems have a natural fit into the simulation environment [77]. The format of sampled data is illustrated in Figure 2. The signal updates periodically and at each update, the signal has a value and that value is held constant until the next update.

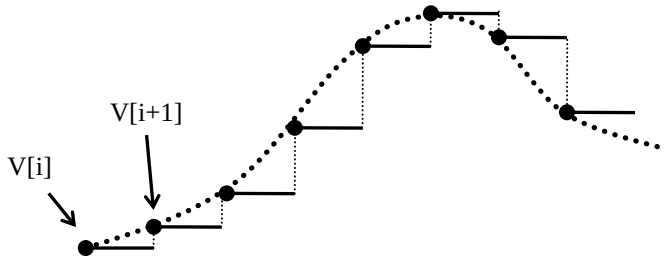

Figure 2 – Sampled data representation of continuous time signal

Some analog systems such as delta sigma modulators (DSM) and switched capacitor (SC) filters can be modeled with sampled data without much difficulty. Take the first order DSM in Figure 3 as an example. The output of the modulator will be sampled directly by a digital system and when properly designed, all nodes will settle when the clock edge arrives. In other words the system is synchronous. The same can be said of SC filters. Figure 4 shows the corresponding signal flow graph for the first order DSM assuming $C_F = C$.

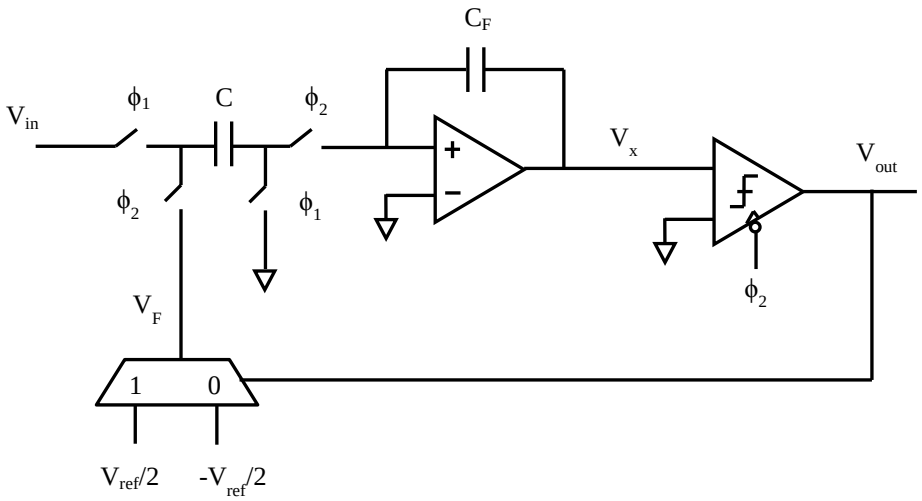

Figure 3 – Switched capacitor implementation of 1st order DSM

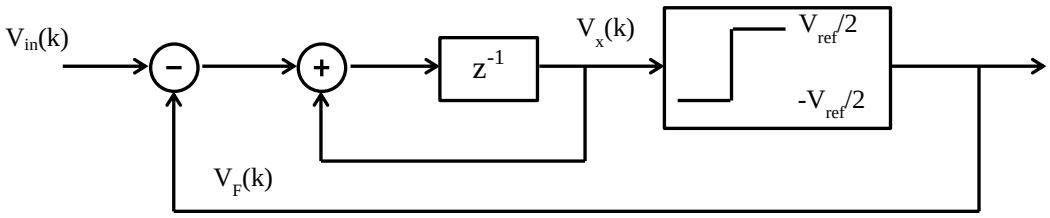

Figure 4 – Signal flow graph for 1st order DSM

A difference equation that describes the system can be extracted from the graph as below:

$$\mathrm{V_x}(k+1) = V_{in}(k) - V_F(k) + V_x(k) \tag{11}$$

$$where\ V_F(k) = \begin{cases} \frac{V_{ref}}{2}, V_x(k-1) > 0 \\ -\frac{V_{ref}}{2}, V_x(k-1) < 0 \end{cases}$$

Using SystemVerilog as an example, the pseudo-code for the first order DSM is shown in Listing 1.

```
module DSM (
  input clk,
  input real Vin,
  output real Vout
);

parameter real Vref = 0.5;
real Vx, VF;

always @ (negedge clk) begin
  // fill in difference equation here
  Vx = Vin – VF + Vx;
  VF = sign(Vx)*(Vref/2);
  Vout = sign(Vx);
end

endmodule
```

Listing 1 – Example model of first order delta-sigma modulator

Recall from Section 2.5 that MATLAB has been used extensively to model delta sigma modulators and switched capacitor filters. Here, digital Verilog languages with real number extensions provide similar capabilities. For other continuous time analog systems, various transformation techniques to approximate them as discrete-time systems can be found in literature. The most common is the bilinear transform [77], which converts analog transfer functions to discrete-time difference equations using a sufficiently high sampling rate (by the Nyquist criterion). The use of the forward Euler integration

technique to perform the conversion has also been demonstrated in [80]. A variety of circuits have been modeled using the above concepts; these circuits include successive approximation ADC's [77], auto-gain control circuit (AGC) [77], switched capacitor amplifiers [80], phase-interpolators [52], oscillators [78] and all-digital PLL's [79].

2.6 Summary

Validation of a mixed-signal SoC requires the concurrent simulation of both analog and digital in order to verify the complex interaction between the two [76]. Unfortunately, basic analog and digital simulators are inadequate for this job. The digital simulator cannot solve DAE's and the analog simulator is too slow to compute waveforms for the digital back-end, which nowadays is likely to contain several million gates [78]. This chapter examined several research areas that address the issue of mixed-signal validation.

Fast SPICE experiments with the ideas of cleverer matrix manipulation and computation to speed up analog simulation, as well as efficient digital structure recognition and storage to allow for relaxation on digital simulation accuracy and hence improved digital simulation speed in an analog environment. Currently, however, they often cannot provide adequate simulation speed for large mixed-signal SoC [80]. Macromodeling and piecewise-linear simulators try to reduce the complexity of the nonlinear system that analog solvers need to solve and hence accelerate simulation. Mixed-mode simulation leverages the efficiency of specialization, and advocates for the preservation of accuracy for analog as well as speed for digital. It accomplishes its goals by bringing together an analog simulator and a digital engine through appropriate continuous/discrete conversion interfaces and timing synchronization mechanisms.

Behavioral modeling seems to have the most potential in terms of simulation time reduction (up to three orders of magnitude). Many modeling languages and model

examples have been reviewed; some are faster, others are more easily connected to digital designs. Overall, the models are varied. It is not uncommon for engineers to have their favorite language and write what was needed at the moment [80]. This ad hoc nature causes compatibility issues and is a detriment to the efficiency of the validation process. What's presented in the rest of this thesis is a set of guidelines that regularizes the process of creating behavioral models in an event-driven environment as well as an investigation of the applicability of these guidelines to mixed-signal SoC validation.

Chapter 3

Behavioral Modeling Approach

Based on the information provided in Chapter 2, behavioral modeling is currently the only approach that might be fast enough for mixed-signal SoC validation. In order for behavioral modeling to work well, the models must satisfy several criteria. When behavioral models are inserted into the top-level design for simulation, correlation between system success/failure and sub-circuit or interface correctness/error is dependent upon the equivalence of the behavioral models and their circuit implementation. Example equivalence checking methods include transfer matrix matching under linear system assumption [81], simulation trace matching [61], and finite-state-machine-based macromodel generation followed by state space exploration [83]. Regardless of the checking method used, to have the models verified at all stipulates that the pins of the model must match those of the schematics. This is the first criterion. Second, as discussed in Chapter 2, writing behavior models to raise abstraction level is a commonly-accepted solution to increase the simulation speed by hiding low-level details of the architecture [72]. Therefore, the models should simulate relatively quickly. Lastly, the mission of functional validation prior to tape-out is not only checking correctness of wire connections and signal routing, but also making a rough performance analysis "over night" [76]. Hence, capturing important circuit dynamics in the models is highly desirable.

With these goals in mind, this chapter will describe the necessary elements to writing behavioral models for SoC validation. First, a modeling language needs to be chosen and the reasons for choosing SystemVerilog as an example in this thesis will be discussed in Section 3.1. Then, in order to fit into a digital simulation environment that's unidirectional, proper circuit partitioning is necessary and will be the topic of Section 3.2. Next, even though SystemVerilog is supplemented with real number capabilities, it is still a discrete-time simulator, and therefore an appropriate method of representing analog signals is presented in Section 3.3. Lastly, given the proposed signal representation, the models must compute their outputs in an efficient fashion and an approach will be described in Section 3.4. Throughout all these steps, designer's knowledge is indispensable and will be relied upon to aid the modeling of analog circuits.

3.1 Modeling Language

Among the available modeling languages described in Section 2.5, SystemVerilog is used to write all the specific examples that follow; however, the methods presented in this thesis can be applied to any event-driven simulator with real number capabilities. The choice of this particular modeling language is rather arbitrary, but SystemVerilog is not without its many advantages. The digital languages (VHDL/Verilog/SystemVerilog) extended with real numbers have the potential for at least an order of magnitude increase in speed compared to Verilog-A/Verilog-AMS/VHDL-AMS [52]. Simulink and SystemC-AMS can achieve similar simulation speed at the system level, but models written in these languages have little resemblance to the actual physical implementation [9] and are very much disconnected with circuit design tools. SystemVerilog, on the other hand, is currently the go-to language for digital validation among all the digital modeling languages, and therefore eliminates the need to recreate the digital portion of the SoC for a different platform. Due to the above benefits, as well as in anticipation of the continued increase in analog/digital interaction and the size of mixed-signal SoC (and the

commensurate need for scalability), SystemVerilog is chosen as the representative modeling language.

3.2 Circuit Partitioning

Since the behavioral models need to run in a digital simulator - SystemVerilog to be specific – the first challenge to tackle is related to the unidirectional modeling framework of the digital environment. In general, digital gates are unidirectional [16] – the output of one gate drives the input of the next gate and the output of the driven gate doesn't affect the output of the driving gate. The digital simulator is designed to deal with unidirectional signal propagation and uses the fact that a gate's input changes cause it to re-evaluate that gate.

With analog circuits, however, unidirectionality is not guaranteed. A good example is current summing nodes. In a current DAC (Figure 5), for example, the output of the current sources is current, while the feedback resistor drives a voltage back to the input node of the buffer. This causes conflict. In another example, the frontend of a single-slope ADC (Figure 6) could be intuitively split into a sampler with a sampling cap, a switchable current source and a comparator. Equivalence checking will work under this particular way of partitioning, but the input of the comparator would have two drivers – the sampler's voltage output and the current source's current output. The current source injects charge into the capacitor, which then sets the voltage. Like the feedback resistor in the DAC example, this current source creates a non-unidirectional node at the comparator input.

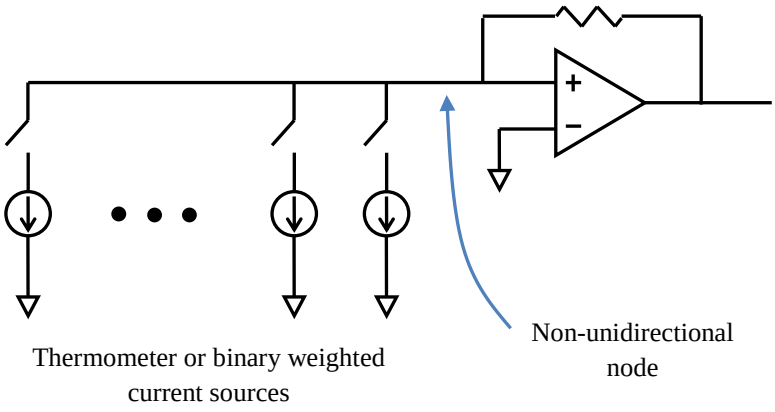

Figure 5 – Current summing node in DAC

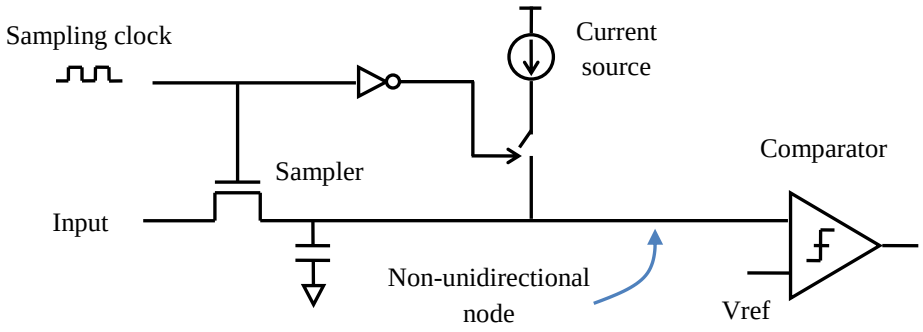

Figure 6 – Current summing node in single-slope ADC frontend

In both examples, a current summing node cannot be on the boundary of a stand-alone module; instead, circuit blocks must be combined so that these nodes are absorbed into the interior of a module. Equivalence checking would then be performed on these combined modules. For the current DAC, all the current sources need to be combined into a cell with a single current output that drives the buffer. Furthermore, by design, the input of the current buffer is a good virtual ground, hence the voltage at this node doesn't move much. With this design intuition, the buffer with the feedback resistor must be modeled as one block with a current input and voltage output. In the single-slope ADC frontend, the sampler and current source must be combined to form a block with a single voltage output, which then drives the comparator. It should be noted that the schematics

can retain hierarchy, but it is important to have a wrapper for the combined block that can be checked against the functional model. The implication for validation using a digital simulator then is that hierarchies that contain non-unidirectional nodes should be carefully planned as early as the schematic design phase.

A procedure to determine the appropriate boundary of analog wrapper modules is illustrated in Figure 7. To arrive at a model that has closer resemblance to the physical implementation, start with existing blocks in the schematic. Since these blocks most likely made intuitive sense to the analog designer, the models will be able to leverage the mathematical tools that were used, in the first place, to design these blocks to better capture the circuit dynamics. Non-unidirectional nodes usually occur when multiple current sources are summed or both current and voltage signals co-exist at a single point. A circuit designer would know best where these nodes are located in a schematic. If non-unidirectionality exists, combine as few of the blocks as possible until all current summing nodes are within a module. Smaller blocks are preferred since they are more likely to have well established theories that encapsulate their analog behaviors and hence are more beneficial towards the modeling of circuit dynamics.

It might be possible to use an automatic program to execute the partitioning, as long as the circuit designers annotate “current” or “voltage” for all ports in a design. The automation program can then construct a graph of all the connections, detect nodes with multiple current drivers or current/voltage contention, and hide those nodes until there aren’t any left. Of course, the analog designer might want to prune the final result according to a design’s specific characteristics. For example, the computerized program might conclude that the entire DAC in Figure 5 is a single module; however, if the designer is aware that the virtual ground is a good approximation, then he/she might decide to have the current sources as one module, and the current buffer as a separate module.

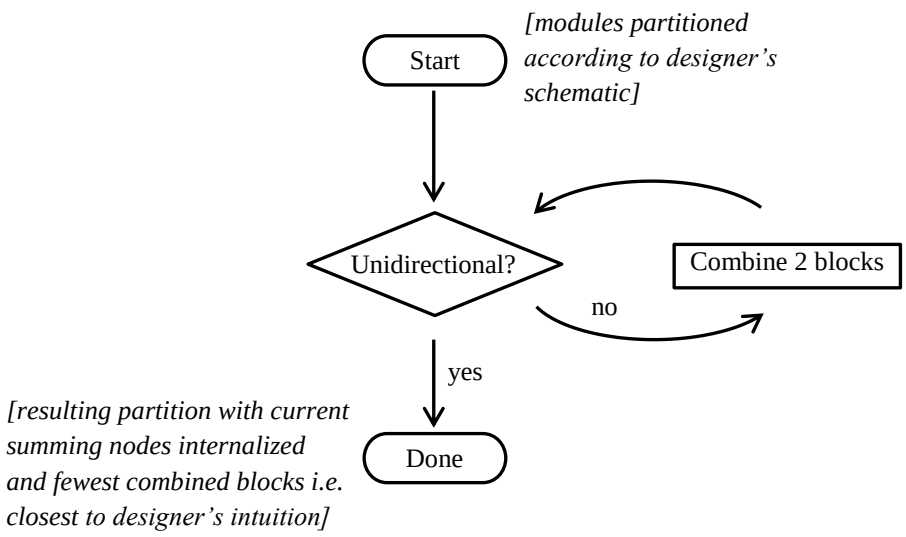

Figure 7 – Circuit partitioning procedure

3.3 Signal Representation

After the analog circuits are properly segmented into unidirectional blocks, the next challenge to overcome is the fact that digital simulators work with Boolean values and discrete events, while analog signals are continuous-time and continuous-value. Even though simulators like SystemVerilog have real number extensions and is considered to be continuous-value, the digital environment is still discrete-time. This section will explain the issues with the sampled data representation of analog signals, propose a piecewise linear (PWL) representation and demonstrate that this augmented representation is capable of interaction with digital signals as easily as the sampled data representation.

3.3.1 Sampled Data Representation

When it comes to approximating a continuous time signal as discrete time, the most obvious method is to use a Nyquist sampled representation. Recall that this method has been described in Section 2.5 as part of the background on behavioral modeling.

Details of writing models for a delta-sigma modulator and switched capacitor filter were discussed as well. The ports of these models are real-valued and any digital simulator with real number extensions can handle this with ease. These circuits are also clocked and their output is synchronous. Since the evolution of the states in the system marches to the edges of one or more clock signals, the sampled data representation fits in naturally with signals that are updated periodically according to these clocks and held constant throughout the period until the next update.

Although this approach has proven to work very well for "synchronous" analog systems, many analog circuits do not fall into this category. A classic example is a clock-less comparator. As Figure 8 shows, a comparator toggles its output from low to high as its positive input increases in value beyond its negative input. In the continuous time world, the crossing point is indicated in the figure. When the input is represented as sampled data, the crossing point will be shifted in time. This error becomes smaller as the sample rate increases; however this means that an analog signal requires many finely spaced samples leading to prohibitive simulation time. The fundamental issue here is that the digital simulator cannot interpolate between two signal updates until the second sample arrives.

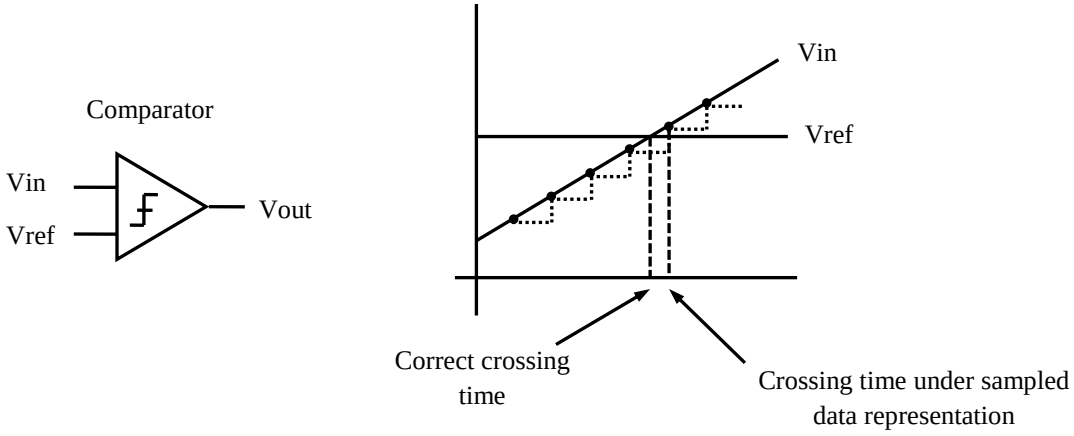

Figure 8 – Comparator behavior using sampled data representation

A similar deficiency of the sampled data representation can be seen in a track-and-hold circuit (see Figure 9). For simplicity, assume that the input is a sinusoid with the same frequency (100MHz in the illustration) as the sampling clock so that the same point on the input signal is sampled repeatedly. Under ideal conditions, the sampling clock has no jitter. The resulting sampled values are identically -0.5878 as shown in Figure 10.

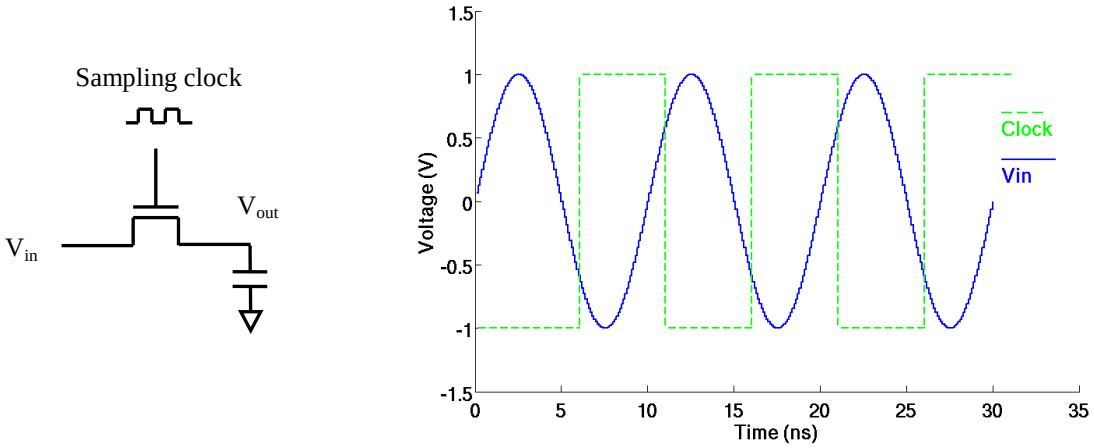

Figure 9 – Track-and-hold using sampled data representation

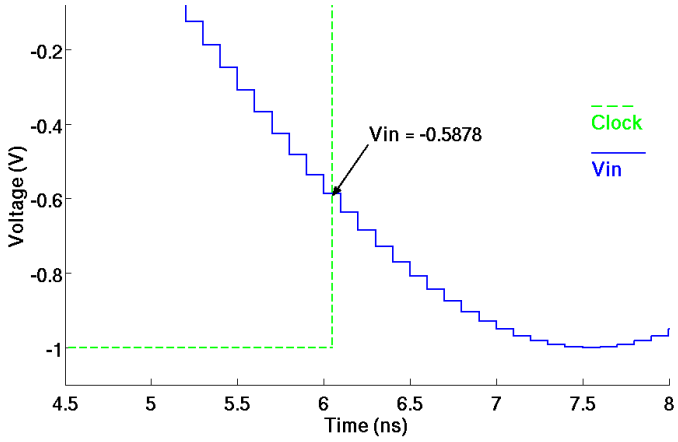

Figure 10 – Sampled data representation with jitter-less clock

Now consider superimposing an rms jitter of 50ps on the sampling clock, while the sampled data representation is still used for the input. The sampled values are shown in Figure 11. In this case, the error on the sampled values compared to the case without jitter is either zero or equal to the difference between two consecutive input signal updates (or sometimes, when the clock happens to deviate quite far, the difference is between the ideal value and two signal updates away). What occurs in reality, however, should be an error distribution that's close to a Gaussian.

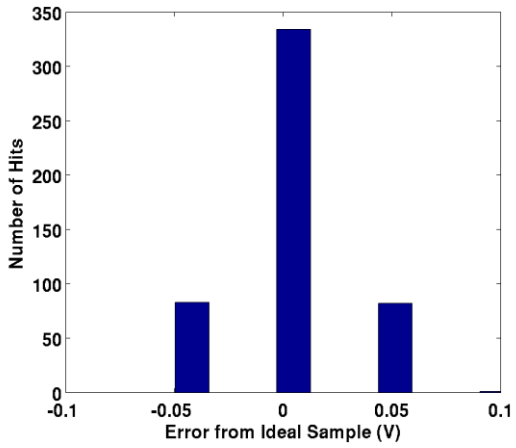

Figure 11 – Samples captured by jittery clock under sampled data representation

3.3.2 Augmented Representation

When dealing with asynchronous analog circuits, the sampled data representation of analog signals falls victim to a lack of information on the signal shape between sample updates. A solution is to augment the samples to contain more information. The concept of samples or signal updates will still apply since the digital simulator is discrete-time. One form of data supplement will be considered here: piecewise linear (PWL) signal representation. In SystemVerilog, pin-accuracy can be retained by defining structures that contain more than one element and passing these structures across module ports. Listing 2 shows a structure containing a starting value v1 of the signal and a slope, as well as how it can be defined as the type of a module's input/output port.

```
typedef struct {
  real v1;
  real slope;
} pwl_struct;

module TH (
  input pwl_struct Vin,
  input clk,
  output pwl_struct Vout
);

. . .

endmodule
```

Listing 2 – SystemVerilog structure for piecewise linear representation

With this structure, analog signals can be described in a piecewise linear manner, in which each linear segment starts at the value and continues on the slope. The time instance of each signal sample or update is indicated by a change in either element of the structure, and therefore is not restricted to any particular period like the sampled data representation.

The PWL representation, like any approximation method, will cause errors. It is interesting to study the errors introduced into the simulation result of a design when its continuous time input is represented in piecewise linear format. To that end, a pure sinusoid of a particular frequency f_{sig} is employed here for demonstration purposes. A fixed update rate f_{update} for the piecewise linear representation is assumed for convenience. In effect, the frontend of the design consumes and processes a sampled signal reconstructed with an ideal first-order hold; in the frequency domain, the input sinusoid would look like Figure 12 with copies of the sinusoid spectrum shifted by integer multiples of the signal update rate f_{update} and their magnitudes modified by a $sinc^2$ function.

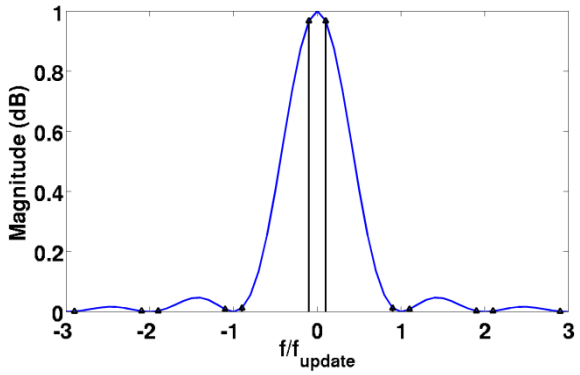

Figure 12 – Spectrum of piecewise linear representation of a sinusoid

If the attenuated spectrum copies are considered to be unwanted signal content or "errors," then a metric for measuring the amount of error introduced by the PWL representations could be formulated as the difference in power between the signal tone and the largest spur. Figure 13 plots this difference for various ratios of signal update rate to sinusoid frequency. For any band-limited system, the circuits that process this sinusoid will further attenuate the spurs. The amount of suppression depends on the system's bandwidth and the chosen signal update rate.

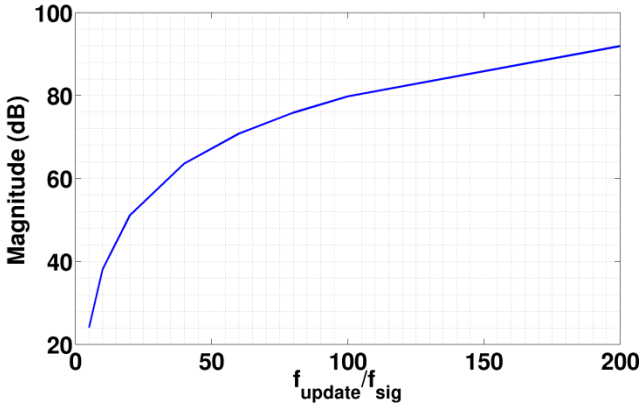

Figure 13 – Signal to error ratio vs. signal update rates for constant time internal PWL waveform

This analysis indicates that the PWL representation results in small spurs in the continuous time domain, while the sampled data representation does not introduce any distortion to synchronous systems in the discrete time domain. However, although the signal update rate is fixed in the above discussion for convenience, the piecewise linear representation does not stipulate such a fixed rate. A signal may exhibit different rates of change throughout the simulated time frame; the update rate may be faster when the signal slope changes quickly, or it may be slower when the signal barely moves. This freedom cannot be easily afforded by models leveraging the sampled data representation and bilinear transforms, since such transforms require a known and fixed sample rate.

The key value of the PWL representation, however, lies in its ability to allow digital simulators to generate asynchronous events, thereby solving the issue with the sampled data representation that was identified in the preceding section. To illustrate the enabling abilities of the PWL representation in general terms, Figure 14 depicts all the circuits and interfaces that can now be modeled (the sampled data representation is suitable for modeling only the right half of the figure).

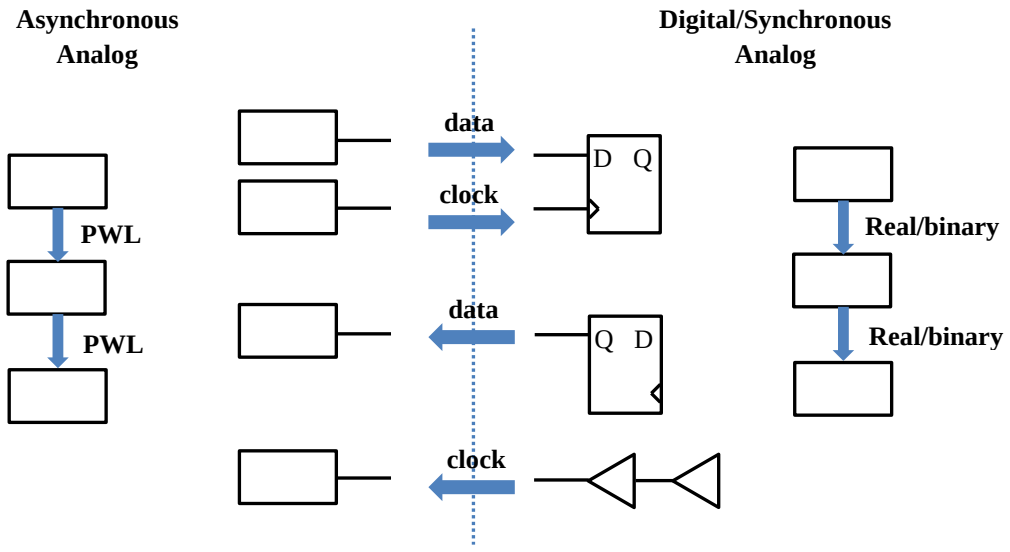

Figure 14 – Block to block interactions in mixed-signal design

First consider a PWL signal crossing from the analog to the digital domain. The signal could cross as a data signal or a clock signal. If it crosses as a data signal, then there will usually be a digital clock that samples it and the sampled value can be interpolated at the clock edge according to the slope of the signal. This case is similar to the track-and-hold example. Using the same sampling clock with 50ps rms jitter and the same input signal represented in PWL format, the distribution of the sampled values is portrayed in Figure 15. Compared to Figure 11, this distribution more closely resembles a realistic profile.

If the signal is a clock signal that's generated in the analog domain and supplied to the digital domain, then the timing of the clock edges are critical for the digital circuits. In this case, any delay or skew of the clock signal will need to be accounted for in the analog block that generated the signal. This case was illustrated by the continuous time comparator example in Section 3.3.1. The asynchronous crossing of the comparator can now be resolved since the slope information allows the simulator to interpolate.

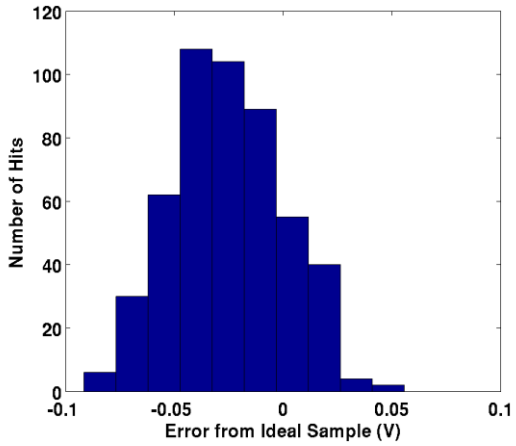

Figure 15 – Samples captured by jittery clock under piecewise linear representation

Similarly, the signal flow from synchronous to asynchronous circuits can also be modeled. The treatment is the same whether it's a data or clock signal. The format of the signal needs to be first converted to the PWL structure, in which the slope element would be identically 0 in this case. If the rise or fall time is important for modeling the receiving analog module's behavior, then slopes and delays can be attached to the staircase signal to make them trapezoidal signals. This can be done as a frontend to the analog module or as a separate module since digital signals typically have a few buffers or inverters to increase their drive strength for the downstream analog blocks.

Lastly, the PWL signal representation allows a chain of asynchronous circuits to be modeled. In order for a chain to exist, the individual analog models need to consume a PWL signal from the previous block and produce a PWL signal for the next block. The details of the required computation will be the topic of the following section.

In summary, the commonly used sampled data representation of analog signals is unsuitable for modeling asynchronous analog circuits. A piecewise linear signal representation (or some form of data supplementation) solves this issue. Providing the value and slope of the signal during each update allows digital simulators to generate asynchronous events for asynchronous circuits and therefore sets up the stage to better capture circuit dynamics – a goal of modeling writing established at the beginning of the chapter. In addition, the user defined structure available in SystemVerilog lets multiple pieces of data to be passed through a port as single entity and therefore pin-accuracy of the model can be preserved under PWL signal representation.

3.3.3 Another Augmented Representation: XMODEL

The XMODEL [55] is a modeling approach developed contemporaneously and independently from this work. It seeks to write behavioral models that describe analog functions as nearly linear filters in the s-domain (possibly after some domain transformation [54] to place the circuit in a mostly linear domain) and consequently uses

the s-domain representation of analog signals. Data augmentation is achieved via the equivalence between a time domain signal and its Laplace transform.

Jang et al. show in [55] that waveforms of source elements available in SPICE can be captured in the form of x(t) with their equivalent Laplace domain expressions X(s) as shown below:

$$x(t) = \sum_i c_i t^{m_i} e^{-a_i t} \quad \rightarrow \quad X(s) = \sum_i \frac{b_i}{(s + a_i)^{m_i+1}} \tag{12}$$

The set of coefficients $\{a_i, b_i, m_i\}$ uniquely identifies an analog signal. The output of a model can then be computed by multiplying the s-domain representation of the input with the s-domain transfer function of the system. The resulting expression will have the same form as (12) after partial-fraction decomposition. The inputs and outputs of all models are therefore a set of real numbers corresponding to the coefficients of the signals' Laplace-domain representation. This set of coefficients changes in size and value at discrete instances to reflect the signal change in time domain. Consequently, unlike the PWL representation, XMODEL does not require steps in time; however, reconstruction is necessary to view the waveforms in time domain. A high-speed link including a transmitter, channel, a continuous time equalizer (CTLE), and a decision feedback equalizing receiver has been demonstrated in [55], and a switching boost converter in [56]. Among these models, the CTLE achieved ~990x speed-up compared to HSPICE. The two different representations each have their own advantages, and which is better depends heavily on the type and complexity of waveforms and systems modeled.

3.4 Module Output Computation

As shown in Figure 14 of Section 3.3.2, the input and output of analog circuits are now a multi-element, real-valued structure – no longer the Boolean used in typical digital designs. Hence, a way of computing a PWL output from PWL input is fittingly the next challenge. The linear abstraction of analog circuits and the concept of domain translation mentioned in the behavioral modeling section of the background will be leveraged in the process. Recall that analog result surfaces are smooth in some domain; it could be the voltage or current domain, or some transformed domain such as phase and time. This smoothness is what makes analog validation practical.

A continuous time amplifier, for example, is smooth in voltage. This circuit is biased at an operating point on its smooth result surface and the vicinity of the bias point can be captured by a linear system (possibly with small non-linearity). The amplifier might be made to move to a different operating point through the adjustments of, for example, the bias current. Though the system that describes the behavior of the new operating point will be different, it nonetheless remains linear. In another example, a ring-based voltage controlled oscillator's output is roughly a square wave in the voltage domain and therefore is not smooth. When viewed in the phase domain, however, the VCO accumulates phase by integrating frequency over time. Integration is a linear operation and the VCO frequency is a (piecewise) linear function of the control voltage. Therefore, if phase was taken as a VCO's output, the system would be smooth. A circuit designer's knowledge must be called upon to determine a domain of linearity for a particular circuit and the model's output is computed in this domain (followed by a possible translation to the voltage domain for the downstream modules if necessary).

Based on this linear assumption, the computation of the PWL output of a module due to a PWL input can be accomplished in three steps. First compute the continuous time domain response of a module due to a single linear segment on its input. Then based on the time constants of the system, a piecewise linear approximation is formed. Repeat

with the next linear segment that arrives on the input. Lastly, in order to be efficient, output updates that are within a certain error tolerance are removed. The following sections will describe the process of computing a module's output in more detail.

3.4.1 Time Domain Response

A piecewise linear input can be viewed as a series of delayed inputs each with a different initial value and slope. The total transient response is a trajectory traced out by the evolution of the system's states when stimulated with successive delayed inputs. Since the final states at the conclusion of the current input segment are the initial states for the next input segment, and each input segment is in the same form (i.e. value and slope), it is sufficient to examine the behavior of the system due to a single linear segment and repeat the computation for the sequence of linear segments as they get updated.

Under the assumption of linear intent, the superposition property allows a piecewise linear segment to be decomposed into a step with magnitude equaling to the initial value element of the PWL structure, and a ramp that starts at 0 but increases at the rate indicated by the slope element of the PWL structure. The total response of the system, then, is composed of the response to the step, the response to the ramp and the decay of the initial states of the system. Figure 16 illustrates this decomposition.

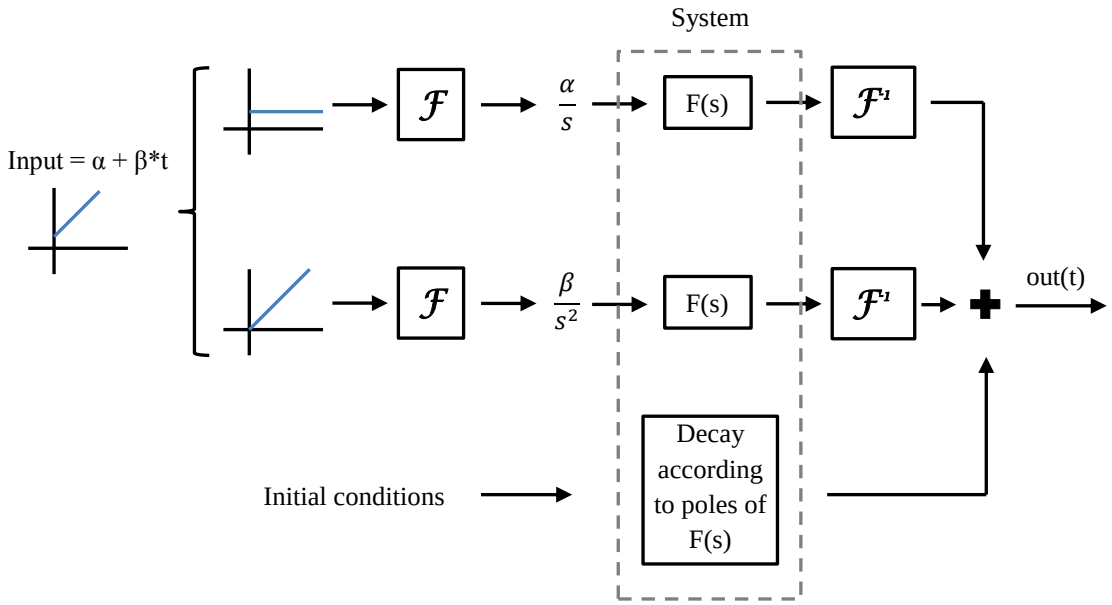

Figure 16 – Time domain response to input linear segment

The input linear segment with α being the initial value and β being the slope is divided into a step of magnitude α and a ramp of rate β. These two parts correspond to $\frac{\alpha}{s}$ and $\frac{\beta}{s^2}$ respectively in the Laplace domain. For any linear module with a transfer function F(s), the time domain waveform of the driven response (due to the new step value and the new ramp) can be calculated using the inverse Laplace transform method. This is shown as the top two paths in the figure. The decay of the initial states is subject to the dynamics imposed by the poles of the system in question and this translates to exponentials in the time domain. Adding this third path completes the picture for the time domain response.

To be more concrete, take the example of a single-pole system in Figure 17 (this could be a track-and-hold in track mode, an amplifier, an RC network, etc.). The total output response to a linear segment on the input is a sum of exponentials. The last term depicts the decay of the initial state, i.e. the previous voltage on the capacitor slowly drains away. The rest of the terms constitute the driven response due to the linear segment. This response can be further decomposed into a response to the step and a

response to the ramp. The response to the step is represented by term number three – an exponential approach to the step value. The response to the ramp is represented by the first two terms, in which the first term indicates that the output voltage tracks the slope on the input, while the second term indicates a finite error in the tracking that converges to -βτ as time approaches infinity. Other linear systems will yield similar closed-form equations that compute the module's output, as a function of time, in response to a linear ramp on the input.

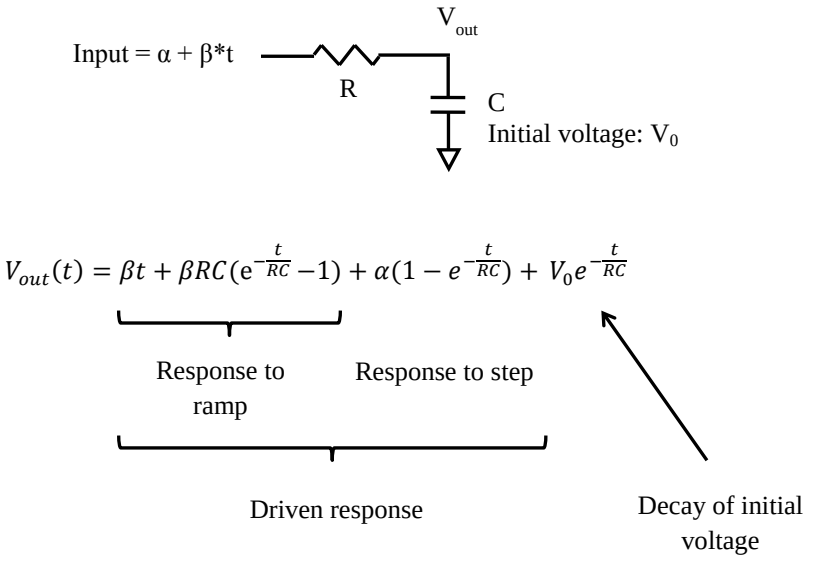

Figure 17 – Time domain response of single pole system to a linear input segment

3.4.2 Forming Piecewise-Linear Output

The previous section enables the computation of a module's continuous time output waveform when it is stimulated by a linear ramp. It is necessary now to convert this waveform into piecewise linear segments for the following block to process. In order to obtain a reasonable segmentation, the time constants in the system need to be examined. Continuing with the single-pole system example, and assuming a step input (the slope element is 0), Figure 18 illustrates output waveforms of two systems with different pole locations. Systems that respond quickly (i.e. small time constants) produce

signals that change more quickly and therefore require shorter linear segments. In Figure 18, T1 (less than T2) is needed for the faster system.

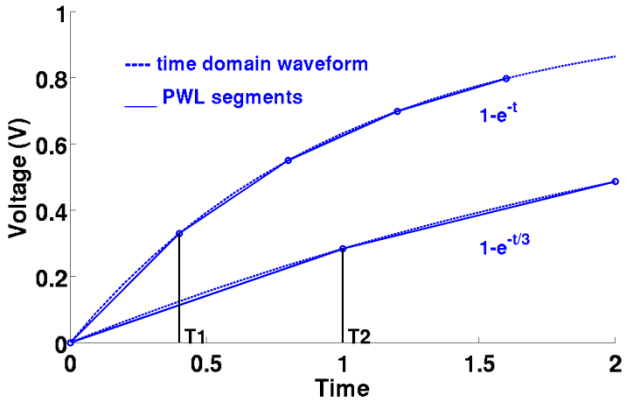

Figure 18 – PWL segment length for systems with different time constants

The general idea is to estimate the length of time after which the output response deviates too much from a linear ramp, output immediately the segment describing the signal from now until that time, and return after that length of time has expired to compute the output waveform again to determine the next linear segment. For most linear systems, the time until the output waveform deviates too much from a ramp is a fraction of the smallest time constant in the system that's of importance. Figure 13 from Section 3.2.2 can, in addition, be used as a guideline to estimate this length of time using the smallest time constant that's important as the equivalent f_{sig}. The amount of spur that can be tolerated will be upper-bounded by the noise performance of the circuit in question as well as the bandwidth of the subsequent block (a lower bandwidth block will attenuate the spurs more than a high bandwidth block and therefore, larger spurs could be acceptable with little impact on the performance of the design as a whole).

3.4.3 Filtering Output Updates

The procedures described so far will be able to propagate an analog signal in piecewise linear format through a chain of analog modules. Each input update will result

in one or more output updates (which will become the input updates for the next block) and each update constitutes a value and a slope. The caveat is that with Boolean logic, it is very clear when an output changes, but with continuous time signals it is not so clear. A procession of updates might be all within some delta from each other, and this will cause an unnecessarily large number of events. In designs with feedback, the number of events might even grow unboundedly. In Figure 19, suppose that there is a single input update and this leads to an update on the error signal Δin, which then causes a change on V_{out}. Because of the feedback loop, V_{out} will cause V_{fb} to change, which then forces a second update on Δin. These updates of little or no real change in signal value will continue forever without any advancement in time and the simulator will hang.

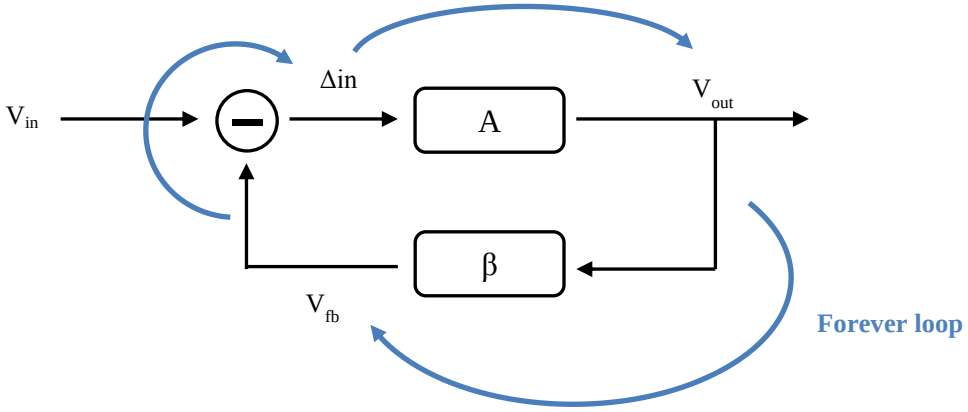

Figure 19 – Forever loop caused by a single input update

One way to eliminate these forever loops is through the observation that signals that have "settled" don't need to be updated. Here, "settled" isn't taken in its traditional sense of the signal approaching a zero slope, but rather the signal having a similar slope as the previous output segment, i.e. approaching a linear ramp. Listing 3 illustrates a test function that can be used to determine whether the signal has settled. The newly calculated output segment is compared to the previous segment by extending the previous segment along its slope and determining the difference in the final value of both segments. If the final values differ by more than the tolerance set by the user, then the function

returns true and the output is updated with the new numbers for its value and slope elements. If the function returns false, the output of the module is not updated, hence preventing blocks in feedback from computing more updates. Figure 20 illustrates the above operations graphically.

```
function update (pwl_struct old_seg, new_seg);

  real old, new;

  update = 0;

  old = old_se.g.v1 + (dT+T)*old_se.g.slope;
  new = new_se.g.v1 + T*new_se.g.slope;

  if (abs(old – new) > `ERR_TOL) update = 1;

endfunction
```

Listing 3 – Filtering unnecessary output updates

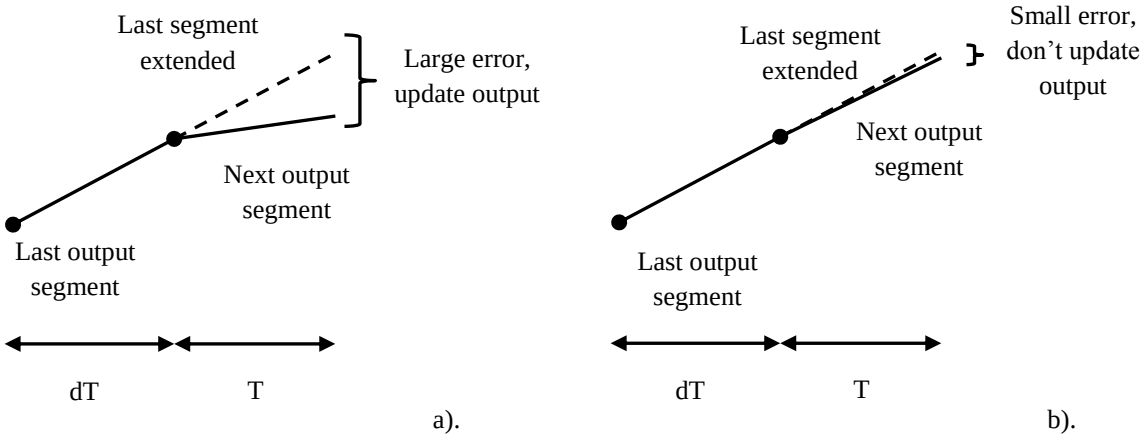

Figure 20 – Filtering unnecessary output updates

3.5 Summary

This chapter establishes three goals of behavioral modeling for mixed-signal validation – the models need to be pin-accurate to facilitate equivalence checking between model and schematic, fast so that many test vectors can be run with practical time costs, and they need to capture circuit dynamics to permit a rough performance evaluation of the entire SoC. Then, a set of guidelines to achieve these goals is described. Circuits need to be first partitioned into unidirectional blocks so that they fit into the digital simulation environment. Any analog signals need to have supplemental information in order to accommodate their asynchronicity in a discrete-time simulator. In this case, the piecewise linear approximation is the chosen representation. The smoothness of the analog output surface and the linear intent of circuits are leveraged to quickly compute a module's output waveform in closed-form equations. Lastly, piecewise linear segments of the waveform are created with extra care taken to avoid unnecessary updates. Designer's knowledge is heavily intertwined with each step in writing a behavioral model, from circuit partitioning, to selecting an appropriate domain of linearity that allows module outputs to be evaluated quickly, to taking into consideration time constants in a system for output waveform segmentation. The next chapter will demonstrate how these techniques can be utilized in the creation of behavioral models for different types of analog circuits.

Chapter 4

Creating Analog Behavioral Models

Given the general approach described in Chapter 3, it is now possible to model a variety of analog circuits. This chapter provides more details on how to create these models. Once the models are created, it is necessary to check that they are being used in the region for which they have been characterized and modeled. Therefore the creation of models is discussed first in Sections 4.1 to 4.4, and Section 4.5 then describes how to use assertions to perform these operating region checks.

Since analog circuits are very diverse in nature, a rough categorization of analog functionality will be used to help structure the discussion of modeling details in the first four sections. The division is based on input/output port characteristics and is illustrated in Figure 21.

Output / Input	Analog	Digital
Analog	A-to-A	A-to-D
Digital	D-to-A	D-to-D

Figure 21 – Circuit categories based on input/output characteristic

Some analog signals are analog in the voltage domain, i.e. the detailed shape of the signal is necessary to describe system behavior, and ports carrying these signals are branded as analog ports. Other analog signals affect system behavior only at their transition edges. In these cases, all the analog information is contained in the switching points. As a result, these signals may be represented as binary values, and hence their corresponding ports are labeled as "digital". Under this categorization, analog sub-circuits can be divided into four groups as shown in Figure 21.

It turns out that the most fundamental analog functionality is analog inputs and outputs (the A-to-A category). These circuits filter some analog input to produce their outputs. All the other circuit categories (D-to-D, A-to-D and D-to-A) are either variations or can be built with the A-to-A circuits as a foundation. Given the importance of the A-to-A category, models of this type of circuit will be discussed first in Section 4.1, followed by a section for each of the remaining three circuit groups. A specific circuit will be used for demonstrative purposes for each category, since models of circuits in the same category are similar. In addition, recall that one of the goals of behavioral modeling is to obtain some rough estimate of a system's performance and because no circuit is ideal, each representative model will also illustrate some methods for including non-ideal behavior from the circuit.

4.1 A to A (Filter-like) Circuits

The behavior of this class of circuits can be roughly described as "filter" in that the analog input is filtered through a transfer function to produce an analog output. The shape of the input and output waveforms have significant impact on the performance of the circuit itself and the overall system. Examples of circuits that fall into this category include programmable gain amplifiers (PGA), transimpedance amplifiers (TIA), continuous time linear equalizers (CTLE), continuous time filters (RLC networks,

biquads, etc.) and linear voltage regulators. The guidelines presented in Chapter 3 can be pieced together without much modification to form a model of these filter-like circuits.

The pseudo-code for a generic A to A module is shown in Listing 4. The module's ports consist of a PWL input and a PWL output, as expected. The variable *out_int* is utilized as a memory element to keep track of the current, internally calculated output value (recall that due to output filtering, the value on a module's port may differ by a user set tolerance). To control the formation of a piecewise linear output, the *always* loop is activated either when the input changes or when the internal evaluation signal is triggered, indicating the time between internal evaluations, and it is time to compute the next segment on the output. *T* is usually a fraction of the smallest time constant in the system being modeled, as discussed in Section 3.3. The *always* loop performs three actions. First, it calls the *compute()* task to determine the circuit's response to the input linear segment. Then, it applies the *update()* function to the newly computed output and determines whether the output should be updated. The last action is to schedule an output evaluation point after a time period of *T* has expired, so that the next linear segment on the output can be computed. The *compute()* function is at the core of the A-to-A module since it is responsible for computing the output waveform. The starting value of the module's output at the current time is first calculated based on the response of the system to the input starting from the last time the function *f* was evaluated to the current time. Function *f* is the closed-form equation that describes the system's response to a linear ramp, as discussed in Section 3.3. The resulting value is stored in the memory element *out_int*. Then the behavior of the system is projected forward, using the same function *f*, to a time period of *T* later. This second value is stored in a temporary variable *out_in_T*. The second task performed by the *compute()* function is to formulate the newly calculated output in the predefined *pwl_struct* that delineates a piecewise linear segment. The *v1* element of the *pwl_struct* is the value of the variable *out_int*, while the *slope* element is computed using the value of *out_in_T* in conjunction with *out_int* and *T*. Variable *out_curr* is used to hold this structure temporarily before it is determined

whether to update the module's output port to the new values in this structure. The helper function *update()* rounds out the components of this generic A-to-A module. An example of this function was presented in Section 3.3.3, in which the decision to update the output is made by examining the difference between extending the current output along its slope and updating to the new values.

```
module A2A (
  input pwl_struct in,
  output pwl_struct out
);

real out_int;

always @ (in or eval) begin
  compute();
  if (update(out, out_curr)) out = out_curr;
  #T eval.trigger;
end

task compute();
  out_int = f(in_prev, current_time-previous_compute_time, out_int);
  out_in_T = f(in, T, out_int);

  out_curr.v1 = out_int;
  out_curr.slope = (out_in_T – out_int) / T;
endtask

function update (pwl_struct old_seg, new_seg);
  // compares old_seg with new_seg
  // see section 3.3.3 for example
endfunction
```

Listing 4 – Generic A-to-A module

The various analog sub-circuits that belong to the A-to-A filter group (PGA, TIA, CTLE, RLC networks, etc.) can now be modeled as variations of this generic module.

The distinguishing tone of each of the circuits lies in their respective transfer function; hence, applying the generic model to the different blocks entails replacing the function *f* in the *compute()* task with the appropriate closed-form equations. A single-pole system would use the equation derived in Figure 17. Transfer functions of higher order systems with multiple poles and zeros can be decomposed via the partial fractions method into a series of single pole/zeros transfer functions, and therefore, the function *f* would take the form of a sum of exponentials.

The *compute()* function also plays an important role in the inclusion of non-idealities when modeling these circuits. In some cases, the non-idealities present themselves as a varying transfer function (and hence a varying function *f*) whose coefficients change depending on the value of the inputs. For instance, the gain of an amplifier may decrease as the input amplitude increases. Or in the simple sample and hold shown in Figure 9, the on resistance of the transistor may fluctuate as the voltage across its gate and source nodes varies, and in effect, the time constant of the system becomes non-constant throughout simulation. In other cases, the non-idealities can be modeled as an additional transfer function from a second input to the circuit's output. For example, the supply voltage of an amplifier could affect its output behavior. Since the supply typically does not vary very much and amplifiers are designed to have good power supply rejection, the power supply effect can be taken as an independent transfer function from the supply input to the amplifier output. Whatever the imperfection-causing agents are, SPICE simulations are needed to determine how the basic transfer function changes. Listing 5 illustrates the use of parameters to modify a block's output computation function *f* according to input values, as well as the addition of a second transfer function to the output. Of course, many more possibilities exist given the host of mathematic operations, data structures and general programming capabilities that are available in SystemVerilog. The functions *g1* and *g2* need to be extracted from SPICE simulations, and while in Listing 5 these are cast into algebraic functions, other forms such as lookup table could also be used.

```
module A2A_nonideal (
  input pwl_struct in,
  output pwl_struct out,
  input pwl_struct vdd
);

…

task compute();
  // input amplitude effects on circuit gain
   gain =  g1(in);

  // input/output effect on circuit time constant
  tau = g2(in, out);

// compute output using sum of 2 transfer functions to include
supply effects
  out_int = f1(in_prev, t, out_int, gain and/or tau) + f2(vdd_prev, t,
out_int);
  …
endtask

endmodule
```

Listing 5 – Inclusion of non-idealities in A-to-A modules

4.2 D to D Circuits

Circuits under this classification have inputs and outputs with all their analog information contained in the timing of their transition instants. The details in the shape of the waveforms are not important. Therefore, the models actually have binary (0 or 1) signals on their ports; however, these circuits are different from normal digital logic in the sense that they do not need to be composed entirely of digital gates and it is only that their domain of smoothness is in the time or delay domain so that their inputs and outputs may be represented as “digital” signals with correct transition times. As a result, a D-to-D

sub-circuit is typically a block that changes the timing of clock signals or digital circuits that require detailed description of their delays. Examples include phase interpolators, duty cycle correction circuits and digital circuits under the influence of supply voltage fluctuations. Delays in SystemVerilog (in fact, any flavor of Verilog) are denoted by the # directive and writing models for D-to-D circuits entails the composition of appropriate equation(s) to quantify the amount of delay needed. A similar kind of analog modeling is done here as with the A-to-A models; the difference is that the computation domain is time or phase and the analog information is expressed as a change in time rather than a change in voltage.

Listing 6 provides the pseudo-code for a phase interpolator with a possible implementation shown in Figure 22. The purpose of an interpolator is to construct a clock signal with a phase that's somewhere in between the two input clocks, clk1 and clk2. Figure 23 shows representative waveforms on the inputs and output. As *wt* increases from 0000 to 0011 to 1111, more inverters pass clk2 as their output resulting in a stronger presence of the second phase such that the interpolated clock is delayed more. Note that both the inputs and output are full-rail square waves, and therefore are not smooth in the voltage domain. Nonetheless, the delay of the interpolated clock is a linear function (under ideal conditions) of the digital control bits, *wt*, and the phase/delay difference between the two input clocks. Thus, the analog behavior of the phase interpolator is modeled in time domain.

The particular implementation shown has additional configuration bits including the 4-bit *wn* which adjusts the drive strength of the pull down transistors in order to modify the duty cycle of the output clock, and the 4-bit *cap* which tunes the time constant of the circuit so that across all process corners, the input clock edges are slowed down enough to permit a smooth interpolation, but not too slow so as to cause excess jitter. These will be treated as non-idealities in the phase interpolator and discussed later in this section. The validated operation region of this circuit is limited to a maximum input clock

frequency as well as a maximum time difference between clk1 and clk2. Assertions to check these operating conditions will be discussed later in Section 4.5.

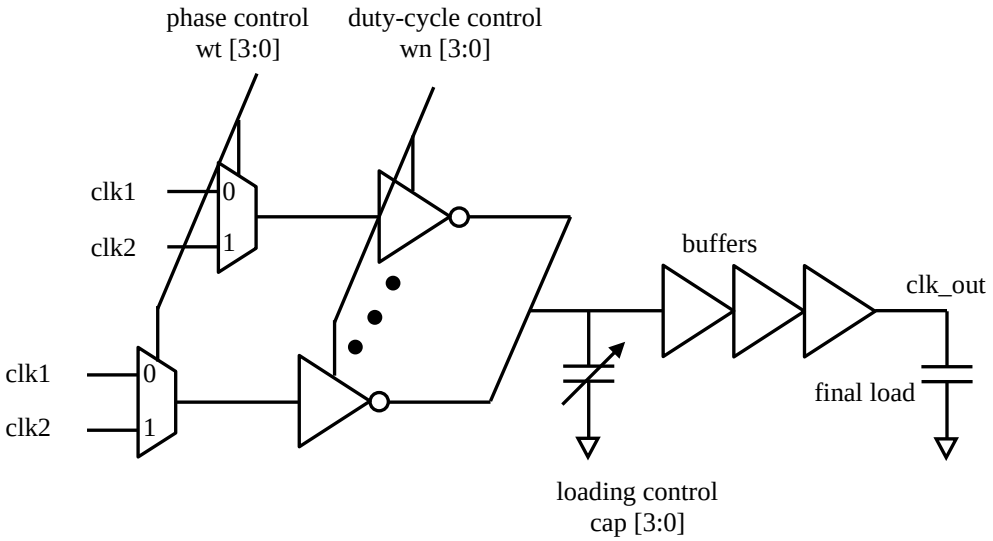

Figure 22 – A phase interpolator implementation

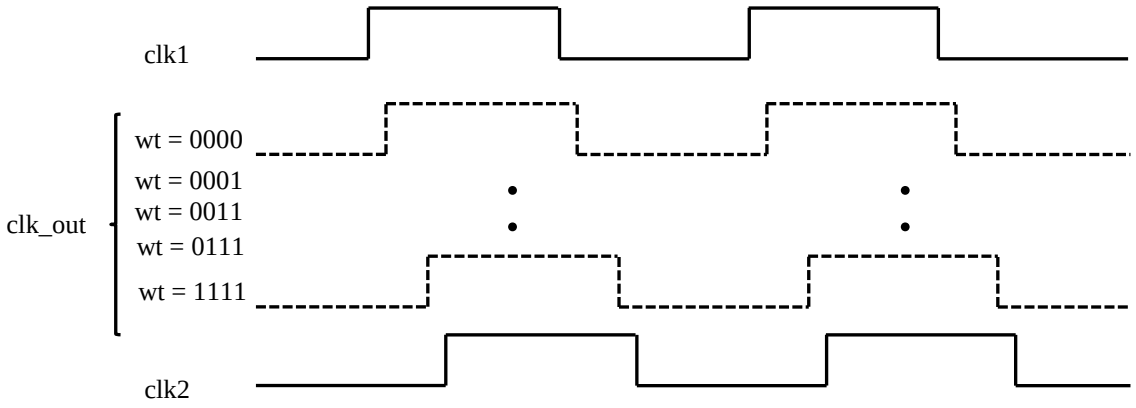

Figure 23 – Phase interpolator sample waveforms

In the ideal model, an *always* loop is used to calculate the fraction to be applied to the input clock delay difference to determine the overall delay of the interpolated output. The function *toDec()* helps this process by first converting the *wt* bits into an integer. Concurrently, a second *always* loop sensitive to level changes in *clk1* is used to record the

times at which *clk1* exhibits an edge. The final two *always* loops react to edges in *clk2* and they use their time of activation as well as the edge time from *clk1* to determine the phase difference between the two clocks (denoted *d*). Lastly, the interpolation delay is calculated and the output is toggled to the correct value after this delay.

```
module interpolator (
  input clk1, clk2,
  input [3:0] wt, wn, cap,
  output clk_out
);

  always @ (wt) frac = toDec(wt) / 4;

  always @ (clk1) t = get_current_time();

  always @ (posedge clk2) begin
    d = get_current_time() – t;
    clk_out <= #(insertion_delay + d*frac) 1;
  end

  always @ (negedge clk2) begin
    d = get_current_time() – t;
    clk_out <= #(insertion_delay + d*frac) 0;
  end

  function toDec (input [3:0] code);
    case (code)
      4'b0000: toDec = 0;
      4'b0001: toDec = 1;
      4'b0011: toDec = 2;
      …
    endcase
  endfunction

endmodule
```

Listing 6 – Ideal phase interpolator module

Non-idealities such as the duty cycle correction control (*wn*), edge rate control (*cap*) and supply (*vdd*) effects can be modeled by including these as parameters in the function that computes delay. Listing 7 provides an illustration.

```
module interpolator_nonideal (
  input clk1, clk2,
  input [3:0] wt, wn, cap,
  input vdd,
  output clk_out
);

…

  always @ (posedge clk2) begin
    d = get_current_time() – t;
    calculate_delay();
    jitter = $dist_normal(seed, mean=0, sigma=insertion_delay*x%);
    delay = insertion_delay + jitter;
    clk_out <= #delay 1;
  end

…

  task calculate_delay();
    insertion_delay = f(wt, wn, cap, d, vdd);
  endfunction

endmodule
```

Listing 7 – Phase interpolator with non-idealities

SPICE simulation is first performed to extract a mapping of *wt*, *wn*, *cap*, *d* and *vdd* to the final phase interpolator delay. Ideally, all these have a linear relation to the delay. Therefore, to accommodate some nonlinearity, it can be hypothesized that the

relationship is modeled by a polynomial (13) whose coefficients are determined by fitting the SPICE results.

$$delay = (\alpha_1 wt + \alpha_2 wn + \alpha_3 cap + \alpha_4 d + \alpha_5 vdd)^p \quad (13)$$
$$= \sum \beta_i (wt)^j (wn)^k (cap)^l (d)^m (vdd)^n$$
$$where\ p\ is\ the\ order\ of\ nonlienarity\ modeled$$
$$and\ 0 \leq j + k + l + m + n \leq p$$

The order of the polynomial depends on the desired fitting error. Lower order models may result in larger errors, but would have fewer terms and hence require less computation. It is at the discretion of the model-writer to select an equation template appropriate for his/her particular application. Finally, jitter due to transistor noise can be added on top of the above non-idealities using any of the random number generator functions available in SystemVerilog. Listing 7 utilizes the $dist_normal function which generates a Gaussian random variable.

To summarize, the conceptual foundation underlying D-to-D models are similar to that of A-to-A models. Because all the analog information is contained in transition timing, the signals on the input and output ports of this class of circuits have binary values, and while the model syntax may look different from the A-to-A models since time is handled differently from voltage values by the modeling language, the same idea of “filtering” an input to produce an output with small deviations from a perfectly linear response applies. In the A-to-A case, the domain of smoothness and thus computation is mostly voltage or currents, and for D-to-D circuits that domain is time.

4.3 A to D Circuits

Some examples of circuits in this category include comparators, latches and voltage controlled oscillators (VCO’s); they share the common characteristic that their

behavior depends on the details of the input waveforms while the downstream blocks consider only the transition instant of the output generated by these blocks. As a result, the inputs to these circuits are in piecewise linear form and their outputs are digital signals. The solution to the construction of the corresponding models can be simplified if one realizes that these circuits can be thought of as an analog circuit followed by an ideal slicer. Thus all A-to-D circuits can be modeled by appending an ideal slicer to an A-to-A module which, fortunately, has already been solved. This model construction is illustrated in Figure 24. The analog circuitry and dynamics of the circuit are capture by the A-to-A frontend, while the ideal slicer converts the detailed waveform into transition edges by calculating the time at which point the waveform crosses a threshold. Circuitry after the slicer can be modeled by digital gates.

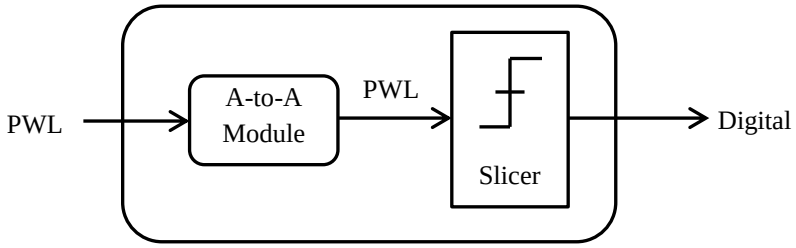

Figure 24 – A-to-D module block diagram

Listing 8 provides the pseudo-code for an ideal slicer with a parameterized threshold. When the input changes, a time *t* until the signal crosses the threshold is computed from the value and slope of the input. If the threshold is greater/less than the value of the input, and the input has a positive/negative slope, then a crossing will occur. In either case, the computed value of *t* is positive, and this is the condition used to determine if the output needs to be toggled. Finally, the output is changed in the computed amount of time to the correct value according to the polarity of the input slope. If the input changes before the scheduled output change occurs, then the newly determined output event will override the previous one. The benefit of parameterizing the

threshold is that this module can be initialized multiple times to different thresholds and each instance overwrites the default threshold value of the module.

```
module slicer (
  input pwl_struct in,
  output out
);

  parameter threshold = 0;

  always @ (in) begin
    t = (threshold – in.v1) / in.slope;
    if (t > 0) begin
      if (in.slope < 0) out <= #(t) 0;
      else out <= #(t) 1;
    end
  end

endmodule
```

Listing 8 – Ideal slider module

To illustrate the usage of the slicer module, the circuit example for this section will be a comparator whose output is either digital high (represented by 1 in the model) or low (0 in the model). Listing 9 outlines the model of the comparator.

This comparator module internally instantiates three sub-modules. The A2A frontend would depict the signal dependency of the comparator delay. The slicer module is the ideal slicer presented in Listing 8 parameterized such that it toggles its output when the filter frontend produces a signal that crosses half the supply voltage or some other reference voltage. The last block is a D-to-D output buffer that delays the comparator output depending on the final loading (since the ideal slicer assumes no load). It is useful to parameterize this block with the technology specific “fanout of 4” value so that it is easily reusable for different technology nodes. With the three components working in

succession, the digital signal provided to subsequent circuits will properly reflect the effect of the comparator's dynamics on signal timing.

```
module comparator (
  input pwl_struct in,
  output out
);

  parameter comp_thresh = vref;

  A2A frontend (.in(in), .out(pwl_struct frontend2slicer));

  slicer #(.threshold(comp_thresh))
     slicer (.in(pwl_struct frontend2slicer), .out(out_ideal));

  outbuf #(.cload(cap_val)) outbuf (.in(out_ideal), .out(out));

endmodule

…

module outbuf (
  input in,
  output out
);

  parameter cload = 10fF;

  always @ (in) out <= #(m*cload+b) in;

endmodule
```

Listing 9 – Comparator module

Non-idealities can be attached to all three sub-modules of the comparator if a more detailed description of the circuit is necessary. Listing 10 illustrates a few possibilities. The comparator performance could suffer from input offset errors, for

example; or the comparator has a tunable offset. The effect could be conveyed through the adjustment of the input signal to the A-to-A frontend that subtracts out the offset voltage. The operating point of the comparator may be subject to different bias conditions as well as supply voltages. Both effects can be modeled together in the A-to-A frontend as a parameterized transfer function or as a "switched" system in which the transfer function is chosen among several versions depending on the value of the bias control and supply voltage. In the slicer sub-module, a different supply voltage means setting the threshold to a different value. A pin can be added to the slicer for this purpose instead of using a fixed-upon-run-time parameter. All the above imperfections can be categorized as changes to the circuit dynamics due to controlled inputs and this method of modeling is valid as long as the bandwidth of these controlled inputs is much less than that of the signal bandwidth (which is usually the case).

Any noise, jitter and stochastic non-idealities of the comparator can be lumped into the final D-to-D block as a random variable on the delay. To describe all of the above non-idealities, simulations need to be first carried out in a SPICE simulator and the result either recorded as a lookup table or fitted to a parameterized function, which then is incorporated into computations in the model.

```
module comparator (
  input pwl_struct in,
  input vdd,
  input [2:0] bias, [2:0] offset,
  output out
);

…

  always @ (offset) begin
    case(offset)
    3'b000: in_adj = in – vos0;
    3'b001: in_adj = in – vos1;
    …
    endcase
  end

  A2A frontend (.in(pwl_struct in_adj), .vdd(vdd), .bias(bias), .out(pwl_struct
frontend2slicer));
  slicer slicer (.in(pwl_struct frontend2slicer), .threshold(vdd/2), .out(out_ideal));
  outbuf outbuf(.in(out_ideal), .out(out)); // add jitter, see Listing 7

endmodule

module A2A (…);

…

  task compute();
    // see Listing 4 and Listing 5
    // replace f with the appropriate transfer function
    // parameterized by bias and supply
  endtask

endmodule
```

Listing 10 – Comparator with non-idealities

4.4 D to A Circuits

D-to-A circuits are the opposite of A-to-D circuits; their behavior hinges on the distinct instances of change on their inputs (and sometimes the rise/fall slopes), and their outputs are analog in nature because the circuits for which they provide inputs require detailed features of the signals in order to capture their own dynamics. Examples of D-to-A circuits include digital to analog converters (resistor string, current, capacitor, etc.) and charge pumps. Because of their opposite behavior compared to A-to-D circuits, the construction of the model for the D-to-A is, naturally, a flipped version of the A-to-D circuits. This construction is portrayed in Figure 25. Here, the filter-like module which captures any circuit dynamics is preceded by an ideal digital to piecewise linear conversion module.

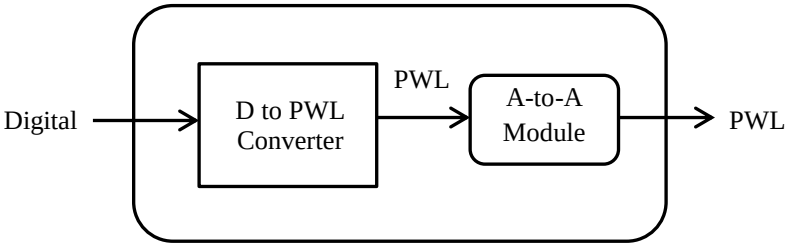

Figure 25 – D-to-A module block diagram

To use a current DAC as an example, the outline of the model is shown in Listing 11 (refer to Figure 5 for a representative circuit implementation). The unit current sources or binary weighted sources naturally constitute individual modules and it might be tempting to leave them as such; however, as posited in Chapter 3, current summation needs to be encapsulated in a module. Therefore, all the current sources must be combined with the output buffer to yield a module with a single voltage output and this is the module sketched out in Listing 11. Though the individual current sources still exist as sub-modules, a wrapper encloses all of them and it is this larger module that should be used in schematic-model equivalence checking.

```
module DAC (
  input [3:0] Din,
  output pwl_struct out
);

  D2A #(.iout(20e-6)) I0 (.Din(Din[0]), .out(pwl_struct i0));
  D2A #(.iout(40e-6)) I1 (.Din(Din[1]), .out(pwl_struct i1));
  …

  assign Iout = i0 + i1 + i2 + i3;

  A2A i2v (.in(Iout), .out(out));

endmodule

module D2A (
  input Din,
  output pwl_struct out
);

  parameter iout = 20e-6;

  always @ (posedge Din) begin
    pwl_out.v1 = iout;
    pwl_out.slope = 0;
  end

  always @ (negedge Din) begin
    pwl_out.v1 = 0;
    pwl_out.slope = 0;
  end

endmodule
```

Listing 11 – Current DAC module

```
module DAC (
  input [3:0] Din,
  output pwl_struct out
);

  D2D #(.del(48ps)) D0 (.in(D[0]), .out(Dmod[0]));
  D2D #(.del(51ps)) D1 (.in(D[1]), .out(Dmod[1]));
  …

  D2A #(.inom(20e-6)) I0 (.Din(Dmod), .n(0) .out(pwl_struct i0));
  D2A #(.inom(40e-6)) I1 (.Din(Dmod), .n(1), .out(pwl_struct i1));
  …

endmodule

module D2A (
  input n, [3:0] Din,
  output pwl_struct out
);

  parameter inom = 20e-6; // nominal current
  initial iout = $dist_normal(seed, mean=inom, sigma=1% inom);

  always @ (posedge Din[n]) begin
    pwl_out.v1 = 0;
    rt = g(Din[3:0]); // variable rise time
    i_n = $dist_normal(seed, mean=0, sigma=0.01% inom); // current noise
    pwl_out.slope = (iout + i_n) / rt;
    #rt pwl_out.v1 = iout + i_n;
    pwl_out.slope = 0;
  end

  always @ (negedge Din[n]) begin
    …
  end

endmodule
```

Listing 12 – Non-ideal current DAC module

The example model in Listing 11 works as follows. When the digital control bits change, they are converted to piecewise linear current outputs. The D2A modules handle the conversion by attaching a slope of zero (or the actual rise and fall time) to the digital input and outputting a PWL structure. All the currents are then summed together forming a staircase signal in the main module. Once the digital input is converted, the *i2v* module is an A-to-A block that is used to transform the staircase current signal into a voltage. In the ideal case, the final voltage output is also a staircase signal, but other desired shapes are possible by modifying the transfer function of the A-to-A module.

Listing 12 demonstrates several non-idealities of the DAC circuit. The digital control bits might have different path delays and pick up noise in the form of jitter along their way to the DAC. This is taken care of with a group of D-to-D modules that adjust and randomize the timing of each bit individually. The staircase current signal from the ideal version of the model might be too optimistic, and for certain applications, it might be beneficial to describe the transient behavior of the current step transitions. The approach to modeling this would be to include the rise and fall times by attaching non-zero slopes in the conversion process. The slopes could be different as the output impedance of the DAC is changed by different numbers of current sources being connected to the output. Relaying this information to the individual current sources is accomplished by passing the entire code after timing adjustments, instead of a single bit, to each D-to-A module and varying the rise/fall times accordingly. Furthermore, the current sources could be affected by transistor mismatch and have different static output current. This mismatch can be emulated as a random variable that’s initialized once during the instantiation of each current source. Lastly, noise of the current sources and buffer can be superimposed on their respective outputs using the various random generators in SystemVerilog (Listing 12 opts to use the Gaussian distribution as in the previous sections). As usual, all the necessary data would be extracted from SPICE simulations before being incorporated into the model.

So far, models of four different groups of analog circuits have been created. The next critical step is to ensure that these models are not being used outside the region for which they have been tested and validated. The following section will deal with constructing automatic checks that detect any such violations.

4.5 Assertions

When writing any model, the coverage of behavior is limited, whether it's the fundamental behavior or the non-idealities, to the environment in which the SPICE level circuits have been explored. For example, common mode ranges are generally limited, so are acceptable peak input amplitudes and supply ranges. If during simulation, the model's inputs wander outside the region for which the circuit has been tested, there is no guarantee that the behavior can be well described by an extrapolation of the behavior from the verified region. As a consequence, the models have an obligation to inform users when their inputs are out of bounds.

SystemVerilog provides assertion capabilities through the *assert()* procedural statement and the usage is demonstrated in Listing 13. When the conditional expression of the *assert* evaluates to x, z or 0, then the assertion fails and the simulator writes an error message. Several severity levels are available: $fatal, $error (the default), $warning and $info. In Listing 13, the amplifier module checks for the correct range of bias current and supply voltage. The bias current has an additional polarity characteristic that requires an assertion. To distinguish P and N MOS current sources and sinks, it is good practice to adhere to, for instance, a positive polarity for a PMOS source and NMOS sink. The time at which an assertion fails may also be recorded as shown in the check for supply range. It is useful to have such a time stamp on the assertion because the amplifier might be powering up or down so that assertion fails during these times are not errors and can be filtered out based on the time stamp. Listing 13 also includes assertion examples for the phase interpolator from section 4.2. Recall that the interpolator has been verified for a

limited phase difference between the two input clocks and the *wt* control bits are thermometer coded. Therefore, the model must assert error when the phase difference is too large or the control bits accidentally became binary coded.

```
module amplifier (…
  input vdd, ibias
);

  always @ (vdd) begin
    assert (vdd < vdd_max & vdd > vdd_min)
      else $error("%t: Supply out of range. Comparator may not work.", $time);
  end

  always @ (ibias) begin
    assert (ibias > 0) else $fatal("Inverted comparator bias current direction.");
    assert (ibias < ibias_max & ibias > ibias_min)
      else $error("Bias current out of range. Comparator may not work.");
  end

endmodule

module interpolator ( …
  input [3:0] wt, wn, cap,
  input vdd
);

  always @ (d) begin
    d_max = f(cap);
    assert (d < d_max) else $error("Input phase difference too large.");
end

  always @ (wt)
    assert (wt is thermometer coded) else $error("wt control bits out of range.");

endmodule
```

Listing 13 – Assertions Examples

4.6 Summary

The creation of a complete behavioral model requires two components: a description of the circuit function (the model) and a set of assertions that describe when the model can be trusted. The trust region is coded by using assertions. The model itself can be realized using techniques introduced in Chapter 3. Although analog circuits are highly diverse, being able to model all the circuit types fundamentally depend on being able to analyze filter type circuits – the A-to-A modules. The other three groups discussed in this chapter are either a variation of these filter modules or can be built by adding simple modules to these filters. D-to-D blocks process delay or manipulate timing of clock signals, and therefore can be thought of as A-to-A blocks in the time or phase domain. The A-to-D models can be obtained by placing a slicer after an A-to-A module, while placing a digital to PWL converter in front of an A-to-A module produces a D-to-A module. Additional D-to-D blocks can then be positioned at the digital end of either A-to-D or D-to-A in order to create more realistic timing in these blocks.

Chapter 5

Experimental Results

Chapter 3 first laid out the fundamental concepts needed to create behavioral models well-suited for mixed-signal SoC validation, and then Chapter 4 detailed the construction of models for different types of commonly encountered circuit modules. In this chapter, larger analog circuits will be modeled by drawing on insights gained in the two previous chapters. The behavior of the composite models as well as their simulation speed will be compared with that of SPICE simulation, in order to verify that they exhibit the three desirable characteristics (pin-accurate, fast and capture realistic circuit behavior) that enable system-level validation. In the last section of this chapter, a breakdown of CPU activity during SystemVerilog simulation will be presented in order to propose a possible acceleration technique to further shorten simulation time of the models.

5.1 Track and Hold

For any system that interface to the outside world, an analog-to-digital converter (ADC) is a critical component [95], and while the architecture of the ADC could be one of many, most ADC's are sampled data systems, and work better if their input doesn't change during the conversion (for example, improved speed and linearity [97]). Thus

most converters have an explicit track and hold (T/H) circuit to create a stable voltage when processing is performed. An important metric for these T/H circuits is the distortion performance (in terms of spurious-free dynamic range, SFDR) and this section will demonstrate that using the techniques formulated in Chapters 3 and 4, this behavior can be captured during behavioral modeling.

5.1.1 Circuit Description

Figure 26 depicts the block-level diagram of a 250MS/s open-loop bottom-plate-sampling track and hold, which is one of many architectures used for a T/H. The differential circuit accommodates an input common mode of 0.7V, an output swing of 1.6Vpk-pk and an output load of 500fF. Figure 26 also illustrates the timing details of the circuit's set of switches that performs the bottom-plate sampling operation. The frontend samplers are manipulated such that the switch (p1e) on the bottom plate of the sampling capacitor opens slightly earlier than p1, so that there will be very little signal dependent charge injection once the input signal is sampled on the capacitor. This switching arrangement improves the linearity of the samplers.

The main source of nonlinearity in this differential sampler is caused by the variable on-resistance of the transistor during tracking mode, and this manifests as higher order distortion. For example, if the quadratic model of a transistor were used, the third order distortion HD3 can be estimated by equation (14)

$$\mathrm{HD3} \cong \frac{1}{4}\frac{A^2}{(V_{GS}-V_t)^2}2\pi f_{in}RC_S \tag{14}$$

where A and f_{in} are the amplitude and frequency of the input, R is the quiescent resistance of the sampling switch, and C_s is the sampling cap [101]. Even though equation (14) is presented here, it will not be used for modeling purposes; instead, it will help give insight into how to design this circuit. Looking at equation (14), since the frequency and

amplitude of the input signal are often not under the control of the designers, the main route to improve linearity is to suppress the contribution from the biasing voltages. To this end, the samplers' gates are bootstrapped to a voltage one supply (1.8V) above the input signal during the tracking phase and returns to 0 during the sampling phase. The concept of a bootstrapped NMOS sampler is illustrated in Figure 27 [96].

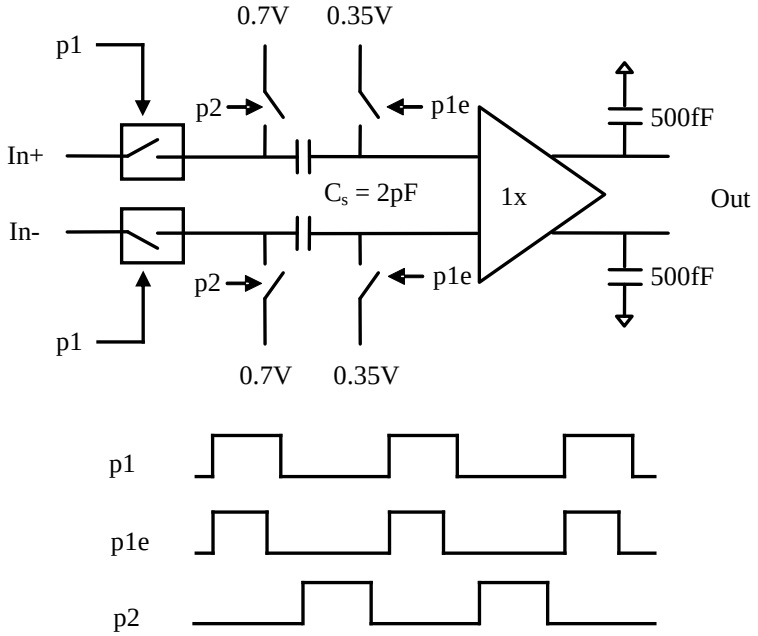

Figure 26 – Track and hold circuit block diagram

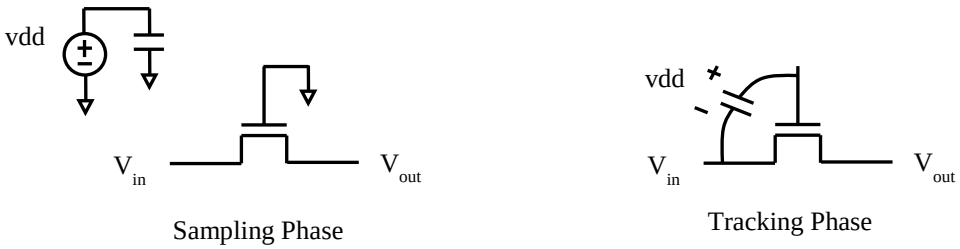

Figure 27 – Bootstrapping concept

In order to drive the large load specified, an output buffer in the form of a pseudo-differential source follower is employed. The nonlinearity of the source follower structure

can be observed from its large signal transfer characteristics (again based on quadratic transistor models):

$$\mathrm{V_o} = V_i - \left(V_{t0} + \gamma \left(\sqrt{2\phi_f + V_{SB}} - \sqrt{2\phi_f} \right) \right) - \sqrt{\frac{2I_D}{\mu C_{OX} \frac{W}{L}}} \tag{15}$$

In a source follower, the source node voltage is the output voltage; therefore, the dominant source of distortion comes from the second term in the bracket, which is the output dependent threshold voltage of the transistor (also known as the body effect). One method of improving the distortion is by tying the body terminal of the input transistor to the source node. With the most commonly used N-well process, this connection can only be made on the PMOS transistors. Therefore PMOS is used for the source follower; however, this leads to the issue of incompatible common mode voltage - the required 0.7V from the input signal is too high for PMOS transistors. Thus a set of switches between the frontend sampler and the output buffer is added to translate the common mode voltage from 0.7V to 0.35V.

The aggregate non-linearity of the track and hold is composed of contributions from the frontend sampler and the output buffer. The intermediary switches are designed to settle completely in each of the phases and therefore do not degrade the distortion performance. For the same reason, the sampled noise of the entire circuit consists of settled and uncorrelated noise samples from the sampler as well as white noise from the output buffer.

5.1.2 Model Description

Since all nodes in the track and hold are unidirectional, one intuitive partition is to have one module for the sampler and one module for the output buffer. Both modules can be categorized as analog in and out modules, and will be discussed in succession.

In the sampler module, it makes sense to separate the sampling function from the level-shifting function performed by switch p2. Since by design, p2 is sized properly to allow signals to fully settle, the level-shifting sub-module is simply an all-pass filter that copies the sampled voltage to the buffer input with a common mode shift from 0.7V to 0.35V.

The main function of the sampler module is the sampling operation, which is controlled by the sampling clock p1. When p1e falls, the switch connecting the capacitor to 0.35V turns off and the output of the sampler no long changes. For this case, nothing needs to be done by the model. When p1 is high, the switch together with C_s forms an RC filter. The output computation of a single pole system was discussed in detail in Section 3.3 and the outline of an A-to-A module was presented in Section 4.1. The distinguishing feature here is that the on-resistance of the switch varies according to the input value and is included in order to introduce distortion into the model. The transistor resistance (r_{ds}) can be approximated as drain-source voltage (v_{ds}) divided by drain-source current (i_{ds}), where i_{ds} is a function of gate-source voltage (v_{gs}) and v_{ds}. During output computation the variable R in Figure 17 is replaced with the value of $r_{ds} = v_{ds}/i_{ds}$. Figure 28 plots the 2-D function for i_{ds} from SPICE simulation. Since the sampler is bootstrapped, v_{gs} is close to the supply voltage, and the range of v_{ds} that could develop across the NMOS sampler is restricted to the input swing of 0.8V. Therefore, only a part of the surface in Figure 28 is needed to fit i_{ds} as a function of v_{gs} and v_{ds}. A second order polynomial model is assumed and the resulting fit is shown in Figure 29 with the goodness of the fit annotated.

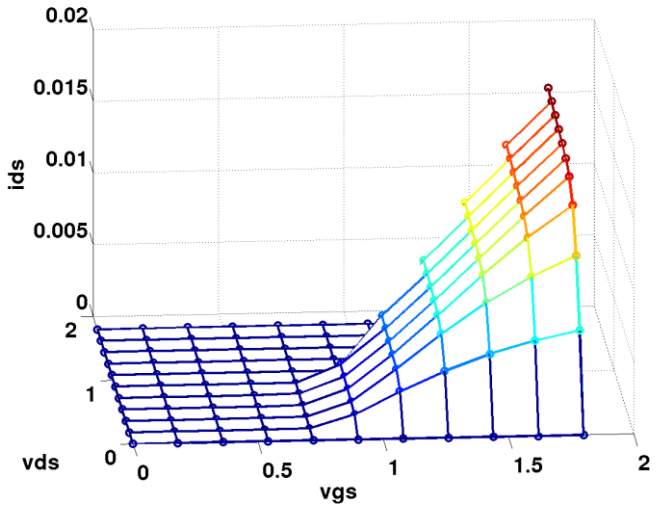

Figure 28 – Sampling transistor current vs v_{gs} and v_{ds}

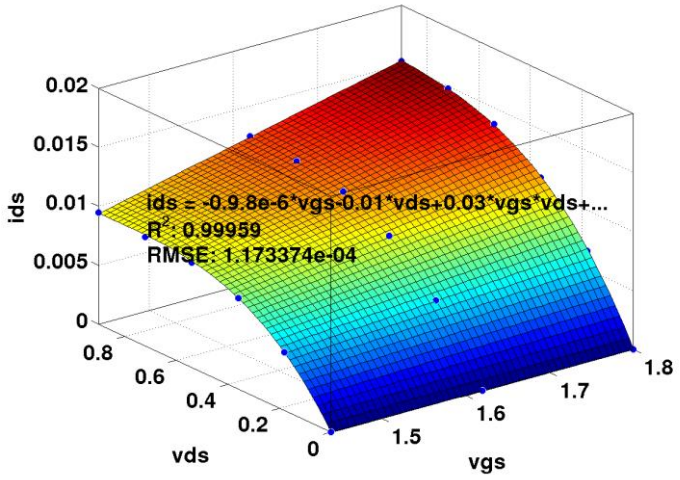

Figure 29 – Sampling transistor current extracted as function of v_{gs} and v_{ds}

In order to obtain the correct value of v_{gs} for the computation of r_{ds} in real time, a helper function is created within the sampler module that mimics the bootstrapping operation. Ideally, the gate voltage boosting capacitor should maintain v_{gs} at the supply voltage when the sampling clock p1 is high, however due to effects such as charge sharing, the bootstrapping is not perfect. SPICE simulation was performed to yield (16)

as the equation that results in the correct v_{gs}. The R^2 value of the fit is 1 and the RMSE is 0.14mV.

$$v_g = 0.9256 v_{in} + 1.511 \tag{16}$$

The finite fall time of the gating signal on switch p1 is another potential source of non-linearity in the T/H. As Figure 30 illustrates, a transistor turns off when its gate to source voltage falls below the transistor threshold voltage, Vth. When the fall time of the gating signal is slow compared to the input signal, the actual sample time will be different from the ideal sample time.

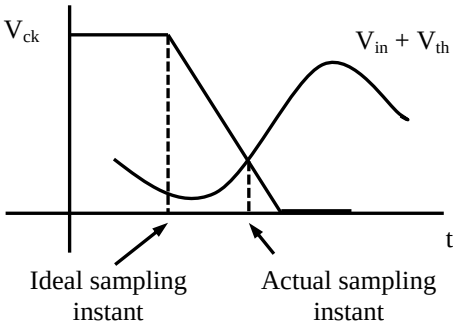

Figure 30 – Signal dependent sampling instant

In this design, care was taken to make the fall time 50ps so that the distortion coming from the non-ideal sample time is insignificant, and therefore this nonlinearity is not modeled. If, however, this effect were to be modeled, the approach would be to insert a sampling clock adjustment module inside the sampler. This module would be composed of a D-to-A module cascaded with an A-to-D module. The D-to-A module attaches the simulated rise and fall times to the digital sampling clock signal, making it an analog signal. The A-to-D module then computes when the difference between the gating signal and the input signal falls below V_{th} and toggles a digital signal. Finally, the sampler will take this modified digital signal as its gating signal.

The output buffer is the second A-to-A module in the T/H, whose transfer function is approximately a single-pole system with close to unity gain. The exact parameters of the transfer function are obtained from SPICE simulation. The model construction is the same as the sampler during tracking phase, i.e. a filter module. This bare-bones transfer function models the portion of the buffer distortion due to the buffer's limited ability to settle to the correct value each sampling cycle. A second source of distortion modeled here is the input dependent gain of the buffer. To incorporate this non-ideality, the equation in Figure 17 for a single-pole system is modified to the following:

$$V_{out} = g \times \left[\beta t + \beta RC\left(e^{-\frac{t}{RC}} - 1\right) + \alpha(1 - e^{-\frac{t}{RC}})\right] + V_0 e^{-\frac{t}{RC}} \quad (17)$$

where g is the DC gain of the source follower as a function of its input amplitude and is evaluated for every output. The form and coefficients of this function is determined from SPICE simulation and sketched out in Figure 31.

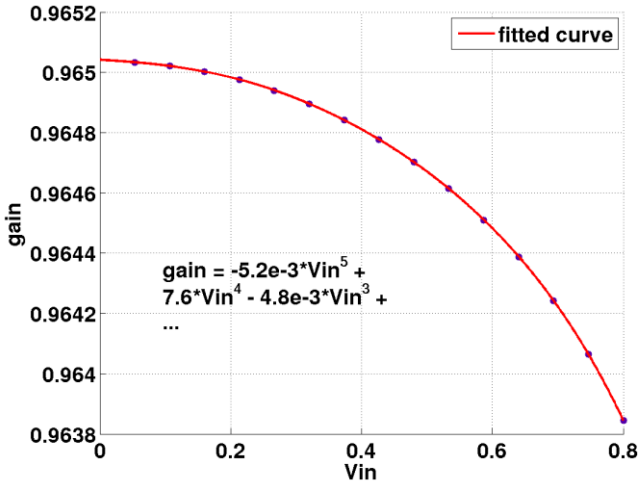

Figure 31 – Buffer gain extracted as a function of input amplitude

Lastly, the total integrated noise voltage at the output of the track and hold is simulated in Spectre to be 136.3uVrms (Figure 32 shows the output noise power

spectrum from 125Hz to 125MHz). Since the noisy samples are uncorrelated, this value is used directly as the standard deviation of a Gaussian random variable that is added to the buffer's output.

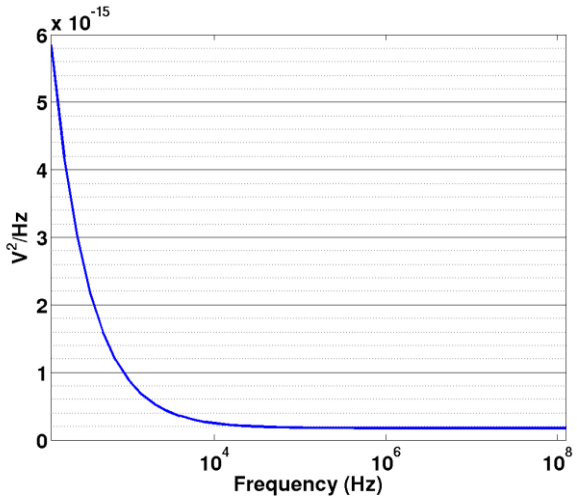

Figure 32 – Track and hold total output noise power spectrum

5.1.3 Simulation Results

To test the model for distortion performance, a sinusoid of amplitude 0.8V and frequency 2045/4096*f_s (where f_s is 250MHz) is sent to the model. Figure 33 shows a selection of transient waveforms. The top strip is the input waveform. The piecewise constant appearance is a display artifact; the slope value can be seen as the input PWL structure is expanded below the waveform. The effective sampling clock and the output of the sampler are shown next. The sampler is seen tracking the input signal when the sampling clock is high, and holding its output when the sampling clock falls. Then, switch p2 is seen moving the common mode of the sampled signal for the input of the 1x buffer. Lastly, the buffer output is shown in the second last strip and the clock in the last strip is used to sample this output to determine the distortion performance of the model.

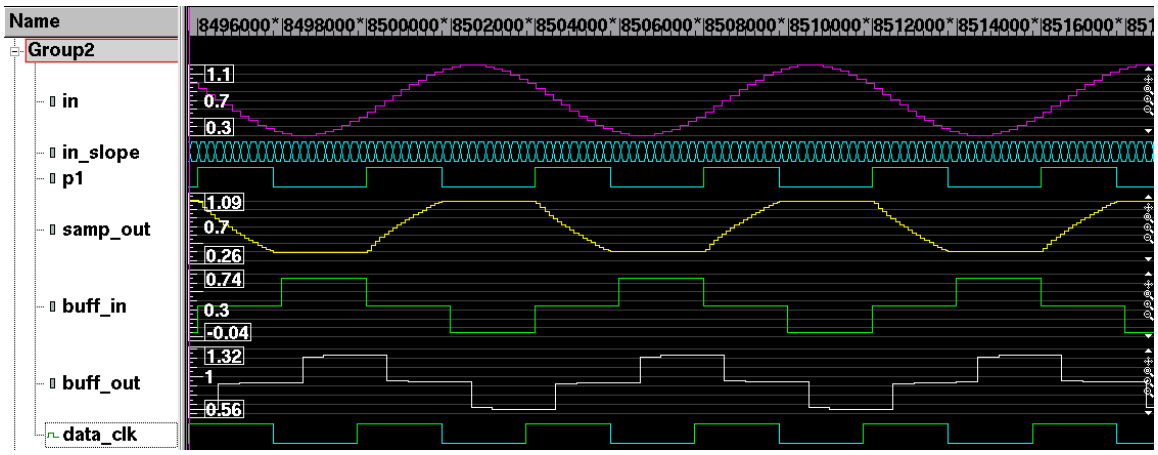

Figure 33 – Transient waveforms of T/H model

Figure 34 plots the 4096-point FFT from Spectre simulation and shows an SFDR of 61.7dB.

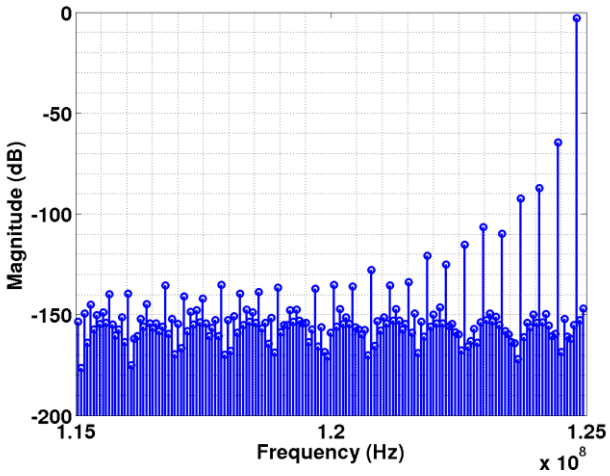

Figure 34 – T/H output spectrum (Spectre)

In SystemVerilog simulation, it is possible to turn on all the second order effects described in the previous section one by one; Table 3 breaks down the contributions of each to total nonlinearity. The magnitudes of the signal tone as well as that of the 3rd, 5th and 7th harmonics are listed. The even order harmonics are suppressed due to the

differential nature of the track and hold. The reference model (first row of Table 3) is set by a completely linear model – nominal sampler resistance independent of input voltage and no gain compression from the output buffer. Figure 35 plots the FFT result of this linear model. As expected, there is no distortion. Moving down Table 3, it can be concluded that the bootstrapping scheme works well in reducing nonlinearity by almost removing signal dependent v_{gs} variation; the remaining distortion from the sampler originating from input dependent v_{th} is captured by accurately modeling r_{ds} as a function of all three transistor terminal voltages; and the gain compression of the output buffer adds approximately the same amount of distortion to the whole system as the sampler alone.

Table 3 – Contributions to distortion in track and hold

	Fundamental	**3rd**	**5th**	**7th**	**SFDR**
Linear model	-2.37dB	-148.06dB	-147.91dB	-149.01dB	145.69dB
+ v_{gs} variation	-2.36	-78.84dB	-96.62dB	-107.32dB	76.48dB
+ v_{th}/v_{ds} variation	-2.40	-68.73dB	-86.26dB	-96.69dB	66.32dB
+ buffer distortion	-2.41	-65.33dB	-83.51dB	-94.97dB	62.92dB

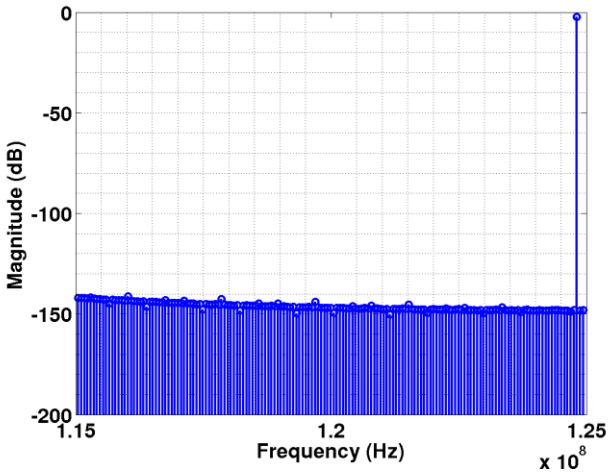

Figure 35 – T/H output spectrum (SystemVerilog without distortion effects)

Adding noise to the model containing all the non-idealities produces the FFT plot shown in Figure 36 with an SFDR of 62.8dB. Summing up all the noise bins in the figure yields a total noise voltage of 135.4uV$_{rms}$ (compared to 136.3uV$_{rms}$ from Spectre plot in Figure 32). Table 4 lists the tones and SFDR obtained from this model and that of Spectre simulation.

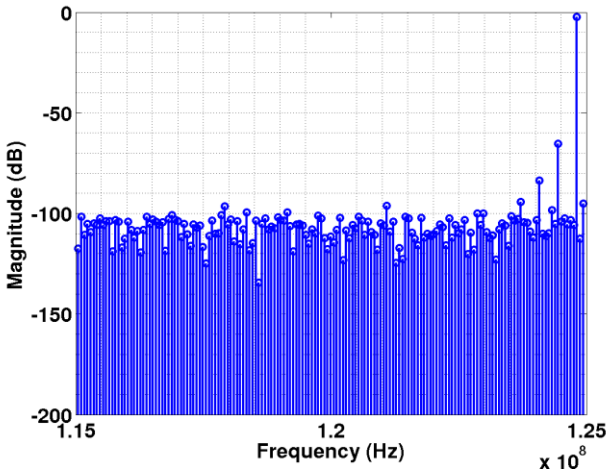

Figure 36 – T/H output spectrum (SystemVerilog with distortion and noise)

Table 4 – Output tones of track and hold

Harmonic	**SPICE**	**SystemVerilog**
Fundamental	-2.86dB	-2.41dB
3rd	-64.58dB	-65.3dB
5th	-87.14dB	-83.56dB
7th	-92.29dB	-94.24dB
SFDR	61.7dB	62.8dB

5.2 DC-DC Switching Regulator

DC-DC switching regulators are indispensable to many mixed-signal SoC's. In fact, several regulators may be needed in most cases [94]. One of the critical performance metrics is, of course, the regulated steady state output voltage. Other details that could be of interest include the startup and settling behavior and the response of the regulator to a change in its load. This section models a buck converter with a novel digital control loop [94].

5.2.1 Circuit Description

The block diagram of the buck converter is shown in Figure 37.

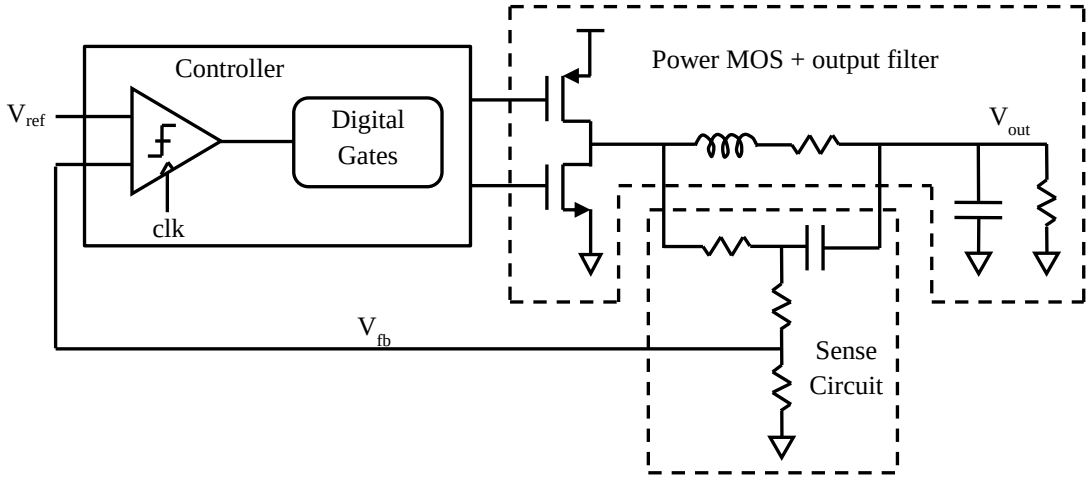

Figure 37 – Buck converter block diagram

This regulator outputs 1.8V from a 2.5V external supply. The range of current draw supported is 2mA to 100mA. Intuitively speaking, the feedback loop creates a duty-cycled waveform that controls the power MOSFETs with the correct average voltage. It is important to never switch on both the P and N MOS at the same time, in order to avoid excessive short circuit current. The inductor and capacitor network connected to the

output then act as a filter to remove the high frequency content of the switching operation and provides just the average voltage to the load.

The feedback loop that regulates the average voltage of the switching waveforms includes a sense circuit that monitors the instantaneous output voltage and a compensation or control logic block that tunes the on-off ratios of the power MOSFETs such that the observed instantaneous voltage is closely matched to a desired reference voltage. In the particular implementation discussed here, the sense circuit approximates the ripple current through the inductor by using an RC filter across the inductor, which then passes through voltage divider. Due to the divide ratio, the reference voltage used is 1.2V so that the output will be regulated at 1.8V.

The control logic employed here is of the constant-off-time variety to take advantage of the mostly digital nature of this scheme. A clocked comparator determines whether the sensed voltage is above or below the 1.2V reference. The power MOSFET gating signals are determined from the following Boolean expression:

$$\mathrm{NMOS_{on}} <= (V_{out} > V_{ref}) + NMOS_{on} \cdot V_{GateP,\ delayed} \tag{18}$$

The NMOSFET will be turned on if at any clock cycle the sensed voltage is greater than 1.2V and will remain on for a fixed amount of time (~2.6ns) as determined by the delay of a delay line. For the rest of the time, the PMOSFET will be turned on. With this scheme, the frequency of one switching period of the power MOSFETs (and in effect, the on-off time ratio) is adjusted constantly to match the output voltage to the desired 1.8V. A small modification to the output filter network is necessary to aid the comparison process. Since the ripple voltage across the inductor is, by design, very small during steady state operation, it may not be enough to trigger the clocked comparator. Therefore, a resistor of 0.5 ohm is added in series with the parasitic resistance of the inductor to synthetically create a larger ripple.

5.2.2 Model Description

The power MOSFETs and LC output filter form a single digital input, analog output module. As described in Section 4.4, a D-to-A module consists of an ideal conversion module followed by a filter. Here, the converter frontend combines the P and N gating signals into a single signal and delays it according to simulated SPICE results since the power MOSFETs are large and require a chain of drivers. Also, an assertion to check that the P and N FETs are not turned on simultaneously is important and is included in the frontend. The filter-like portion of this D-to-A module then models the second order system formed by the inductor and capacitor. Typically, the component values (L, C, R's) are substituted with fixed numbers during each simulation run; however, if the load resistor needs to change in real time, its value can be left as a variable in the equation and changed when necessary.

The sense circuit, by design, does not interact much with the output filter, and therefore is treated as a separate block here. The transfer function of this A-to-A module reflects the high pass filter and resistive divider formed by three resistors and a capacitor. In fact, a single scaled down high pass filter transfer function is used in the model.

Much of the feedback control loop is implemented in digital gates. The delays of these gates affect the response time of the converter and any ripple that may be of interest on the regulated output. In order to better capture these characteristics, the compressed Boolean expression (18) is not used; instead, the actual gates are coded with their respective delays extracted from SPICE simulation. These are D-to-D modules that look similar to the example given in Section 4.2.

The clocked comparator is the other critical component in the control loop and its delay is, hence, also important. As mentioned in Section 4.3, a comparator can be categorized as an A-to-D circuit. The filter frontend, in this particular case, models a first-order system represented by the continuous-time pre-amplifier in the comparator

implementation. The pre-amplifier amplifies the difference between the sense circuit output and a reference voltage; then, a cross-coupled latch clocked at 2.9GHz makes a decision on the sign of the difference. The slicer in the model, therefore, embodies this latch. Since delay is the critical property to be captured and the regeneration rate of the latch varies according to its input amplitude, a delay vs. input plot is obtained from Spectre simulation (Figure 38). The resulting curve is fitted to a power function (annotated on the same graph) and incorporated into the slicer backend of the comparator model.

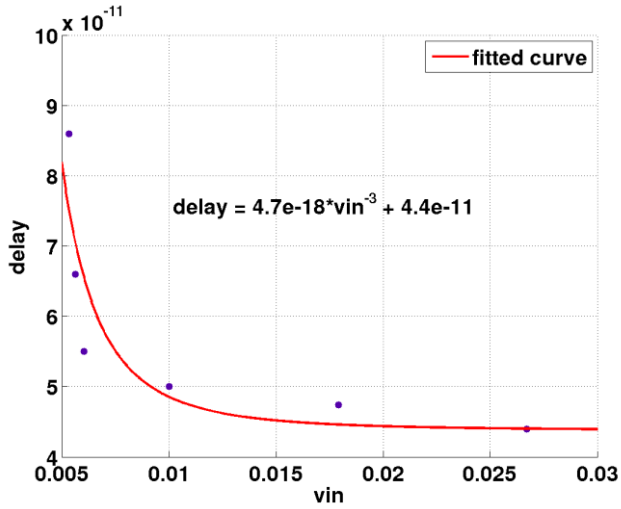

Figure 38 – Clocked latch delay extracted as a function of input amplitude

5.2.3 Simulation Results

Transient simulations are run to measure the dc-dc converter's performance. First, signals at various points in the feedback loop are displayed in Figure 39 to demonstrate the model's correct functionality. The topmost strip shows the steady state output voltage of the converter. Below it is the sensed inductor voltage, which is provided to the comparator for comparison. The next two signals are the clock signal that drives the comparator and the comparator output. When the regulated voltage is too high, the

comparator outputs 1 and vice versa. The final two waveforms are the power MOSFET gating signals resulting from the comparator's decision. These two signals complete the loop by turning off the PMOS (and turning on the NMOS) for a fixed amount of time if the instantaneous converter output is too high, and turning back on the PMOS when the regulated voltage dips below the 1.8V target. To be more exact, the feedback voltage is a superposition of the average voltage at the power MOSFET switch node and the ripple voltage across the inductor. Since the ripple voltage is synthetically amplified by the added series resistance, the feedback voltage reaches 1.2V slightly earlier than the output voltage reading 1.8V, and therefore the final regulated output is lower than 1.8V (1.75V as shown in Figure 39) .

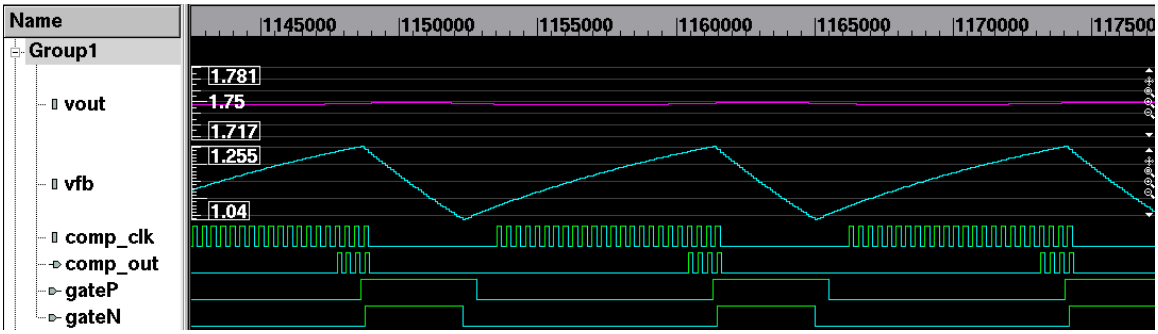

Figure 39 – Transient waveforms of buck converter model

The next two graphs (Figure 40 and Figure 41) compare the startup behavior of the converter. The loading condition shown is 20Ω. The model achieves similar settling time (roughly 800ns) and regulated voltage (1.75V) as Spectre results. The sensed and feedback voltage is also a good match, exhibiting high pass behavior in the various voltage jumps and allowing an amplified ripple voltage to pass through in steady state operation.

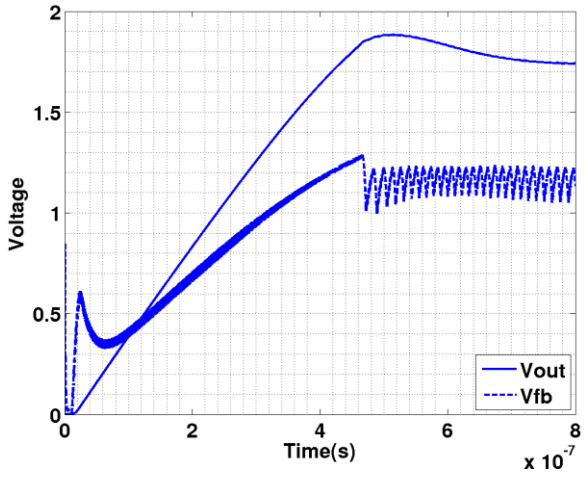

Figure 40 – Startup behavior of buck converter (Spectre)

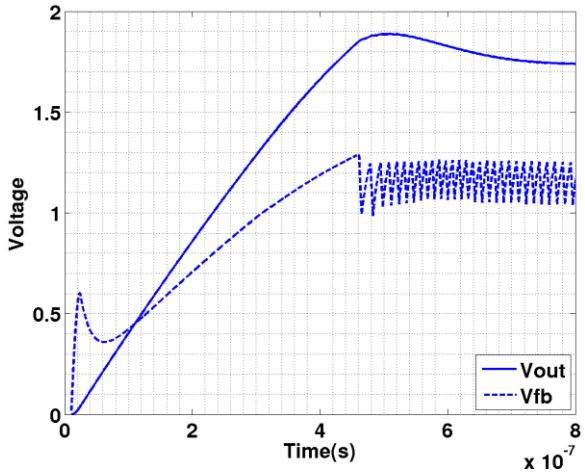

Figure 41 – Startup behavior of buck converter (SystemVerilog)

The load response of the converter is verified in Figure 42 and Figure 43. In this transient simulation, the converter is allowed to reach steady state with a 20Ω load for 1.5us, and then the load is changed to 10Ω. The two figures plot the converter response immediately following the load change. Both the model and circuit netlist simulations reveal a maximum change of about 20mV on the regulated output.

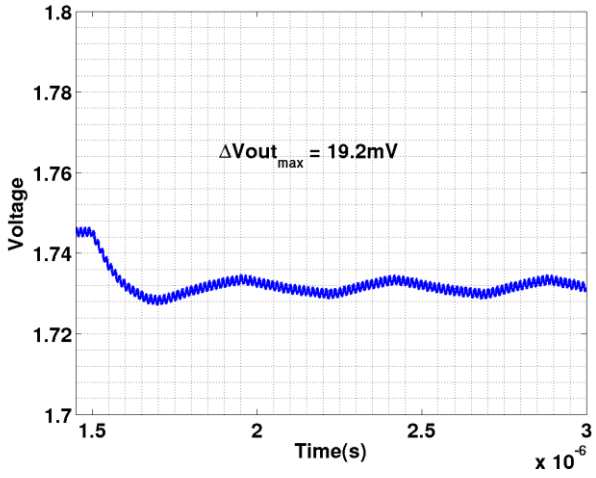

Figure 42 – Buck converter load response (Spectre)

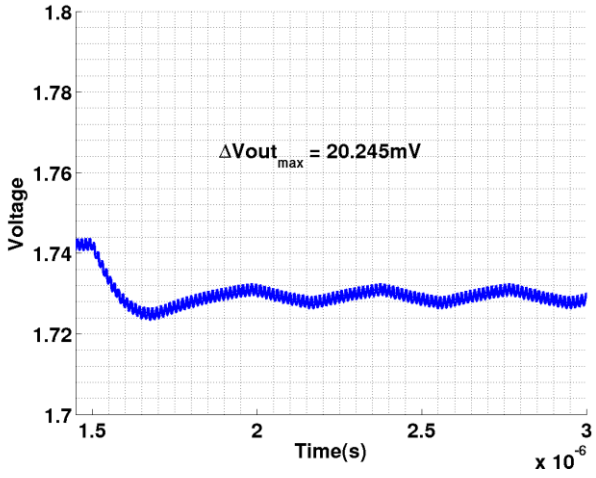

Figure 43 – Buck converter load response (SystemVerilog)

5.3 Phase Locked Loop

The phase locked loop is a clocking system that uses feedback to lock the phase of the output clock to an input clock. By dividing down the output clock, one creates frequency synthesizers useful for many TV [90] and wireless systems [92]. Simpler clock

multiplication ratios are used in digital systems to generate a stable clock for multicore processors and memories [91]. The benefit derived from using a PLL instead of just the input clock include jitter reduction and skew suppression [93]. A PLL's locking behavior, static phase offset, and jitter/phase noise performance are some of the prominently featured metrics during system level integration and hence validation. As this section will show, the techniques developed in Chapters 3 and 4 are capable of modeling these behaviors.

5.3.1 Circuit Description

The circuit modeled in this section is a simple clock multiplication PLL shown in Figure 44 that would be appropriate for high-frequency digital systems. The reference frequency is set to 500MHz, while the PLL outputs a 1GHz clock. Therefore, the divider ratio is 2. The phase frequency detector (PFD) generates up and down pulses proportional to the phase difference between the divided down VCO clock and the reference. These pulses then become gating signals of the charge pump (CP), which outputs proportional current. This current is integrated onto the low pass filter (LPF), which in essence reduces the ripple on the PFD pulses and extracts the average value. Since the PFD pulses occur at the reference frequency of 500MHz, the bandwidth of the loop filter must be much lower (121MHz in this case) in order to achieve the desired filtering. The resulting average value then controls the frequency of the VCO. The VCO is CMOS ring-based and its frequency is tuned through modulating the ring's supply voltage. A regulator is employed to buffer the LPF output voltage and provide enough current for the VCO. Due to this arrangement, the VCO output will have limited swing. Therefore, a buffer (VCOBUF) is necessary to increase the swing to full CMOS rail. Finally, the overall feedback mechanism ensures that the buffered VCO output edges are aligned to those of the reference clock.

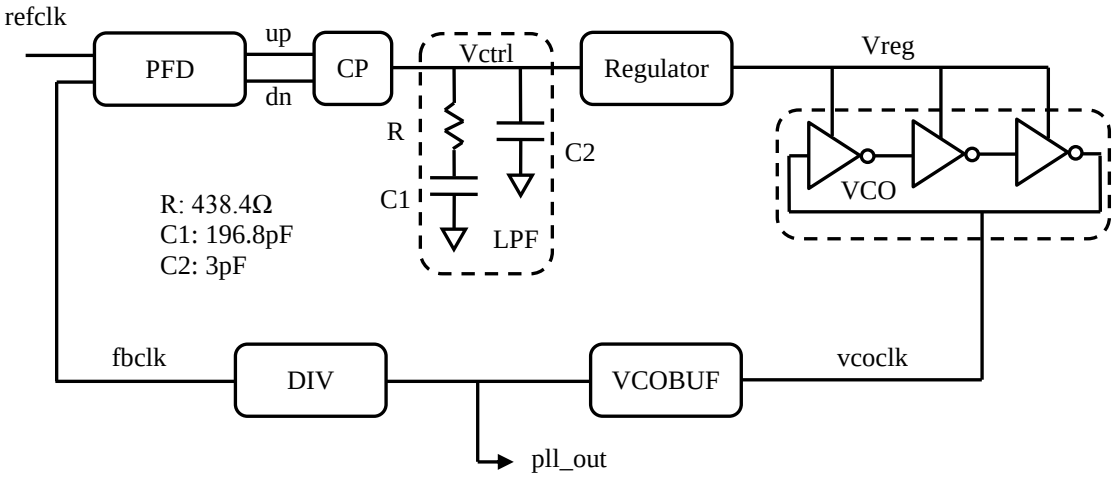

Figure 44 – PLL block diagram

This type of circuit is generally modeled in the phase domain, and the resulting model is shown in Figure 45. Note that this linear model as a whole is not relied upon when writing the SystemVerilog model; instead, the PLL will be modeled as several separate modules and the closed-loop behavior will naturally result from the connection of these modules. This brief discussion of the linear model is presented here to provide background for the jitter/phase noise simulation results in Section 5.3.3, since that is a very important metric for a PLL design.

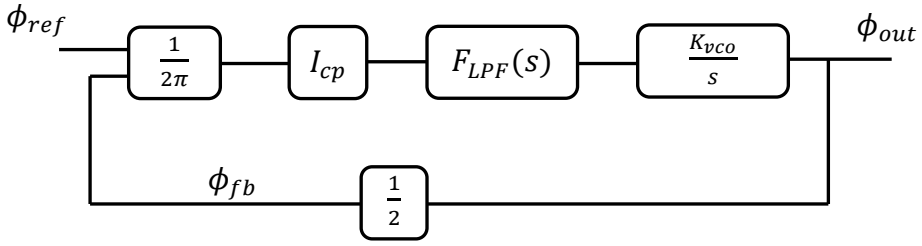

Figure 45 – Linearized PLL model

From classical feedback theory, the closed loop transfer function from reference phase modulation to PLL output phase deviation can be written as:

$$\frac{\phi_{out}}{\phi_{ref}} = \frac{2L(s)}{1 + L(s)} \tag{19}$$

$$L(s) = \frac{1}{2}\frac{I_{cp}}{2\pi} F_{LPF}(s) \frac{K_{vco}}{s}$$

where L is the loop gain. Figure 46 plots the transfer function for this specific PLL implementation and indicates a bandwidth of approximately 15MHz with peaking at 5MHz.

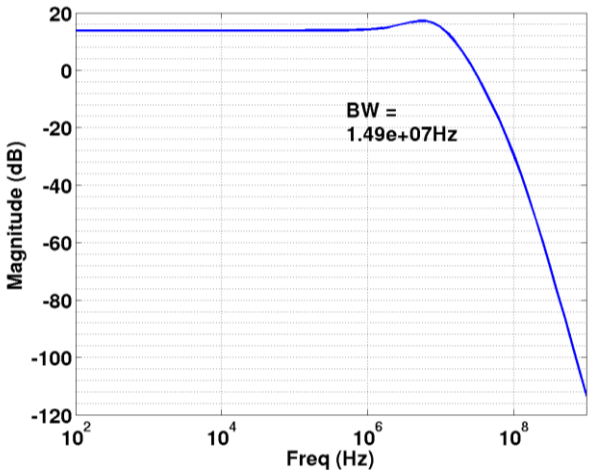

Figure 46 – Closed loop PLL transfer function

The phase noise transfer functions for various sub-blocks of the PLL can be derived as follows: noise from the PFD and divider are low pass filtered by the PLL; noise from the VCO and VCO buffer are high pass filtered. The total phase noise at the PLL output is the addition of all the noise sources, and has a power spectral density (PSD) similar to Figure 47. Reference noise dominates at low frequencies (not modeled in SystemVerilog); PFD, CP and divider noise at mid frequencies; and the VCO at high frequencies beyond the bandwidth of the PLL. Most notably, the PSD is expected to have a peak around the peaking frequency and bandwidth of the closed loop transfer function due to VCO noise.

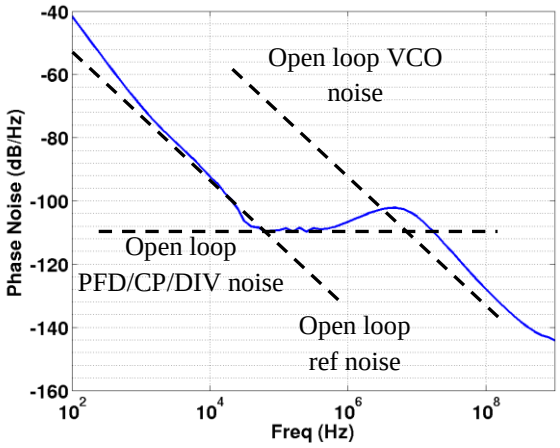

Figure 47 – Typical PLL phase noise plot

5.3.2 Model Description

The diagram in Figure 44 provides a good starting point for the partitioning and categorization of blocks in the PLL. A non-unidirectional node exists in the loop at Vctrl. The charge pump injects currents at this node, and this current is integrated onto the loop filter, resulting in a control voltage for the VCO on the same node. Therefore, the charge pump and the loop filter must be combined to form a single block. This combined block has two digital inputs (up and dn) and a single voltage output, making it a D-to-A module. The categorization of other blocks in the PLL follows from the signal type designation for each of their inputs and outputs. All the clock signals (ref, fbclk, divclk and vcoclk) are digital types. The up and dn pulses are also digital. Vctrl and Vreg are analog signals. Thus, the PFD, VCOBUF and DIV blocks are D-to-D circuits, while the regulator is an A-to-A circuit, and the VCO is an A-to-D circuit. Each block will be discussed in more detail below.

The construction of the charge pump and low pass filter model is as described in Section 4.4: a digital to PWL converter followed by an A-to-A filter. The frontend converter is responsible for mapping the PFD output pulses into the correct PWL current

output. A few non-idealities are dealt with here. First, the up and dn pulses undergo slightly different delays before being turned into currents. From Spectre simulation, these two delays are extracted as functions of the supply voltage (a range of 0.8V-1.3V is used for the fit). After the delays, the digital pulses are converted to PWL structure. The on currents of the P and N sources are somewhat mismatched (103uA vs. 70.8uA as observed in Spectre simulation), and they vary according to supply voltage as well as the drain voltage of the transistors. Again, these non-ideal effects are distilled into closed-form equations and the value element of the PWL structure takes on a number computed from these equations. The converter frontend can also be setup to include noise of the charge pump. A total rms noise current of 1.88nA is obtained from Spectre simulation and used as the standard deviation of a Gaussian random variable that is added to the PWL current structure.

The A-to-A filter backend of this D-to-A module transforms the PWL current into a PWL voltage, Vctrl. The transfer function of the filter component is equivalently the impedance of the loop filter. Noise of the LPF originates from the resistor in the network and its effect is added to Vctrl by constructing an internal A-to-A block whose transfer function is the noise transfer function of the resistor to the LPF output, and whose input is the flat noise voltage of $\sqrt{4kTR}$.

The regulator is a block with analog input and output. Its output is modeled as the sum of two transfer functions: one from the input and the other from its supply. SPICE simulations show that the input to output transfer function is not affected very much when the supply is varied between +/- 10%. Therefore, it is acceptable to take the supply as a second input to the regulator and add the response to both inputs at the output. If the two transfer functions affected each other, the model template can be changed to select between several versions of one transfer function according to the value of the parameters that affects it. Noise performance is also extracted from Spectre (467.44uV_{rms} of total

integrated noise) and this result is added to the model using random number generators available in SystemVerilog as before.

The VCO is an analog to digital block that operates nearly linearly in the phase domain. The filter portion of the VCO model is an integrator with phase as its state variable and

$$\Delta\phi = \int K_{vco} V_{ctrl}(t) dt \tag{20}$$

as the equation that governs the evolution of this state variable (in other words, its transfer function is proportional to 1/s). The slicer backend determines when the phase crosses π and schedules the output to be toggled at that time. The effect of substrate disturbances is integrated into the model by fitting the VCO frequency as a function of both the control voltage and the substrate voltage, so that Kvco in equation (20) becomes the sum of Kvctrl and Kvsub. Jitter is also modeled. A VCO exhibits accumulating jitter since each phase disturbance is permanent and the phase error add up over time [98]. Therefore, at each state (i.e. phase) evaluation point, the phase noise accumulated since the last evaluation point is added to the incremental phase resulting from the state equation alone.

The remaining three blocks (PFD, VCO buffer and divider) are all D-to-D modules. The delays through these blocks are carefully characterized through Spectre simulation and fitted into functions of supply voltage over a range of 0.8V-1.3V. Time-domain noise analysis is also done in Spectre to obtain the jitter performance and these results are used as the standard deviations for Gaussian random number generators that add small deviations to the nominal delays.

5.3.3 Simulation Results

Transient waveforms for several signals in the PLL loop are shown in Figure 48 to illustrate the circuit's operation in steady state.

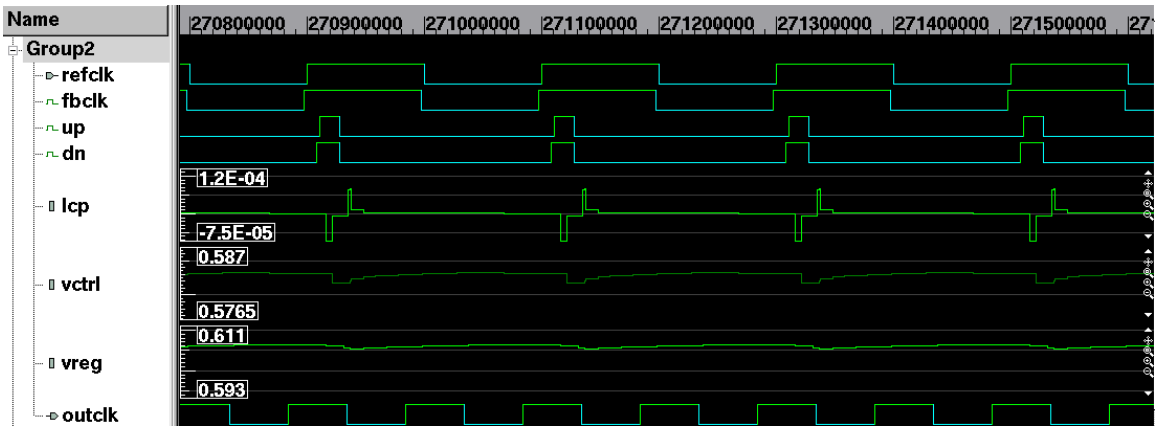

Figure 48 – Transient waveforms of PLL model

The first two strips are the reference clock and the divided down PLL clock, respectively. The divided clock (and hence the VCO clock) can be seen to be locked to the reference with some static phase offset. The next two strips are the up and dn pulses from the PFD. The dn pulse is wider since the PLL clock is slightly ahead of the reference. The current pulses due to the PFD output are shown next. The P and N MOS currents are mismatched, but over each reference clock cycle, the total charge injected into the LPF is 0. Vctrl and Vreg are shown next. The slight ripple is due to the staggered turn on of the P and N MOS currents in the charge pump. The VCO buffer output is the final signal shown here, and it has twice the frequency of the divided down clock in the second strip.

Figure 49 and Figure 50 compare the model locking behavior with that of circuit schematic. Both indicate a locking acquisition time of around 2us. Note that the plotted voltage, Vreg, is the output of the regulator to the input of the VCO (as opposed to Vctrl which is the output voltage of the charge pump and low pass filter going into the

regulator). The slight discrepancy near the beginning of Vreg is due to the regulator being offline for Vctrl less than 0.2V, and this behavior was not included in the model. In addition, a static phase offset of 33ps is measured in SystemVerilog and 34.5ps in Spectre.

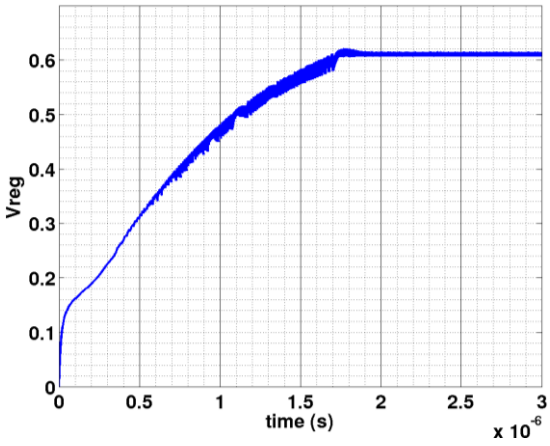

Figure 49 – PLL locking behavior (Spectre)

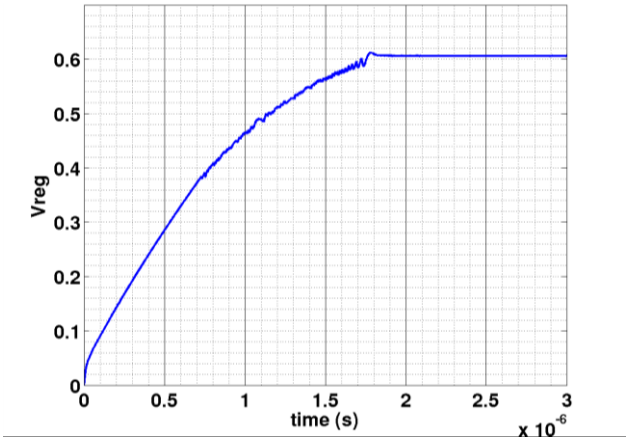

Figure 50 – PLL locking behavior (SystemVerilog)

Next, the noise/jitter properties of the PLL sub-blocks are turned on one by one in the model and their respective contribution to output jitter is listed in Table 5. The VCO is by far the most significant jitter source. Since all the noise contributions are relatively independent of each other, a total jitter of 7.9ps can be expected by adding all the rows in

an rms fashion. This total is essentially from the VCO. Eliminating all the other sources changes the total jitter by 0.6%. Then, with all the noise sources turned on in the model, and plotting 8000 clock edges, the jitter histogram shown in Figure 51 results. The 8ps of total rms jitter achieved matches well with the expected value from Table 5. In addition, a jitter of 7.78ps calculated from Spectre phase noise simulation further confirms the match.

Table 5 – PLL sub-block contribution to output jitter (SystemVerilog)

Noise Source	**PLL Output Jitter**
PFD	0.556ps
CP	0.126ps
LPF	0.124ps
Regulator	0.612ps
VCO	7.85ps
VCO Buffer	0.394ps
Divider	0.126ps

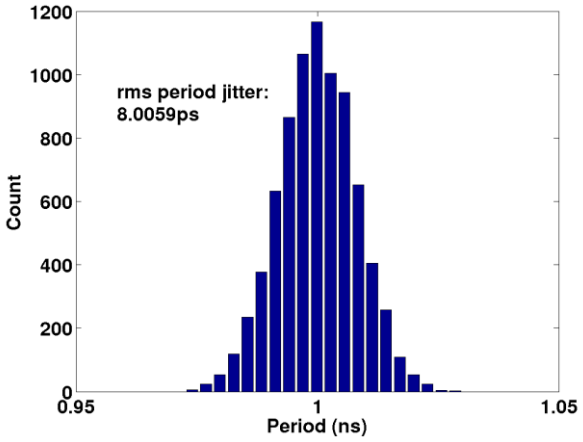

Figure 51 – PLL output jitter histogram (SystemVerilog)

Figure 52 is a phase noise plot constructed from 430k output edges generated by the model. Welch's method [100] with an FFT size of 4096 and overlap of 2048 is utilized. As expected, the phase noise peaks around the PLL closed-loop peaking frequency of 5MHz due to the high pass nature of the VCO noise transfer function.

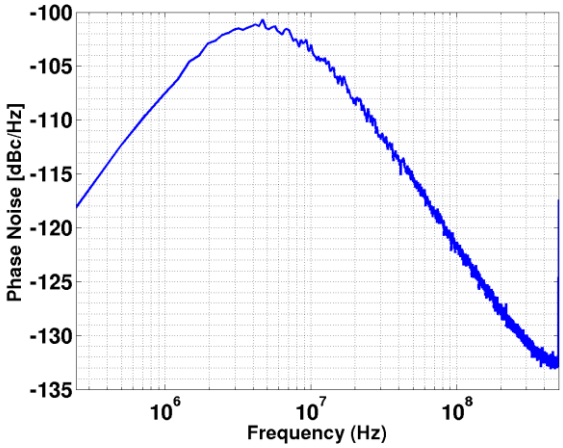

Figure 52 – PLL output phase noise spectrum (SystemVerilog)

To verify the modeling of supply effects, voltage steps are applied to the PLL supply. The green line in Figure 53 illustrates the steps of +100mV from nominal at 3us and -100mV from nominal at 4us. The transient output clock edge offset from the reference vs. time is plotted in the same graph. Again, the model results and Spectre simulation show good correlation.

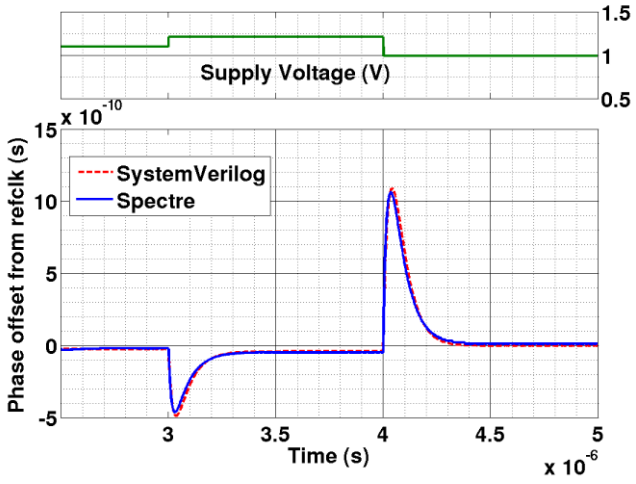

Figure 53 – PLL output phase response to supply disturbances

5.4 Simulation Speed Analysis

As the above sections have shown, the three models are pin-accurate and their behaviors are closely matched with those of the corresponding circuits simulated in Spectre. A third criterion for behavioral models (recall from Chapter 3) is that they should be able to run quickly in a digital simulator. This section will investigate how well these models perform in this respect.

The simulation speed of a model is proportional to how often the model is evaluated. There are two factors that control the rate of output evaluation: the input signal update rate and the internal re-evaluation rate. The input update rate could be set by a user selected rate for a test signal, or it could be a byproduct of the error tolerance used on the preceding module's output event filter. The internal re-evaluation rate is set as a fraction of the smallest time constant in the system.

In order to determine the reasonable settings for each model, the error tolerance for output filtering will be fixed (1uV for all voltages and 0.1uA for the charge pump

current in the PLL model) while varying the input test signal update rates as well as the internal re-evaluation rates for individual sub-blocks. The variations in these rates will cause small deviations in the measured behavior from the models. Based on the degree of impact, an appropriate set of rates can be chosen for the models. The results for each of the three circuits will be discussed below in succession.

Table 6 enumerates the different rate scenarios (in decreasing order) studied for the track and hold, and Figure 54 shows the corresponding T/H SFDR. Spectral accuracy up to the 7^{th} harmonic (approximately -92dB from signal tone) can be used as a modeling target, to ensure that the 3^{rd} harmonic – equivalently the SFDR – can be extracted correctly from model simulation. With this requirement, settings #6 and #7 from Table 6– input updated every 100ps, sampler re-evaluated every 100ps or 500ps, and output buffer re-evaluated every 500ps – provide the necessary accuracy with comparatively less computation than the other scenarios. This result can be explained intuitively by examining how the T/H works and some of the design parameters used in the circuit.

Table 6 – Internal re-evaluation rates for T/H

Scenario	Input	Sampler	Buffer	# evals	% filtered
1	40ps	100ps	100ps	764.7k	0.006
2	40ps	100ps	500ps	633k	0.001
3	40ps	100ps	1ns	616.5k	0.0008
4	40ps	300ps	500ps	633k	0.0009
5	40ps	500ps	500ps	633k	0.0009
6	100ps	100ps	500ps	275.4k	0.002
7	100ps	500ps	500ps	275.4k	0.002
8	500ps	1ns	1ns	68.26k	0.004

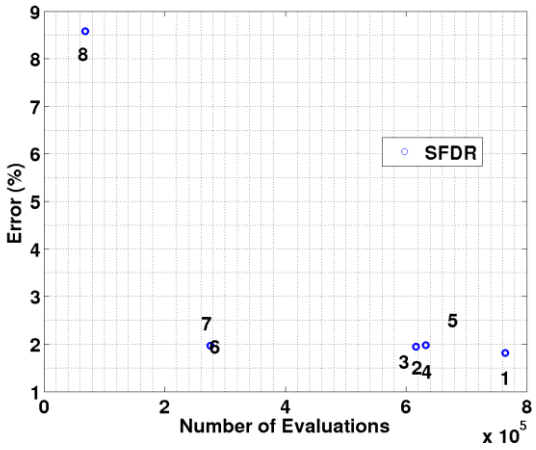

Figure 54 – T/H SFDR error for various re-evaluation rates

Three blocks need to have their evaluation rates set: the input generator, the sampler and the output buffer. The rate of the input signal update can be estimated with the help of Figure 13. Assume that the energy of any spur resulting from representing the input signal as PWL should be less than the 7^{th} harmonic distortion (approximately -92dB). If the input signal were updated every 200ps, Figure 13 indicates that the largest spur will be roughly 65dB below signal tone. The PWL input signal then is filtered by the sampler with a 3dB bandwidth of about 760MHz, and this will provide an additional 16dB of spur suppression. However, this is not sufficient to achieve -92dB. Thus, the input signal is updated every 100ps. Since the input to the sampler is already updated every 100ps (less than one tenth of the sampler's bandwidth), setting the re-evaluation rate for the sampler any higher is unnecessary. As can be observed from Table 6 and Figure 54, using 100ps (scenario #6) or 500ps (scenario #7) for the sampler makes no difference in the number of evaluations or the SFDR result.

The input of the buffer is a staircase signal and only the value the output settles to near the end of the sampling cycle is important. Recall that the buffer is modeled as a single-pole system with a DC gain adjustment proportional to the input amplitude. Since the input doesn't change over the period that the buffer is evaluating, not many re-

evaluation points are needed, as long as the final value is computed correctly. The buffer is given 2ns to settle, and its time constant is 183ps; therefore, if a re-evaluation rate of every 500ps is used, then at the last of the 4 computation points within the 2ns window, the buffer will have had enough time to settle, and it is this output segment that's used for further processing in a larger system. As a result, the input update rate is really what sets the re-evaluation rate for this entire pipeline, and adding any more internal events only increases complexity without benefitting the accuracy.

In the case of the DC-DC converter, three blocks need to set their internal re-evaluation rates: the power MOSFET and output filter, the sense circuit, and the pre-amplifier of the comparator in the feedback control loop. Table 7 lists the different scenarios studied. Figure 55 plots the startup and settling behavior for each scenario. Figure 56 shows the converter's response to a load change for same set of settings. From Spectre simulation, the switching frequency of the converter is approximately 100MHz, and this is an indication that the power MOSFET and output filter should be evaluated at least every 10ns. This intuition is corroborated by scenario #9 which evaluates the power path every 20ns and yields vastly different simulation results than the other settings. Evaluation at such a low rate creates large errors in the linear approximation of the output voltage waveform, which is then given to the comparator in the feedback loop to compute the gating signals for the power MOSFETs. Due to the errors in this approximation, the comparator triggers at incorrect times, causing the duty cycle to be erroneous and hence shifting the regulated voltage. The evaluation rates of the sense circuit and pre-amplifier have much less impact on the overall model behavior because the sense circuit outputs nearly piecewise-linear waveforms (see Figure 39) and computation in excess of what's dictated by the output filter module will give similar results. Overall, scenario #6 – evaluating all modules every 1ns or at one tenth of the nominal switching frequency – seems to provide a good compromise between accuracy and speed for both metrics.

Table 7 – Internal re-evaluation rates for DC-DC converter

Scenario	Power FET /output filter	Sense	Pre-amp	# evals	% filtered
1	100ps	100ps	100ps	96.9k	14.4
2	400ps	100ps	100ps	72.5k	8.7
3	1ns	100ps	100ps	67.9k	8.6
4	1ns	200ps	200ps	36.2k	9.6
5	1ns	400ps	400ps	23k	5.9
6	1ns	2ns	2ns	10.5k	2.3
7	5ns	2ns	2ns	5.47k	2.7
8	10ns	5ns	5ns	3.3k	1.2
9	20ns	20ns	20ns	2.46k	0.08

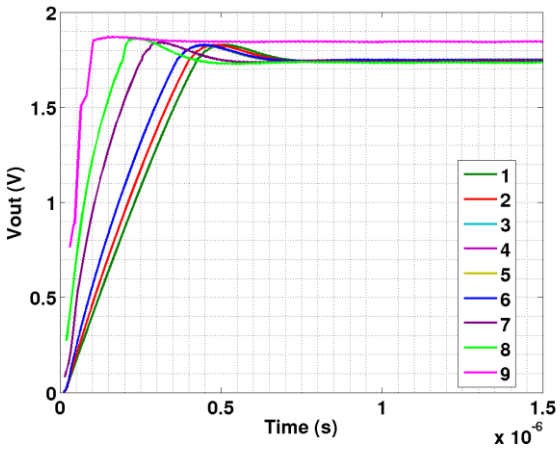

Figure 55 – Buck converter startup behavior for various re-evaluation rates

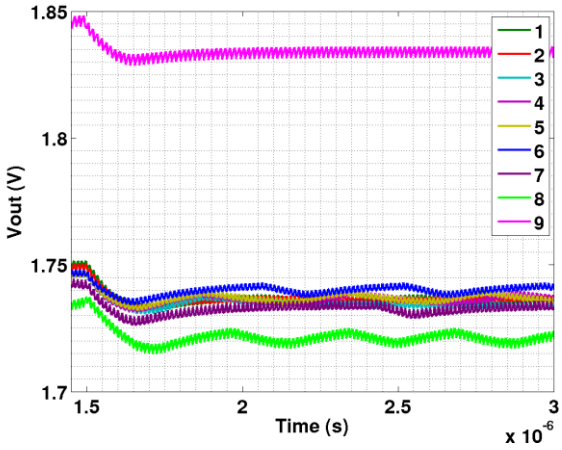

Figure 56 – Buck converter load response for various re-evaluation rates

The re-evaluation rates for the PLL model is lower-bounded by the reference clock frequency much like the T/H example in which the input signal updates set the evaluation points for all downstream blocks. Table 8 documents the CP/LPF and regulator re-evaluation rates investigated. There is no significant impact on PLL steady state behavior across all scenarios. Evaluating all blocks at 1ns (i.e. the PLL output frequency) causes only small deviations in the locking behavior as well as the response to a 180 degree phase step (see Figure 57 and Figure 58). The accuracy of finer second order non-idealities such as jitter begins to fall apart beyond scenario #6 (i.e. when the evaluation rates approach and exceed half the PLL output period). Therefore, setting #6 will be chosen as the balanced choice for the PLL evaluation rates.

Table 8 – Internal re-evaluation rates for PLL

Scenario	CP/LPF	Regulator	# evals	% filtered
1	100ps/100ps	100ps	199.3k	4.3
2	500ps/100ps	100ps	194.1k	3.2
3	4ns/100ps	100ps	182.4k	3.4
4	500ps/500ps	100ps	126k	0.38

5	500ps/1ns	100ps	122.8k	0.29
6	500ps/2ns	100ps	120k	0.3
7	500ps/500ps	500ps	71.8k	0.6
8	1ns/1ns	1ns	56k	0.4

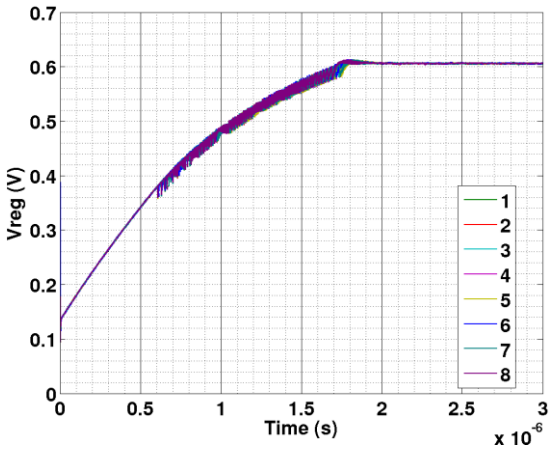

Figure 57 – PLL locking behavior for various re-evaluation rates

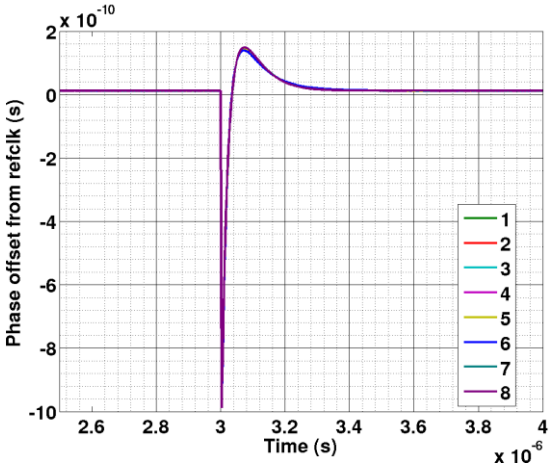

Figure 58 – PLL phase step response for various re-evaluation rates

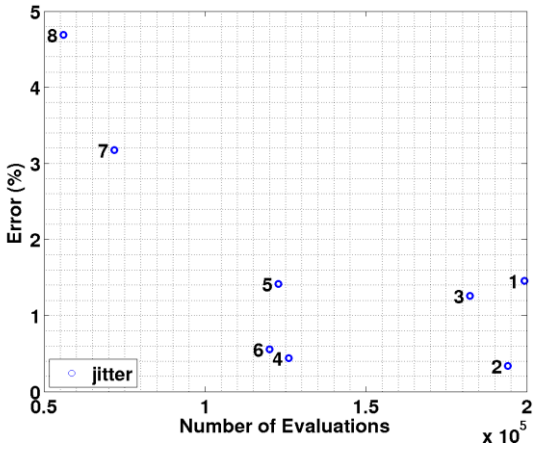

Figure 59 – PLL output jitter error for various re-evaluation rates

So far, the re-evaluation rates of all the analog blocks have been studied and settings that achieve reasonable accuracy can be determined as scenario #7 for the T/H, #6 for the converter and #6 For the PLL. Relaxing the error tolerance at this point will lead to more filtering, but not much reduction in the number of events since the selected rates are now constrained by some fundamental frequency in the circuit that requires computation regardless of the number of output events from the proceeding block – for example, the input update frequency for the T/H, the converter switching frequency partly constrained by a constant-off time, and the PLL reference clock frequency. To gauge the simulation speeds of the models, the aforementioned settings are used without relaxing the error tolerance, and Table 9 lists the model simulation times vs. Spectre. Because the buck converter control loop contains a great deal of digital logic, a fast SPICE simulator (BDA) is used to achieve fairer comparison in speed.

Table 9 – Schematic and model simulation times

Circuit	Transient Time	SPICE	SystemVerilog
Track and Hold	16.4us	42min 18s (Spectre)	1.84s
Buck Converter	3us	25h 15min (Spectre) 1h 38min (BDA)	0.45s
PLL	3us	33min 48s (Spectre)	0.56s
Digital Design (1GHz, 62k gates)	3us	N/A	16.58s

Table 9 also lists a digital design's RTL simulation speed. The gate count is an estimate from a synthesis tool, and roughly speaking a PLL model is equivalent to 2k digital gates. Twenty different digital designs were also tabulated for their simulation times and the equivalent gate count for the PLL model is plotted in Figure 60. The PLL is the equivalent of anywhere between 5340 and 623 gates, with an average of 2115 gates. Note that this is a comparison of simulation speed and is not an indication of the actual complexity of the circuit implementation.

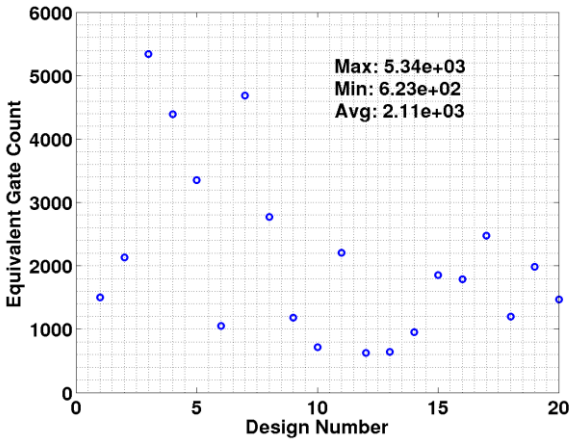

Figure 60 – PLL model equivalent gate count from digital designs

In addition to setting the evaluation rate of a model for a good accuracy and speed tradeoff, it is also interesting to investigate the efficiency of the models in term of CPU time spent on different tasks in a model. This distribution pinpoints any specific task that is dominating simulation time. The processing time of all the A-to-A modules is profiled according to the type of computation performed and the averaged result is shown in Figure 61.

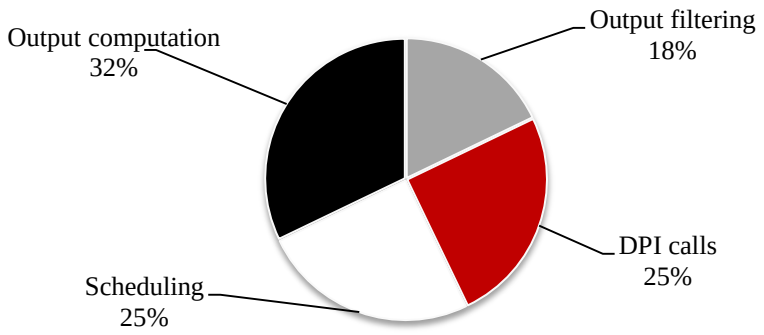

Figure 61 – A-to-A module CPU time distribution

The scheduling of events consumes a quarter of the total simulation time, while the rest is consumed by some form of arithmetic calculation. The calculations are further divided into 32% spent on the computation of output waveform and 18% on deciding whether to update the output (output filtering). The remaining 25% is spent on DPI calls which use external C functions to perform the computation of special functions such as absolute value, sine/cosine and exponentials. This 25% is distributed between the output computation and output filtering operations.

Given the above distribution, one possible way to speed up the simulation of these models is to avoid DPI calls. Custom functions or look up tables have been written and the simulation time needed for 10k calls to each of these functions is listed in Table 10 and compared with DPI call times. The custom functions are roughly 5 times faster. This

indicates that the model simulation times listed previously have a potential to be reduced by 20%.

Table 10 – Simulation time of DPI vs. custom function calls

Function	DPI Call	Custom Function
abs()	1.15s	0.31s
sin()	1.8s	0.34s
exp()	1.53s	0.33s

Overall, the reduction in simulation time from the avoidance of DPI function calls is small compared to selecting the appropriate evaluation rate for each module. If the fastest evaluation rates for each of the composite circuit were blindly used (i.e. scenario #1), then the number of events would be 3x, 10x and 2x more for the T/H, converter and PLL respectively. This translates proportionately to simulation time. Therefore, it is critical to fine tune the evaluation rate for each module according to the particular circuit's characteristic in order to maximize the benefit of the behavioral models when validating mixed-signal SoC's.

5.5 Summary

Behavioral models of three composite circuits are the focus of this chapter. The techniques discussed in Chapter 3 as well as the categorization of circuits described in Chapter 4 are leveraged to create these models. The sub-modules of the track and hold, switching regulator and PLL, together, cover all four types of circuits identified in the previous chapter. Various non-idealities are incorporated into the models to achieve closer resemblance to the circuits' behavior in schematic simulation. The similarity between the obtained data from SystemVerilog and Spectre indicate that performance

metrics such as distortion, startup/settling time and jitter/noise that characterize the circuits can be measured from the models. In addition, the speed analysis section shows several internal re-evaluation rates for the different models and the corresponding model behavior. These results indicate that the internal re-evaluation rates can be set relatively low (and hence benefitting computation speed) according to a fundamental frequency that exists in these circuits. For instance, in the T/H it is the input update rate that determines overall rate of the sampler; in the buck converter, the nominal switching frequency provides a good reference rate; and in the PLL, the reference clock frequency sets a lower bound on all the re-evaluation rates. Furthermore, it is expected that rates somewhat above these fundamental frequencies are needed to measure second order effects such as higher order harmonics in the T/H, transient startup behavior in the converter, and accurate jitter performance in the PLL. With the techniques presented in Chapters 3 and 4, and the insights of this chapter on internal rate selection, behavior models can simulate up to 3 orders of magnitude faster than the circuit netlist, while maintaining critical performance aspects. Based on these results, the three models indeed have demonstrated that they are pin-accurate, fast and able to depict key circuit dynamics, and therefore are suitable for mixed-signal SoC validation.

Chapter 6

Conclusions

Validating today's mixed-signal SoC's is not an easy task because of the large interaction between analog and digital circuits. Due to this tight interface, validation tests need to simulate analog and digital side by side through millions of test vectors, each of which might require ms or more of simulation data. As seen in Chapter 2, traditional tools like SPICE for validating analog circuits and VCS for validating digital circuits cannot be directly used for mixed-signal validation. Among the several solutions reviewed, behavioral modeling has the most chance of succeeding for several reasons. First, it provides the option of sidestepping an analog solver, the use of which inevitably slows down an entire simulation. Second, many modeling languages have built-in algebraic and mathematical operators that can aid in the emulation of analog circuit functions and behaviors. Lastly, the connection to the digital systems of the SoC comes almost naturally for some of the modeling languages so that, indeed, the complete SoC can be simulated.

Behavioral models, however, are currently rather ad hoc, and this thesis provides a framework for creating these models. The approach described in Chapter 3 allows the creation of behavioral models that fit nicely into an event-driven digital environment and remain pin-accurate, fast and faithful to circuit characteristics that designers care about.

There are several key concepts to this approach. First, modules need to be created with unidirectional semantics, which means that analog circuits must be partitioned properly to avoid having non-unidirectional (i.e. current summing) nodes as ports of a module. Next, an appropriate representation of analog signals as a discrete-time sequence of events is necessary in order to fit into the digital simulation framework. Finally, a method of efficiently computing model outputs under the chosen signal representation completes the picture for this modeling approach. As an example, the piecewise linear representation of analog signals is demonstrated along with the corresponding method of output computation, which leverages a (nearly) linear abstraction of analog circuit function as well as the idea of variable domain translation.

These techniques are general guidelines and therefore can be applied to a variety of analog circuits. Chapter 4 illustrated this generality by using the SystemVerilog language to create models of different circuit categories including A-to-A, D-to-D, A-to-D and D-to-A. Among these four categories, the A-to-A models carry slightly more weight because not only are they a category by themselves, they can also be preceded by a digital-to-PWL converter to form a D-to-A model and followed by a slicer to form an A-to-D model. After the models are created, assertions are inserted in order to qualify the usable range of the models and notify the users when inputs exceed the bounds of what was characterized and hence modeled.

Using these techniques on a few real circuits gave promising results. Functional models for a 250MS/s track and hold, a DC-DC switching regulator and a 1GHz PLL all match Spectre-simulated result using many orders of magnitude less time. In conclusion, this dissertation has achieved its goals. By applying the proposed methods, it is possible to create behavioral models that capture important circuit dynamics, maintain pin-accuracy and achieve speeds that are suitable for mixed-signal validation. In addition, two important insights can be gleaned from this experiment. First, a good filter model that’s amenable to asynchronous events is at the center of the modeling process since other analog functions are easier to build if this basic block were available. Second, a

designer's knowledge of the circuit being modeled is indispensible. Designers are aware of the locations of non-unidirectional nodes and therefore can guide circuit partitioning. The design equations used during circuit design, as Chapter 5 showed, are useful in the modeling of circuit dynamics. Lastly, designers can narrow down the list of non-idealities that are important for a specific application, and simplify and focus the models in a meaningful direction.

Bibliography

[1]. M. J. Rewieński, "A Perspective on Fast-SPICE Simulation Technology," in *Simulation and Verification of Electronic and Biological Systems*, Netherlands: Springer, 2011, pp.23-42.

[2]. M. El-Chammas, and B. Murmann, "A 12-GS/s 81-mW 5-bit Time-Interleaved Flash ADC with Background Timing Skew Calibration," *Solid-State Circuits, IEEE Journal of*, vol.46, no. 4, pp. 838-847, Apr. 2011.

[3]. E. Janssen et al., "An 11b 3.6GS/s Time-Interleaved SAR ADC in 65nm CMOS," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2013 IEEE International*, 2013, pp. 464-465.

[4]. S. Parikh et al., "A 32Gb/s Wirelines Receiver with a Low-Frequency Equalizer, CTLE and 2-Tap DFE in 28nm CMOS," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2013 IEEE International*, 2013, pp.28-29.

[5]. H. Yamaguchi et al., "A 5Gb/s Transceiver with an ADC-based Feedforward CDR and CMA Adaptive Equalizer in 65nm CMOS," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, 2010, pp.168-169.

[6]. R. B. Staszewski et al., "All-Digital PLL and Transmitter for Mobile Phones," *Solid-State Circuits, IEEE Journal of*, vol.40, no. 12, pp. 2469-2482, Dec. 2005.

[7]. T. K. Jang et al., "A 0.062mm^2 5.3mW 32-to2000MHz Digital Fractional-N Phase Locked-Loop Using a Phase-Interpolating Phase-to-Digital Converter," in *Solid-*

State Circuits Conference Digest of Technical Papers (ISSCC), 2013 IEEE International, 2013, pp.254-255.

[8]. D. S. Kim et al., "A Wireless Sensor Node SoC with a Profiled Power Management Unit for IR Controllable Digital Consumer Devices," *Consumer Electronics, IEEE Transactions on,* vol. 56, no. 4, pp. 2282-2287, Nov. 2010.

[9]. K. J. Kerns, M. Bhattacharya, S. Rudnaya, and K. Gullapalli, "Automatic, Hierarchy-Independent Partitioning Method for Transistor-Level Circuit Simulation," U.S. Patent 8 060 355, Jan. 29, 2009 (pending).

[10]. K. J. Kerns and Z. Peng, "SPICE Optimized for Arrays," U.S. Patent 7 324 363, Dec. 12, 2008.

[11]. M. J. Rewieński and K. J. Kerns, "Optimization of Post-Layout Arrays of Cells for Accelerated Transistor Level Simulation," U. S. Patent 8 091 052, Apr. 30, 2009 (pending).

[12]. H. Fleurkens and P. Buurman, "Flexible Mixed-Mode and Mixed-Level Simulation," in *Proc. Circuits and Systems, 1993 IEEE International Symposium on,* 1993, pp. 2137-2140.

[13]. G. Karypis and V. Kumar, (1998, Nov. 22), *hMetis 1.5: A Hypergraph Partitioning Package* [Online], Available: http://glaros.dtc.umn.edu/gkhome/metis/hmetis/overview

[14]. Cadence Design Systems, Inc., *Using Hierarchy and Isomorphism to accelerate Circuit Simulation* [Online], Available: http://w2.cadence.com/whitepapers/5084_AccelCircuitWP_FNL.pdf

[15]. Tcherniaev A. et al., "Transistor Level Circuit Simulator Using Hierarchical Data," U.S. Patent 6 577 992, Jun. 10, 2003.

[16]. M. Zwolinski, K. G. Nichols, A. D. Brown and M. Awan, "A Mixed-Mode Circuit Simulator," in *Proc. UK IT Conference,* 1990, pp. 390-393.

[17]. J. A. Watts, and T. Kwasniewski, "ROOMMS: A Relaxation-based, Object-Oriented, Mixed-Mode Simulator," in *Proc. Custom Integrated Circuits Conference, 1990 IEEE,* 1990, pp. 5.3.1-5.3.4.

[18]. U. Bretthauer, and E. H. Horneber, "BRASIL: The Braunschweig Mixed-Mode-Simulator for Integrated Circuits," in *Proc. Design Automation, IEEE European Conference on,* 1996, pp. 10-14.

[19]. A. F. Carpenter, J. B. Goslin, and H. J. Kahn, "An Accurate Hierarchical Simulation Engine," in *Proc. 3rd Silicon Design Conference,* 1986, pp. 257-66.

[20]. Y.-H. Jun, and I. N. Hajj, "A Mixed-Mode Simulator for Digital/Analog VLSI Circuits Using an Efficient Timing Simulation Approach," in *Proc. Circuits and Systems, 1991 IEEE International Symposium on,* 1991, pp. 2383-2386.

[21]. Y.-H. Jun, and S.-B. Park, "Piecewise Polynomial Models for MOSFET DC Characteristics with Continuous First Order Derivative," *Electronic Circuits and Systems, IEEE Proceedings on,* vol. 135, no. 6, pp. 241-246, Dec. 1988.

[22]. M. Nishigaki, N. Tanaka, and H. Asai, "Mixed Mode Circuit Simulator SPLIT2.1 Using Dynamic Network Separation and Selective Trace," in *Proc. Circuits and Systems, 1994 IEEE International Symposium on,* 1994, vol. 1, pp. 9-12.

[23]. G. Ruan, "A Behavioral Model of A/D Converters Using a Mixed-Mode Simulator," *Solid-State Circuits, IEEE Journal of,* vol.26, no. 3, pp. 283-290, Mar. 1991.

[24]. A. R. W. Todesco, and T. H.-Y. Meng, "Symphony: A Simulation Backplane for Parallel Mixed-Mode Co-Simulation of VLSI Systems," in *Proc. Design Automation Conference, 33rd ACM Annual,* 1996, pp. 149-154.

[25]. R. D. Chamberlain, and M. A. Franklin, “Analysis of Parallel Mixed-Mode Simulation Algorithms,” in *Proc. Parallel Processing Symposium, Fifth International IEEE,* 1991, pp. 155-160.

[26]. K. M. Chandy and J. Misra, “Distributed Simulation: A Case Study in Design and Verification of Distributed Programs,” *Software Engineering, IEEE Transactions on*, vol. SE-5, no. 5, pp. 440-452, Sep. 1979.

[27]. K. M. Chandy and J. Misra, “Asynchronous Distributed Simulation via a Sequence of Parallel Computations,” *Communications of the ACM*, vol. 24, no. 11, pp. 198-206, Apr. 1981.

[28]. L. M. Sokol, D. P. Briscoe and A. P. Wieland, “MTW: A Strategy for Scheduling Discrete Simulation Events for Concurrent Execution,” *Distributed Simulation,* vol, 19, no. 3, pp. 34-42, 1988.

[29]. N. Abdallah, P. B. Sabet, and A. Greiner, “On the Design of Mixed-Mode Simulators for Modern VLSI Circuits,” in *Proc. Circuits and Systems, IEEE 38th Midwest Symposium on,* 1995, vol. 2, pp. 1168-1171.

[30]. D. R. Jefferson, “Virtual Time,” *ACM Transactions on Programming Languages and Systems,* vol. 7, no. 3, pp. 404-425, Jul. 1985.

[31]. A. Hajjar, R. Marbot, A. Greiner, and P. Kiani, “TAS, an accurate timing analyzer for CMOS VLSI,” in *Proc. Design Automation, IEEE European Conference on,* 1991, pp. 261-265.

[32]. C.-J. R. Shi, “Mixed-Signal System-on-Chip Verification Using a Recursively-Verifying-Modeling (RVM) Methodology,” in *Proc. Circuits and Systems, 2010 IEEE International Symposium on,* 2010, pp. 1432-1435.

[33]. M. T. van Stiphout, J. T. J. van Eigndhoven, and H. W. Buurman, "PLATO: A New Piecewise Linear Simulation Tool," in *Proc. Design Automation, IEEE European Conference on*, 1990, pp.235-239.

[34]. J. J. Yang, S. X.-D. Tan, Z. Qi, M. Gawecki, "Hierarchical Symbolic Piecewise-Linear Circuit Analysis," in *Proc. Behavioral Modeling and Simulation Workshop, 2005 IEEE International*, 2005, pp. 140-145.

[35]. J. Wang, X. Li, and L. T. Pileggi, "Parameterized Macromodeling for Analog System-Level Design Exploration," in *Proc. Design Automation Conference, 44th ACM Annual*, 2007, pp. 940-943.

[36]. J. Roychowdhury, "Reduced-order Modeling of Time-Varying Systems," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol. 46, no. 10, pp. 1273-1288, Oct. 1999.

[37]. A. Hajimiri and T. Lee, "A General Theory of Phase Noise in Electrical Oscillators," *Solid-State Circuits, IEEE Journal of*, vol. 33, no. 2, pp. 179-194, Feb. 1998.

[38]. A. Demir, A. Mehrotra, and J. Roychowdhury, "Phase Noise in Oscillators: A Unifying Theory and Numerical Methods for Characterization," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, vol. 47, no. 5, pp. 655-674, May 2000.

[39]. C. Gu, "Algorithmic Nonlinear Macromodeling: Challenges, Solutions and Applications in Analog/Mixed-Signal Validation," in *Proc. Custom Integrated Circuits Conference, 2013 IEEE*, 2013, pp. 1-8.

[40]. W. Rugh, *Nonlienar System Theory – The Volterra-Wiener Approach*, Baltimore: Johns Hopkins University Press, 1981.

[41]. P. Li and L. Pileggi, "Compact Reduced-Order Modeling of Weakly Nonlinear Analog and RF Circuits," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on,* vol. 24, no. 2, pp. 184-203, Feb. 2005.

[42]. J. Phillips, "Projection-Based Approaches for Model Reduction of Weakly Nonlinear, Time-Varying Systems," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on,* vol. 22, no. 2, pp. 171-187, Feb. 2003.

[43]. B. Moore, "Principle Component Analysis in Linear Systems: Controllability, Observability, and Model Reduction," *Automatic Control, IEEE Transactions on,* vol. 26, pp. 17-32, Feb. 1981.

[44]. E. Grimme, "Krylov Projection Methods for Model Reduction," Ph.D. dissertation, University of Illinois, EE Dept., Urbana-Champaign, 1997.

[45]. C. Gu and J. Roychowdhury, "Model Reduction via Projection onto Nonlinear Manifolds, with Applications to Analog Circuits and Bio-Chemical Systems," in *Proc. Computer-Aided Design, IEEE/ACM International Conference on,* pp. 85-92, 2008.

[46]. C. Gu and J. Roychowdhury, "Generalized Nonlinear Timing/Phase Macromodeling: Theory, Numerical Methods and Applications," in *Proc. Computer-Aided Design, IEEE/ACM International Conference on,* pp. 284-291, 2010.

[47]. A. Odabasioglu, M. Celik, and L. T. Pileggi, "PRIMA: Passive Reduced-Order Interconnect Macromodeling Algorithm," in *Prco. Computer Aided Design, 1997 IEEE/ACM International Conference on,* 1997, pp. 58-65.

[48]. M. Rewienski and J. White, "A Trajectory Piecewise-Linear Approach to Model Order Reduction and Fast Simulation of Nonlinear Circuits and Micromachined Devices," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on,* vol. 22, no. 2, pp. 155-170, Feb. 2003.

[49]. C. Gu, “QLMOR: A Projection-Based Nonlinear Model Order Reduction Approach Using Quadratic-Linear Representation of Nonlinear Systems,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 30, no. 9, pp. 1307-1320, Sep. 2011.

[50]. L. Ljung and E. Ljung, *System Identification: Theory for the User*, Upper Saddle River, NJ: 1987.

[51]. R. Isermann and M. Mnchhof, *Identification of Dynamic Systems: An Introduction with Application*, Verlag Berlin Heidelberg: Springer, 2011.

[52]. C. V. Kashyap, and C. S. Amin, “Raven: A Tool for Automatic Generation of Analog Behavioral Models from Schematics,” presented at *Frontiers in Analog Circuit Synthesis and Verification*, Snowbird, Utah, 2011.

[53]. G. Baker and P. Graves-Morris, *Padé Approximants*, New York: Cambridge University Press, 1996.

[54]. J. Kim, K. D. Jones, and M. A. Horowitz, "Variable Domain Transformation for Linear PAC Analysis of Mixed-Signal Systems," in *Proc. Computer-Aided Design, 2007 IEEE/ACM International Conference on*, 2007, pp. 887-894.

[55]. 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 *Proc. Custom Integrated Circuits Conference, 2012 IEEE*, 2012, pp. 1-4.

[56]. J.-E. Jang, M.-J. Park, and J. Kim, “An Event-Driven Simulation Methodology for Integrated Switching Power Supplies in SystemVerilog,” in *Proc. Design Automation Conference, 2013 ACM/EDAC/IEEE 50th*, 2013, pp. 1-7.

[57]. X. Li, P. Li and L. T. Pileggi, “Parameterized Interconnect Order Reduction with Explicit and Implicit multi-parameter Moment Matching for Inter/Intra-die

Variation," in *Proc. Computer-Aided Design, 2005 IEEE/ACM International Conference on*, 2005, pp. 806-812.

[58]. L. T. Pillage and R. A. Rohrer, "Asymptotic waveform evaluation for timing analysis," *Computer-Aided Design, IEEE Transactions on*, vol. 9, pp. 352-366, Apr.1990.

[59]. P. Feldmann and R. W. Freund, "Efficient linear circuit analysis by Padé approximation via the Lanczos process," *Computer-Aided Design, IEEE Transactions on*, vol. 14, no. 5, pp. 649-649, May 1995.

[60]. S. Little, D. Walter, K. Jones, C. Myers, and A. Sen, "Analog/Mixed-Signal Circuit Verification Using Models Generated from Simulation Traces," *International Journal of Foundations of Computer Science*, vol. 21, no.2, pp.191-210, 2010.

[61]. A. Donzé, and O. Maler, "Systematic Simulation Using Sensitivity Analysis," in *Hybrid Systems: Computation and Control*, A. Bemporad, A. Bicchi, G. Buttazzo, Eds., Berlin Heidelberg: Springer, 2007, pp. 174-189.

[62]. S. Batchu, D. Kulkarni, C. Myers, "Automatic Generation of Abstract Models for Analog/Mixed-Signal Circuits," presented at *Frontiers in Analog Circuit Synthesis and Verification*, Snowbird, Utah, 2011.

[63]. C. Gu and J. Roychowdhury, "FSM Model Abstraction for Analog/Mixed-Signal Circuits by Learning from I/O Trajectories," in *Proc. Design Automation Conference, 2011 IEEE 16th Asia and South Pacific*, 2011, pp.7-12.

[64]. G. Zheng, S. P. Mohanty, and E. Kougianos, "Design and Modeling of a Continuous-time Delta-Sigma Modulator for Biopotential Signal Acquisition: Simulink Vs. Verilog-AMS Perspective," in *Proc. Computing Communication & Networking Technologies, 2012 IEEE 3rd International Conference on*, 2012, pp. 1-6.

[65]. F. M. Kolagar, and H. M. Naimi, "An Approach to Transient Analysis of Bang-Bang Phase Locked Loops for Phase Step Inputs," in *Proc. Electrical Engineering, 2011 IEEE 9th Iranian Conference on*, 2011, pp. 1-6.

[66]. A. Gabr, and T. Kwasniewski, "Unifying approach for Jitter Transfer Analysis of Bang-Bang CDR Circuits," in *Proc. Electronics and Information Engineering, 2010 IEEE International Conference on*, 2010, vol. 2, pp. V2-40.

[67]. J. W. Sun, H. L. You, Z. Y. Li, and X. Z. Jia, "Research on Modeling Control Module of DC-DC Converter for Simulink," in *Proc. Power Electronics and Intelligent Transportation System, 2009 IEEE 2nd International Conference on,* 2009, vol. 1, pp. 129-132.

[68]. J. David, "Verification of CML circuits used in PLL contexts with Verilog-AMS," in *Proc. Behavioral Modeling and Simulation Workshop, 2006 IEEE International,* 2006, pp. 97-102.

[69]. K. Ma, "A Fast and Accurate SystemC-AMS Model for PLL," in *Proc. Mixed Design of Integrated Circuits and Systems, 2011 IEEE 18th International Conference on*, 2011, pp. 411-416.

[70]. Open SystemC Initiative (2010, Oct. 02), *SystemC AMS extension User's Guide* [Online], Available: http://www.systemc.org/downloads/standards

[71]. F. Cenni, S. Scotti, and E. Simeu, "SystemC AMS Behavioral Modeling of a CMOS Video Sensor," in *Proc. VLSI and System-on-Chip, 2011 IEEE/IFIP 19th International Conference on*, 2011, pp. 380-385.

[72]. G. S. Beserra, J. E. G. de Medeiros, A. M. Sampaio, and J. C. da Costa, "System-Level Modeling of a Mixed-Signal System on Chip for Wireless Sensor Networks," in *Proc. Design, Automation & Test in Europe Conference & Exhibition, 2011 IEEE*, 2011, pp. 1-4.

[73]. P. Gang, "Behavioral Modeling and Simulation of Analog/Mixed-Signal Systems Using Verilog-AMS," in *Proc. Information, Computing and Telecommunication, 2009 IEEE Youth Conference on,* 2009, pp. 383-386.

[74]. X. Lai, Y. Zhang, Y. Li, and X. M. Liu, "Behavioral Modeling of Electronic Circuit Module with Verilog-A Language," in *Proc. ASIC, 2001 IEEE 4th International Conference on,* 2001, pp. 155-158.

[75]. B. Troyanovsky, P. O'Halloran, and M. Mierzwinski, "Analog RF Model Development with Verilog-A," in *Radio Frequency Integrated Circuits (RFIC) Symposium, 2005 IEEE,* 2005, pp. 287-290.

[76]. Y. Wang, C. Van-Meersbergen, H.-W. Groh, S. Heinen, "Event Driven Analog Modeling for the Verification of PLL Frequency Synthesizers," in *Proc. Behavioral Modeling and Simulation Workshop, 2009 IEEE International,* 2009, pp. 25-30.

[77]. D. Dumlugol, and D. Webber, "Analog Modeling Using Event-Driven HDL's," in *Proc. VLSI Design, 1994 IEEE 7th International Conference on,* 1994, pp. 53-56.

[78]. R. B. Staszewski, C. Fernando, and P. T. Balsara, "Event-Driven Simulation and Modeling of Phase Noise of an RF Oscillator," *Circuits and Systems I: Regular Papers, IEEE Transactions on,* vol. 52, no. 4, pp. 723-733, 2005.

[79]. T. J. Wen, and T. Kwasniewski, "Phase Noise Simulation and Modeling of ADPLL by SystemVerilog," in *Proc. Behavioral Modeling and Simulation Workshop, 2008 IEEE International,* 2008, pp. 29-34.

[80]. A. Prodic, and D. Maksimovic, "On Behavioral Modeling of a Mixed-Signal Analog to Digital Converter," in *Proc. Computers in Power Electronics, 2002 IEEE Workshop on,* 2002, pp. 100-105.

[81]. B. C. Lim, J. Kim, and M. A. Horowitz, "An Efficient Test Vector Generation for Checking Analog/Mixed-Signal Functional Models," in *Proc. 47th ACM Design Automation Conference*, 2010, pp. 767-772.

[82]. M. Horowitz, M. Jeeradit, F. Lau, S. Liao, B. Lim and J. Mao, "Fortifying Analog Models with Equivalence Checking and Coverage Analysis," in *Proc. 47th ACM Design Automation Conference*, 2010, pp. 425-430.

[83]. T. R. Dastidar, and P. P. Chakrabarti, "A Verification System for Transicent Response of analog Circuits Using Model Checking," in *Proc. VLSI Design, 2005 IEEE 18th International Conference on*, 2005, pp. 195-200.

[84]. G. Zhou, J. Xu, J. Wang, and Y. Jin, "Comparison Study on Digital Peak Current, Digital Peak Voltage, and Digital Peak Voltage/Peak Current Controlled Buck Converter," in *Proc. Industrial Electronics and Applications, 2009 IEEE 4th Conference on*, 2009, pp. 799-804.

[85]. C.-A. Yeh, and Y.-S. Lai, "Digital Pulsewidth Modulation Technique for a Synchronous Buck DC/DC Converter to Reduce Switching Frequency," *Industrial Electronics, IEEE Transactions on*, vol. 59, no. 1, pp. 550-561, 2012.

[86]. R. Redl, and J. Sun, "Ripple-Based Control of Switching Regulators – An Overview," *Power Electronics, IEEE Transactions on*, vol. 24, no. 12, pp. 2669-2680, 2009.

[87]. C. Werner et al., "Modeling, simulation and design of a multi-mode 2-10Gb/sec fully adaptive serial link system," in *Proc. Custom Integrated Circuits Conference, 2005 IEEE*, 2005, pp. 709-716.

[88]. W.M.G. van Bokhoven, "Piecewise-Linear Modeling and Analysis," Ph.D. Dissertation, Dept. of Engineering, Technical University of Eindhoven, Eindhoven, The Netherlands, May 1981.

[89]. K. F. Wong and M. A. Franklin, “Load and Communications Balancing in Multiprocessor Logic Simulation Engines,” in *Proc. International Workshop in Hardware Accelerators*, 1987, pp. 80-89.

[90]. W. Kim, J. Park, J. Kim, T. Kim, H. Park and D. Jeong, “A 0.032mm2 3.1mW Synthesized Pixel Clock Generator with 30psrms Integrated Jitter and 10-to-630MHz DCO Tuning Range,” in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2013 IEEE International,* 2013, pp. 250-251.

[91]. W. Deng et al., “A 0.0066mm2 780uW Fully Synthesizable PLL with a Current-Output DAC and an Interpolative Phase-Coupled Oscillator Using Edge-Injection Technique,” in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International,* 2014, pp. 266-267.

[92]. A. Elkholy, A. Elshazly, S. Saxena, G. Shu, and P. Hanumolu, “A 20-to-1000MHz +/-14ps Peak-to-Peak Jitter Reconfigurable Multi-Output All-Digital Clock Generator Using Open-Loop Fractional Dividers in 65nm CMOS,” in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International,* 2014, pp. 272-273.

[93]. B. Razavi, “Design of Monolithic Phase-Locked Loops and Clock Recovery Circuits – A Tutorial,” in *Monolithic Phase-Locked Loops and Clock Recovery Circuits: Theory and Design*. IEEE Press, 1996.

[94]. S. Arora, D. K. Su and B. A. Wooley, “A Compact 120-MHz 1.8V/1.2V Dual-Output DC-DC Converter with Digital Control,” in *Proc. Custom Integrated Circuits Conference, 2013 IEEE*, 2013, pp. 1-4.

[95]. H. Orser and A. Gopinath, “A 20Gs/s 1.2V 0.13um CMOS Switched Cascode Track-and-Hold Amplifier,” *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 57, no. 7, Jul. 2010.

[96]. A. M. Abo and P. R. Gray, "A 1.5-V, 10-bit, 14.3-MS/s CMOS Pipeline Analog-to-Digital Converter," *Solid-State Circuits, IEEE Journal of*, vol.34, no. 5, pp. 599-606, May 1999.

[97]. Z. Wang and M.-C. F. Chang, "A 600Msps 8-bit CMOS ADC Using Distributed Track-and-Hold with Complementary Resistor/Capacitor Averaging," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol.55, no. 11, pp. 3621-3627, Dec. 2008.

[98]. A. Demir, "Computing Timing Jitter from Phase Noise Spectra for Oscillators and Phase-Locked Loops with White and 1/f Noise," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol.53, no. 9, pp. 1869-1884, Sep. 2006.

[99]. M. J. M. Pelgrom, H. P. Tuinhout and M. Vertregt, "Transistor Matching in Analog CMOS Applications," in *Technical Digest Electron Devices Meeting, 1998 IEEE*, 1998, pp. 915-918.

[100]. A. Oppenheim and R. Schafer, *Digital Signal Processing*, Upper Saddle River, NJ: Prentice-Hall, 1975.

[101]. W. Yu, S. Sen and B. H. Leung, "Distortion Analysis of MOS Track-and-Hold Sampling Mixers Using Time-Varying Volterra Series," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol.46, no. 2, pp. 101-113, Feb. 1999.