ABCD-NL: Approximating Continuous Non-Linear Dynamical Systems using Purely Boolean Models for Analog/Mixed-Signal Verification

Aadithya V. Karthik‡, Sayak Ray, Pierluigi Nuzzo, Alan Mishchenko, Robert Brayton, and Jaijeet Roychowdhury
The Department of Electrical Engineering and Computer Sciences, The University of California, Berkeley, CA, USA
‡Corresponding author. Email: aadithya@berkeley.edu

***Abstract*—We present ABCD-NL, a technique that approximates non-linear analog circuits using purely Boolean models, to high accuracy. Given an analog/mixed-signal (AMS) system (*e.g.*, a SPICE netlist), ABCD-NL produces a Boolean circuit representation (*e.g.*, an And Inverter Graph, Finite State Machine, or Binary Decision Diagram) that captures the I/O behaviour of the given system, to near SPICE-level accuracy, without making any *apriori* simplifications. The Boolean models produced by ABCD-NL can be used for high-speed simulation and formal verification of AMS designs, by leveraging existing tools developed for Boolean/hybrid systems analysis (*e.g.*, ABC [1]). We apply ABCD-NL to a number of SPICE-level AMS circuits, including data converters, charge pumps, comparators, non-linear signaling/communications sub-systems, *etc.* Also, we formally verify the throughput of an AMS signaling system – modelled in SPICE using 22nm BSIM4 transistors, Booleanized with high accuracy using ABCD-NL, and property-checked using ABC.**

I. INTRODUCTION

AMS systems are becoming increasingly important in chip design. In recent times, AMS blocks have become key components that limit system-level performance [2]. Also, AMS components now account for a significant proportion of design bugs, designer time, and debugging cost. However, CAD tools for AMS verification have not kept pace with the rapid growth in complexity of these systems.

An important challenge for AMS verification is the *accurate modelling* of AMS components; because these components can introduce design flaws/loss of performance in a variety of subtle and non-obvious ways, it is important to model their behaviour at or near SPICE-level accuracy.

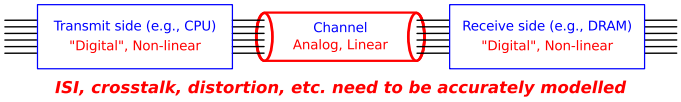

Fig. 1. Schematic of a typical AMS signaling/communications sub-system that arises in signal integrity analysis.

For example, Fig. 1 depicts an AMS system that frequently arises in Signal Integrity (SI) applications. The system consists of digital components on the transmit side (*e.g.*, a CPU), whose outputs enter an analog channel. The channel introduces inter-symbol interference (ISI), crosstalk, *etc.* The other end of the channel (the receive side) has more digital components (*e.g.*, a DRAM/memory controller). A key figure of merit of this system is its throughput, *i.e.*, the maximum bitrate that can be reliably sustained. Guaranteeing the system's throughput is a non-trivial AMS verification problem, and to issue a meaningful guarantee, it is necessary to model this system at SPICE-level accuracy.

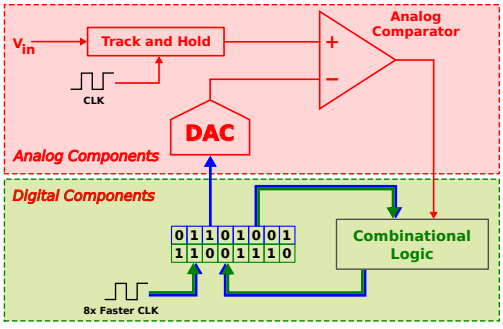

Fig. 2. Schematic of a successive approximation A/D converter (SAR-ADC).

To take another example, Fig. 2 depicts a successive approximation A/D converter (or SAR-ADC). This system contains a number of components, both analog and digital. The system's performance (*i.e.*, its speed, power consumption, *etc.*) is frequently limited by the analog components (*e.g.*, the speed of the DAC, the sensitivity/bandwidth of the comparator, *etc.*). Therefore, to verify the SAR-ADC, it is often necessary to model the analog components at SPICE-level accuracy (even though it may be sufficient to model digital components at the Boolean level).

Most existing approaches to AMS verification (*e.g.*, [3]–[8]), however, *do not* model AMS components at SPICE-level accuracy. The existing approaches are usually based on hybrid systems methods (*e.g.*, [9]–[14]); although these methods can reason about continuous analog quantities, they tend to be limited in terms of scalability. For example, they can fail even for relatively small AMS systems consisting of just 5-10 analog signals. Due to this limitation, existing verification flows usually adopt highly simplified "behavioural" macromodels for AMS components, which do not capture many performance-limiting analog effects (*e.g.*, non-linear operating regions, DC offsets, ISI, distortion, *etc.*). As a result, AMS designers often have to carry out extensive and time-consuming SPICE simulations; this is a tedious, expensive, and error-prone process.

In an attempt to address the above concerns, a technique called ABCD-L [15] was recently proposed. ABCD-L represents linear time invariant (LTI) AMS components using *purely Boolean* models, *i.e.*, without the need for any continuous variables[1]. ABCD-L has been shown to capture the dynamics of linear AMS components to almost SPICE-level accuracy [15]. Also, because ABCD-L models use only "cheap" Boolean variables, and not "expensive" continuous variables, the formal analysis and verification involving these models can be very efficient, using either the state-of-the-art Boolean techniques (*e.g.*, ABC [1]), or in conjunction with existing hybrid systems frameworks. The biggest drawback of ABCD-L, however, is that it applies only to linear systems. Therefore, ABCD-L cannot be used for verifying systems such as the communications sub-system of Fig. 1, or the SAR-ADC of Fig. 2, because these systems are strongly non-linear.

In this paper, we propose ABCD-NL[2], a new method that has all the advantages of ABCD-L, and in addition, works for a large class of non-linear systems. Given a non-linear AMS system (*e.g.*, a SPICE netlist), ABCD-NL produces a purely Boolean model (*e.g.*, as a Finite State Machine (FSM), an And Inverter Graph (AIG), or a Binary Decision Diagram (BDD)) that captures the dynamics of the given system to near SPICE-level accuracy. Furthermore, the Boolean model produced by ABCD-NL is well-suited for use with cutting-edge formal verification/model checking engines such as ABC [1], or with existing hybrid systems frameworks. Also, the Boolean model produced by ABCD-NL can be simulated very efficiently at the logic level (without needing to solve any differential equations), so ABCD-NL can be used as a much faster, almost-as-accurate, drop-in replacement for SPICE, in many applications.

ABCD-NL requires only one condition: a DC input to the given system should eventually result in a DC output. This condition is satisfied by almost all non-linear systems of interest to AMS designers, *e.g.*, D/A and A/D converters, amplifiers and comparators, linear and non-linear filters, equalizers, switches and multiplexers, charge pumps, I/O links, *etc.*

In the next section (§II), we describe the core techniques underlying

[1]Similar ideas have also been suggested in works like [16] and [17].
[2]**A**ccurate **B**ooleanization of **C**ontinuous **D**ynamics – **N**on-**L**inear

ABCD-NL. We then apply ABCD-NL (in §III) to a number of non-linear systems that are of interest to AMS designers. For example, in §III-A, we Booleanize a charge pump/filter system using ABCD-NL. We also apply ABCD-NL to the analog components that make up the SAR-ADC of Fig. 2, such as the D/A converter (§III-C) and the comparator (§III-D). In all these cases, we show that ABCD-NL's Boolean model faithfully reproduces the circuit's behaviour.

We note that the primary focus of this paper is on *modelling AMS systems for verification*, as opposed to the verification itself. However, for completeness, we present (in §III-E) an example where we use the verification tool ABC [1], together with a Boolean model produced by ABCD-NL, to formally verify the throughput of a communications sub-system of the type shown in Fig. 1.

II. CORE TECHNIQUE: A NEW ALGORITHM FOR BOOLEANIZING NON-LINEAR ANALOG CIRCUITS

In this section, we describe the key ideas behind ABCD-NL. As outlined in §I, ABCD-NL takes a SPICE netlist as input, and it produces as output a purely Boolean model (*e.g.*, an FSM) of the given circuit.

The Boolean model represents the circuit's inputs and outputs using finite numbers of bits. The bits returned by the model are interpreted by the user as discretized analog waveforms. The goal is to preserve the behaviour of the original circuit as closely as possible in the Boolean model.

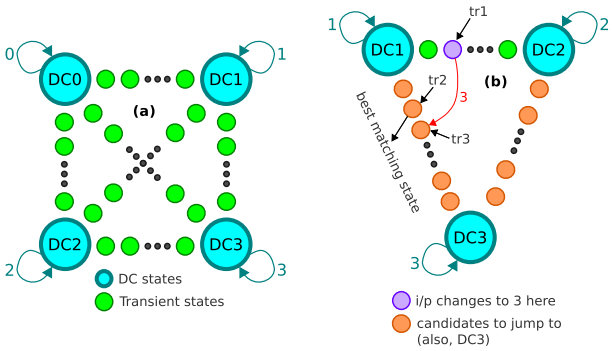

Fig. 3. (a) Structure of the FSM model derived by ABCD-NL from SPICE simulations, and (b) ABCD-NL's method of exploiting continuity to jump from one transient FSM state to another.

Algorithm 1: Converting an analog circuit into a purely Boolean ABCD-NL model

Inputs: SPICE netlist *cir*, Signal list *siglist*, Output signal *sigout*, FSM time step *tstepFSM*
Output: Purely Boolean FSM model *fsm* for the given circuit

```
1  fsm = new FSM()
2  DCInputs = enumerateDiscretizedCktInputs()
3  insertDCFSMStates(fsm, DCInputs)
4  for initInput in DCInputs:
5      initFSMState = encodeFSMState(initInput)
6      for finalInput ≠ initInput in DCInputs:
7          finalFSMState = encodeFSMState(finalInput)
8          cirInputWaveforms = generateStepFunctions(initInput, finalInput)
9          [ts, vs] = SPICESimulateTran(cir, cirInputWaveforms)
10         tSettle = estimateSettlingTime(ts, vs, siglist)
11         numTranFSMStates = ceil(tSettle ÷ tstepFSM) − 1
12         insertTranFSMStates(fsm, initFSMState, finalFSMState, numTranFSMStates)
13         annotateFSMArcs(fsm, initFSMState, finalFSMState, ts, vs, sigout)
14         annotateFSMStates(fsm, initFSMState, finalFSMState, ts, vs)
15 foreach transient FSM state trst in fsm:
16     foreach discrete input inp ≠ trst.finalInput in DCInputs:
           /* use state/output continuity to obtain new FSM arc */
17         nextst = estimateNextFSMState(fsm, trst, inp)
18         outp = estimateDiscreteOutputOnTransition(fsm, trst, nextst, inp)
19         insertFSMArc(trst, nextst, inp, outp)
20 return fsm
```

Fig. 3 (a) depicts the Boolean model produced by ABCD-NL. Each discretized input combination is associated with a *DC state* (that has a self loop) in an FSM. These DC states are akin to DC operating points of the circuit. For example, if the circuit has 2 inputs, and each input is discretized using 3 bits (*i.e.*, 8 levels), the ABCD-NL FSM would have 64 DC states. These states capture the circuit's DC behaviour.

To capture the transient behaviour, ABCD-NL introduces additional *transient FSM states* between every pair of DC states described above. The exact number of transient states introduced depends on the dynamics of the circuit, *i.e.*, the time taken by the system to transition from one DC operating point to another, when a suitable step transition is applied at the input. Together, the DC states and the transient states capture the dynamics of the given system to high accuracy.

Algorithm 1 formally describes the FSM construction procedure used by ABCD-NL. Line 3 creates the DC states described above. For every pair of such DC states, ABCD-NL performs a transient SPICE simulation (Line 9), the results of which are used to create the transient FSM states (Lines 10–12). Note that, if needed, the user can explicitly specify a list of important signals (*siglist*) in the given circuit, which the algorithm takes into account while creating the transient states (by default, the algorithm considers all signals as important). Further, the algorithm uses the SPICE simulations above to label each FSM arc (Line 13) with an appropriate (discretized) output symbol.

Algorithm 2: Simulating ABCD-NL's Boolean model, and post-processing to the analog domain

Inputs: Boolean model *fsm*, Ckt. inputs $u(t)$, FSM step *tstepFSM*, simulation interval $[t_0, t_f]$
Output: Simulation trace [*ts*, *vs*] of the circuit's output *sigout* over the interval $[t_0, t_f]$

```
1  ts = [], vs = []
   /* start at the DC operating point for the input u at time t0 */
2  t_curr = t0
3  u_curr = discretizeInput(u(t_curr))
4  state_curr = encodeFSMState(u_curr)
5  while t_curr ≤ tf:
       /* simulate one time point by looking up the FSM's state transition table */
6      transitionArc = fsm.nextStateArc(t_curr, u_curr)
       /* Look up the o/p on the transition arc, and post-process it from Boolean to analog */
7      output_curr = transitionArc.outputSymbol
8      analogOutput_curr = BooleanToAnalog(output_curr)
       /* record the output */
9      ts.append(t_curr)
10     vs.append(analogOutput_curr)
       /* update the simulation variables for the next time point */
11     t_curr += tstepFSM
12     u_curr = discretizeInput(u(t_curr))
13     state_curr = transitionArc.finalState
14 return [ts, vs]
```

By this time, all the states of the ABCD-NL FSM have been created, and all arcs have been specified for the DC states. However, not all arcs have been specified for the transient states. To fully specify the system, ABCD-NL uses an interpolation-based heuristic, as shown in Fig. 3 (b).

For example, suppose that the discretized version of the applied input switches from one Boolean-encoded value (say, 1), to another (say, 2). Corresponding to this, the FSM starts moving from state DC1 to DC2, as shown in Fig. 3 (b). However, before the FSM reaches DC2 (*i.e.*, when the FSM is in the state marked tr1), let us say the (discretized) input switches again, this time to 3. This forces the FSM to move towards state DC3, and we need a method to determine the next action of the FSM. For this, ABCD-NL takes advantage of the continuity of the underlying analog waveforms (Lines 17 to 19); it selects a transient FSM state that is, in some sense, "closest" to the current state tr1, along the paths DC1–DC3 and DC2–DC3 (this is possible because the ABCD-NL synthesis algorithm (Line 14) internally maintains an estimate of the *analog state* of the circuit at each *Boolean state* of the FSM). This completes the Boolean model generation, and the resulting FSM is returned, which, if necessary, can be transformed into an AIG or BDD using existing tools such as ABC [1]. (Our implementation of ABCD-NL ensures that its output can be directly read in by tools such as ABC.)

Algorithm 2 formalises how the ABCD-NL Boolean model can be simulated in the time-domain, at the logic level. Each time step in this simulation is simply a table lookup (Line 6), so there are no differential equations or Newton-Raphson iterations involved. Thus, ABCD-NL based simulation is much faster than SPICE. For example, even though we have implemented ABCD-NL in Python (a language not known for being fast), it was still 2x to 3x faster than HSPICE, for most examples presented in the next section. We believe that, if the code is re-written in C/C++, it would be quite straightforward to achieve ∼30x speedup over HSPICE.

III. RESULTS

We now apply ABCD-NL to circuits that are of interest to AMS designers. These include, (1) a charge pump/filter system that is relevant to PLL design, (2) a signaling/communications sub-system that involves (non-linear) digital logic interfacing with an analog channel, (3) a D/A converter, and, (4) an analog comparator, the last two circuits being important analog components that make up a SAR-ADC. In each case, we show that the Boolean model produced by ABCD-NL is able to accurately reproduce the SPICE-level analog dynamics of the underlying circuit, including important performance-limiting non-ideal phenomena.

A. Charge pump driving an analog filter

Fig. 4 shows a charge pump driving an analog filter, a system that plays an important role in PLLs. The system works as follows: the transistors M1 and M2 form a current mirror that can pump a current I_0 into the load capacitor C_L, whereas transistors M3 and M4 form an opposing current mirror that can withdraw current I_0 from C_L. The circuit has two inputs, V_{up} and V_{down}. During normal operation, exactly one of these inputs is high; if V_{up} (V_{down}) is high, current is pumped in (out), driving the output voltage V_{out} higher (lower); this is called the charging (discharging) mode of the charge pump, and the system responds most quickly when in one of these modes (*e.g.*, using a 90nm process, response times are typically of the order of tens of nanoseconds).

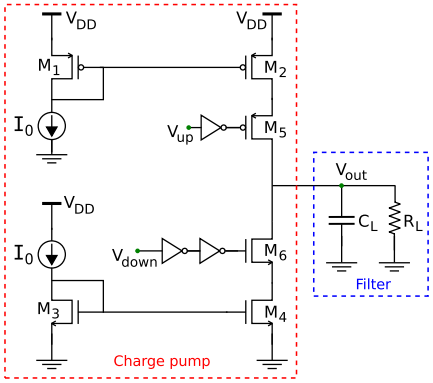

Fig. 4. Schematic of a charge pump driving an analog filter.

In addition to the normal mode of operation, it is also important to capture the behaviour of the charge pump under anamolous inputs; for example, if both inputs are high, the charge pump enters an imbalance-driven mode, which can either charge or discharge the load, depending on operating conditions. This type of charging/discharging is typically much slower than normal operation (*e.g.*, hundreds of nanoseconds response time), because only a small current flows through the load. Finally, if both inputs are off (cutoff mode), the output voltage, under ideal conditions, would remain constant; however, due to leakage currents and the resistive load R_L, the capacitor C_L slowly discharges to a DC voltage that is almost 0. The cutoff mode is the slowest mode of operation of the system, with response time in the microsecond range.

Since PLL performance critically depends on the non-linear charge pump/loop-filter dynamics, our goal is to use ABCD-NL to accurately model the behaviour of the above system under all possible operating conditions: charging, discharging, imbalance-driven, and cutoff. So we designed the system in 90nm CMOS, using BSIM4 device models. We then applied Algorithm 1 (of §II) to Booleanize this system (using 5 bits to encode the output waveform), and we simulated the resulting Boolean model using Algorithm 2, on a range of inputs that covered all four modes of operation. In each case, we compared the output predicted by ABCD-NL's Boolean model, against that predicted by HSPICE. Fig. 5 shows the results, where HSPICE waveforms are shown in blue, and ABCD-NL waveforms are in green. As the figure shows, in spite of the widely differing time scales involved in the four modes, the Boolean model produced by ABCD-NL closely matches the SPICE-level dynamics of the system in all its modes.

Fig. 6 further demonstrates the accuracy and robustness of the Boolean model produced by ABCD-NL. The figure shows a long pseudo-random bit sequence applied as input to the circuit, which switches the circuit in and out of all 4 modes of operation over a long time frame. Throughout this time, it is seen that the Boolean model produced by ABCD-NL (the green waveform) closely tracks the SPICE-simulated output (the blue waveform) of the system. This indicates that ABCD-NL is indeed a powerful and accurate modelling technique, and one that can conceivably be used as a much faster, almost-as-accurate, drop-in replacement for SPICE over long transient runs.

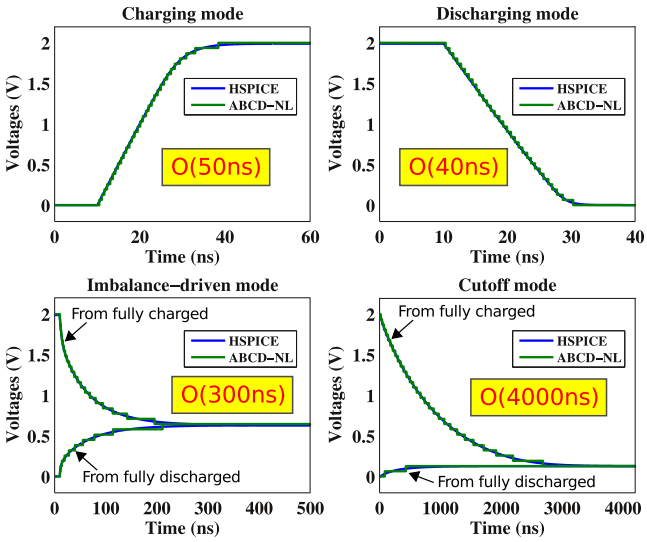

Fig. 5. ABCD-NL accurately captures the behaviour of the charge pump under all four modes of operation, in spite of the widely differing time scales involved.

B. Signaling system: Non-linear digital logic + analog channel

Fig. 7 shows a mixed-signal sub-system, of the type depicted in Fig. 1. As we mentioned in §I, such systems are often encountered in Signal Integrity (SI) applications. In this example, the transmit side takes 3 bits as input (A, B, and C_{in}), and adds them up using a full-adder, thereby producing 2 output bits: the sum S, and the output carry C_{out}. These 2 bits are then sent across an analog channel that consists of several RC stages, inter-linked with coupling capacitances. At the other end of the channel, the receiver cleans the arriving waveforms S_{ch} and C_{ch} using chains of inverters.

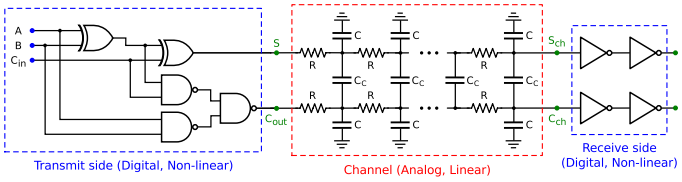

Fig. 7. A signaling/communications sub-system that arises in SI applications.

We have modelled each transistor in the above system using a 22nm BSIM4 analog SPICE model (obtained from [18]). Therefore, the system above exhibits several realistic non-linear analog effects, including leakage currents, loading effects, delays, channel-induced ISI/crosstalk, *etc.* Our goal is to use ABCD-NL to accurately reproduce the analog waveforms at the output of the channel (S_{ch} and C_{ch}), in the presence of these adverse analog effects. This is a crucial requirement for SI analysis.

Fig. 8 shows the results obtained by applying ABCD-NL to the above system. Part (a) shows three randomly generated 40-bit sequences (for A, B, and C_{in}, respectively), applied as inputs to the circuit. In part (b), these inputs are applied at a bitrate of 1 Gbps. At this bitrate, the system behaves in a fairly ideal manner, *i.e.*, distortion, crosstalk, *etc.* are minimal, as seen from the blue HSPICE waveforms S_{ch} and C_{ch} of Fig. 8 (b). Also, the green waveforms in the figure show the predictions made by ABCD-NL's purely Boolean model (using 4 bits to encode each output waveform); as the figure shows, ABCD-NL's purely Boolean model is able to accurately predict the system's dynamics at this bitrate.

Fig. 8 (c) shows the same bit pattern applied to the circuit, but at a higher bitrate (3.2 Gbps). At this bitrate, the system produces considerable

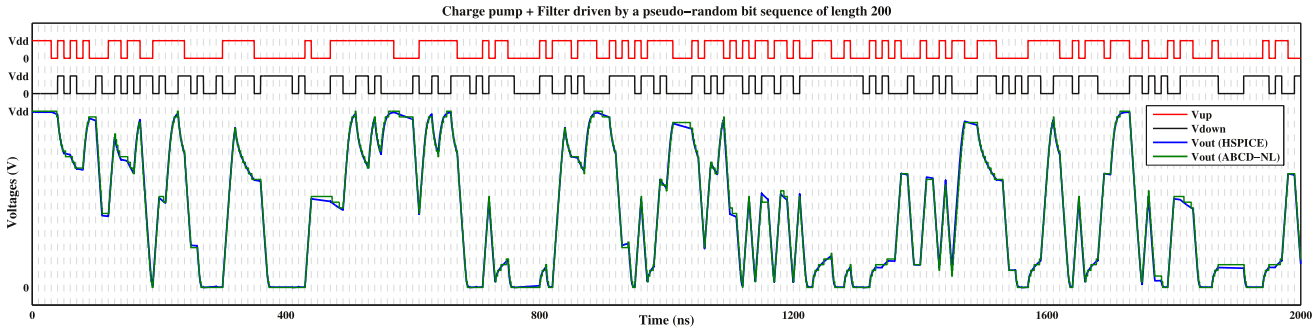

Fig. 6. ABCD-NL can predict the response of the charge pump/filter system, almost to SPICE-level accuracy, even over long time frames that involve rapid switching between all 4 modes of operation.

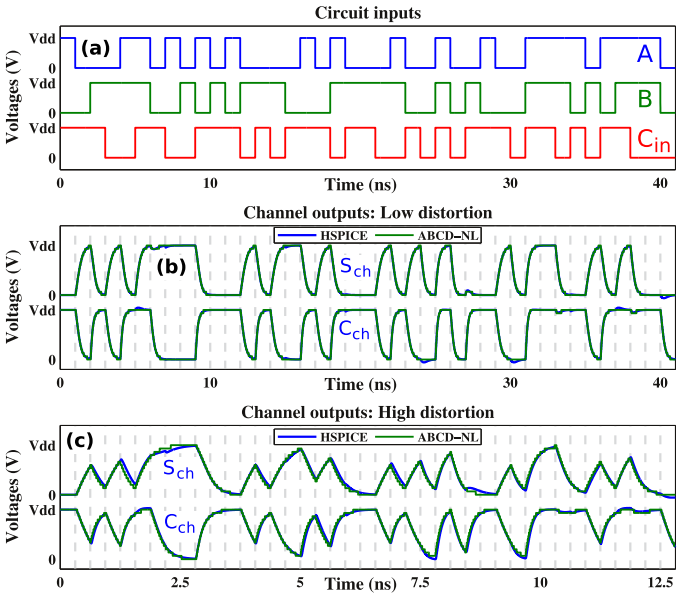

Fig. 8. ABCD-NL closely matches the SPICE-level analog behaviour of the signaling/communications sub-system of Fig. 7, at both high and low bitrates.

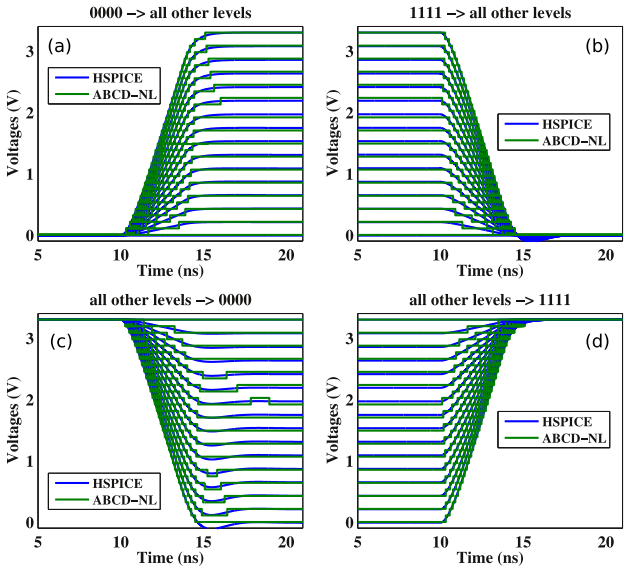

Fig. 10. ABCD-NL closely matches SPICE-level simulation of the D/A converter, for several input bit transitions.

distortion, with significant ISI and crosstalk. Here also, it is seen from the figure that ABCD-NL is able to accurately reproduce the analog dynamics exhibited by the system. This demonstrates that ABCD-NL is indeed a viable modelling technique for SI applications, even when the underlying system exhibits pronounced non-linear analog effects.

C. D/A converter (DAC)

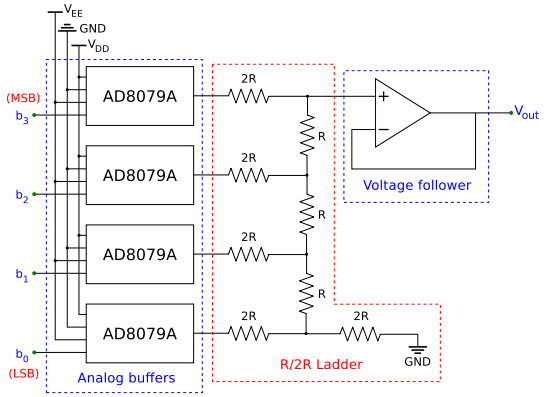

Fig. 9. Schematic of a D/A converter used within a SAR-ADC.

We now apply ABCD-NL to produce a purely Boolean model of a canonical mixed-signal system, a D/A converter used within a SAR-ADC. As Fig. 9 shows, we have a 4-bit D/A converter consisting of four Analog Devices AD8079A analog buffers (SPICE models available from [19]), and an R/2R ladder that feeds into a voltage follower.

A key figure of merit of a D/A converter is speed; so it is important to accurately capture the delay of the system for all possible bit transitions at the input. Fig. 10 shows many of these transitions (due to space constraints, we are unable to show all the transitions), and indeed, it can be seen from the figure that ABCD-NL accurately captures the system's delay for all these inputs (using 6 bits to encode the D/A output).

Furthermore, because our D/A converter is embedded within a SAR-ADC, it is important to have our Boolean model reproduce the system's dynamics for input patterns that are typical to the SAR-ADC environment. Fig. 11 illustrates this environment for a 4-bit SAR-ADC. The red waveform shows a 150kHz sine wave, which is the ADC input. The ADC operates at about 8MHz, so each period of the input generates about 52 ADC samples. Over these samples, the input bits b_0 to b_3 of the D/A converter switch as shown in the top half of Fig. 11. The blue waveform at the bottom of the figure depicts the D/A output, as predicted by HSPICE. And as seen from the green waveform, ABCD-NL is able to reproduce this response very accurately. This shows that ABCD-NL is indeed an accurate and powerful way to Booleanize non-linear data converters for analysing mixed-signal systems.

D. Analog comparator

We now apply ABCD-NL to an analog comparator, another key component in a SAR-ADC. For this demonstration, we shall Booleanize an off-the-shelf comparator (LT1016 from Linear Technology, whose SPICE model is available online [20]), and deploy the resulting Boolean model in a SAR-ADC environment.

Booleanizing a SAR-ADC comparator is a particularly challenging problem because the circuit is highly non-linear, and very sensitive to its (large signal) inputs; for example, a differential input of 1mV elicits a very different response from the system compared to a 2mV (or 1V) differential, in terms of delay, the final steady state solution, *etc.* This necessitates very fine discretization of the input waveforms, which can

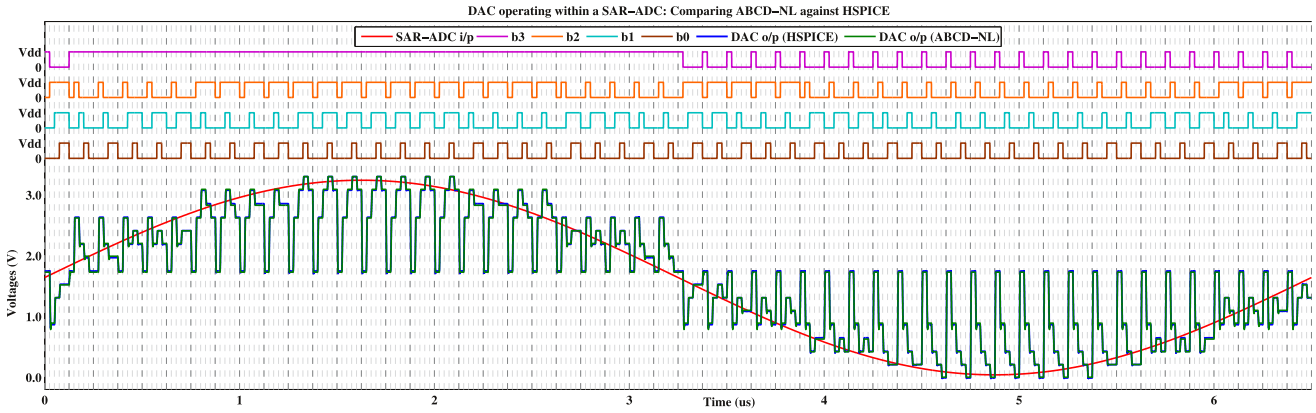

Fig. 11. ABCD-NL, using a purely Boolean model, is able to accurately capture the dynamics of a D/A converter embedded within a SAR-ADC, across a long time frame encompassing several ADC samples.

in turn result in a very large Boolean model unless domain knowledge is used (see below). Moreover, for each input sample of the SAR-ADC, the comparator usually begins at a large differential (of the order of 1V), and the closed-loop dynamics of the system uses feedback to progressively make this differential smaller, until it is of the order of 1mV. And the Boolean model must capture the dynamics of the comparator, to high accuracy, in spite of this large operating range.

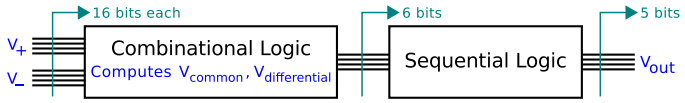

Fig. 12. ABCD-NL can use domain knowledge to achieve better efficiency. For a comparator, rather than directly operating on the input bits, it is significantly more efficient to first transform the input signals into common mode and differential mode components (using combinational logic), which can then be used to drive ABCD-NL's sequential Boolean model.

As indicated above, to reduce the size of the Boolean model, ABCD-NL supports the use of domain knowledge. Fig. 12 illustrates this point for the comparator. Because the input has to be discretized very finely (*e.g.*, using 16 bits or more per signal), it is inefficient to apply ABCD-NL directly to the discretized input. Instead, we use a combinational logic unit to compute the *common mode* and *differential* components of the input, which can be discretized relatively coarsely (*e.g.*, using just 6 bits for the differential mode, and 0-2 bits for the common mode). In particular, the differential component is discretized non-uniformly, placing more emphasis on small differentials (because they exert a powerful influence on system dynamics), and a smaller emphasis on large differentials (where the behaviour quickly saturates).

Fig. 13 illustrates the results. Part (b) of the figure shows that ABCD-NL, with the efficiency enhancements above, is able to accurately reproduce the SPICE-level analog behaviour of the comparator, for a wide range of input excitations. These excitations are generated as follows: initially, the input differential between the inputs V_+ and V_- is chosen to be either *large* (1V) or *small* (1mV). This choice has the effect of biasing the comparator either at a strongly polarized bias point, or a weak bias point. Now, a second choice is made: the input differential is suddenly reversed in polarity, using either a small driving strength (1mV differential), or a large driving strength (1V differential). This choice has a significant impact on response time; for example, a strong differential signal starting from a weakly polarized system state evokes a much faster response than a weak differential trying to flip the system from a strongly polarized state. The goal is to design the Boolean model to accurately capture all the different corner cases, and as Fig. 13 (b) shows, the Boolean model produced by ABCD-NL achieves this goal (using 5 bits to encode the output waveform).

Also, for verification purposes, it is important to model the comparator's departure from ideal behaviour. An important factor in this context is the DC sensing offset of the comparator. For example, Fig. 13 (c) shows a situation where the comparator behaves in a highly unexpected way: even though V_- is always higher than V_+ (*i.e.*, an ideal comparator's output would remain low throughout), the LT1016 actually switches from low to high. Such unexpected behaviour can potentially introduce bit-errors in the context of a SAR-ADC, due to incorrect decisions made by the comparator within the feedback loop. Therefore, while analyzing a SAR-ADC that uses this comparator, it is important to use a comparator model that accurately accounts for such imperfections and shortcomings. And as Fig. 13 (c) shows, ABCD-NL's Boolean model does accurately reproduce the behaviour of the comparator. This is a powerful advantage offered by ABCD-NL – anything that SPICE can predict, the Boolean model can incorporate.

Finally, Fig. 13 (d) shows that ABCD-NL accurately reproduces the behaviour of the comparator when it is embedded in a typical SAR-ADC environment. The ADC, and the input to it, are the same as in Fig. 11. Over one time period of the input waveform (*i.e.*, 52 ADC samples), the top half of Fig. 13 (d) plots the comparator inputs V_+ and V_-. The bottom half of the figure shows that ABCD-NL is able to capture the system's response, almost to SPICE-level accuracy, over this entire time frame.

E. Formal verification with an ABCD-NL model

As we remarked before, the main focus of this paper is the *accurate modelling of AMS components for verification*, as opposed to the verification itself. However, for completeness, we now present an example where we Booleanized an AMS system using ABCD-NL, imported the resulting Boolean model into a verification engine (ABC, [1]), and carried out formal property checking of the model against an AMS-design relevant specification.

Fig. 14. Schematic of a system that was formally verified using a combination of ABCD-NL and ABC. The inverters were designed in a 22nm CMOS process, using BSIM4 models. The channel was modelled as a long RC chain.

Fig. 14 depicts the system that we formally verified. It follows the same pattern as the systems shown in Figs. 1 and 7. As we mentioned earlier, such systems play an important role in SI applications, where it is important to determine, and *formally verify*, the throughput of the system. We have used 22nm BSIM4 models for each transistor in the system, and an analog channel that consists of several RC units chained together.

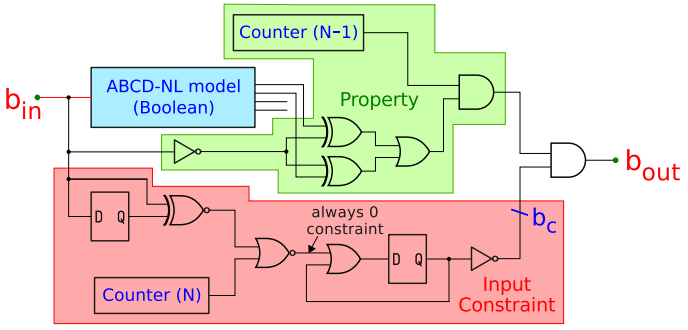

Fig. 15. Encoding the throughput property, along with constraints on the input, in a Boolean form so as to formally verify the ABCD-NL model using ABC [1].

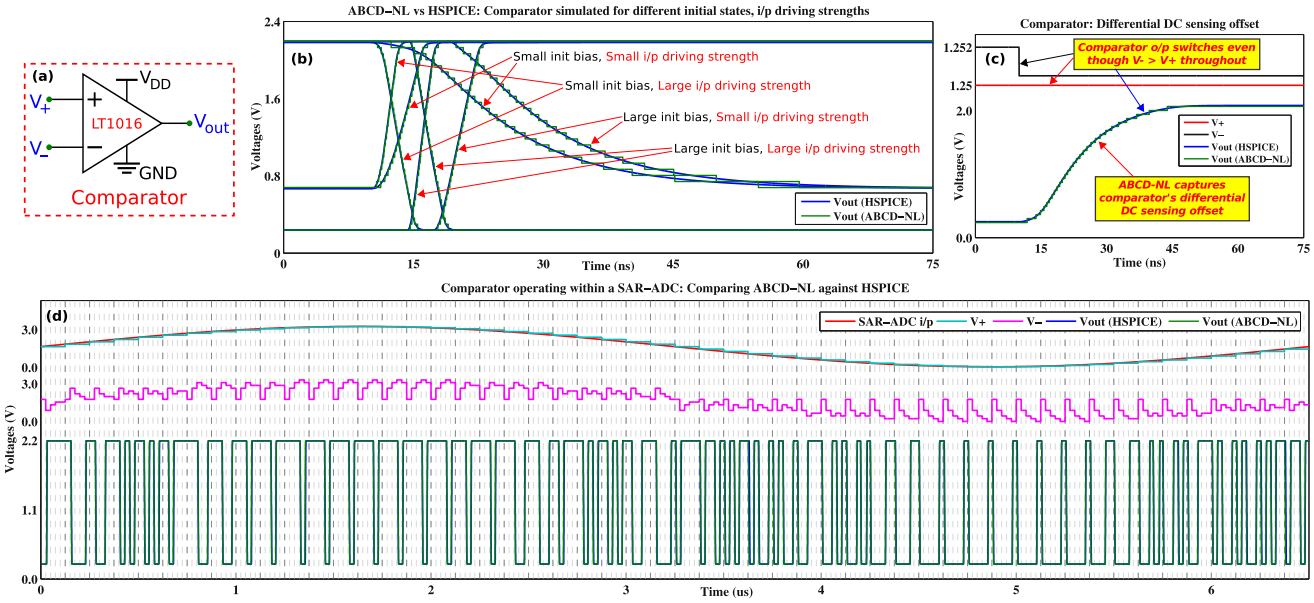

Fig. 13. ABCD-NL applied to a Linear Technology LT1016 comparator (part (a)). Part (b) shows that ABCD-NL is able to capture the dynamics of the comparator over a wide range of differential input excitations (from 1mV all the way to 1V). Part (c) shows that ABCD-NL can duplicate the SPICE-level dynamics of the comparator even when there is serious departure from ideal behaviour. Part (d) demonstrates that ABCD-NL is well-suited to model the comparator in the context of a SAR-ADC.

Fig. 15 shows the verification flow that we used. As the figure shows, the Boolean circuit that is verified consists of three parts, (1) the ABCD-NL Boolean model, (2) the Boolean logic that encodes the property to be checked, and (3) some Boolean logic to encode constraints on the inputs that can be applied to the AMS system. The circuit of Fig. 15 is constructed in such a way that the given AMS design fails to meet its throughput specification if and only if the bit b_{out} can somehow be asserted to 1 by choosing an appropriate sequence of bits applied at b_{in}.

The constraint on the input is that it can change only once per N clock periods of the ABCD-NL model. The time period of ABCD-NL's sequential Boolean model is 10ps (see Algorithm 1). The input constraint is modelled using a counter that outputs a 1 every N clock cycles (Fig. 15). If this constraint is violated, the bit marked b_c immediately becomes 0 and stays there forever, which makes it impossible to assert b_{out} to 1. This ensures that any counter-example returned by ABC would satisfy the input constraint.

The throughput property to be checked is that, given the above constraint on the input, the output should always reach an acceptable state before N clock cycles (*i.e.*, before the input can change). This acceptable state is defined as being ≥ 0.8V for a 1, and being ≤ 0.2V for a 0.

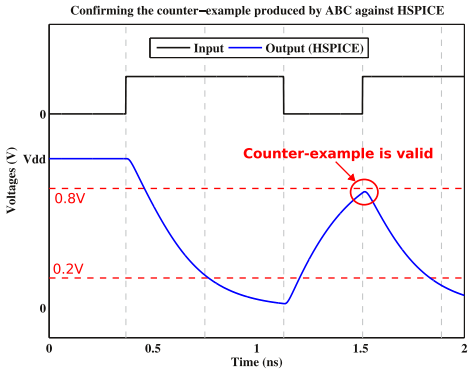

Fig. 16. Checking that the counter-example returned by ABCD-NL + ABC is valid in the analog domain.

Clearly, there is an N_0 such that the above throughput property will fail for all $N \leq N_0$. We can use ABC to quickly zero in on N_0, by incorporating ABC-based verification within a binary search loop. In this way, we were able to determine that $N_0 = 38$. This translates to a throughput of approximately 2.56Gbps. Furthermore, we were also able to confirm, using HSPICE, that for $N = 38$, the counter-example returned by ABC is a valid one in the analog domain (Fig. 16). Therefore, the throughput bound obtained is tight and meaningful. Thus, ABCD-NL is a powerful and capable modelling technique that can be used for AMS verification.

IV. CONCLUSIONS

In conclusion, we have developed and demonstrated ABCD-NL, a new technique for producing purely Boolean models of non-linear analog circuits, suitable for AMS verification. We have applied ABCD-NL to several circuits that are of interest to AMS designers, including charge pumps, signaling/communications sub-systems, D/A converters and comparators used in SAR-ADCs, *etc.* In addition, we have demonstrated a formal verification example where we used ABCD-NL, in conjunction with ABC, to formally verify the throughput of an AMS system.

REFERENCES

[1] R. K. Brayton and A. Mishchenko. ABC: An academic industrial-strength verification tool. In *CAV '10: Proceedings of the 22nd International Conference on Computer Aided Verification*, pages 24–40, 2010.

[2] R. Parker. Analog design challenges in the new era of process scaling. At the 2012 International Workshop on Design Automation for AMS Circuits (co-located with ICCAD).

[3] W. Hartong, L. Hedrich, and E. Barke. Model checking algorithms for analog verification. In *DAC '02: Proceedings of the 39th annual ACM Design Automation Conference*, pages 542–547, 2002.

[4] S. Steinhorst and L. Hedrich. Model checking of analog systems using an analog specification language. In *DATE '08: Proceedings of the ACM Conference on Design, Automation and Test in Europe*, pages 324–329, 2008.

[5] G. Al-Sammane, M. H. Zaki, and S. Tahar. A symbolic methodology for the verification of AMS designs. In *DATE '07: Proceedings of the ACM Conference on Design, Automation and Test in Europe*, pages 249–254, 2007.

[6] S. Little. *Efficient Modeling and Verification of Analog/Mixed-Signal Circuits Using Labeled Hybrid Petri Nets*. PhD thesis, University of Utah, 2008.

[7] M. Althoff, A. Rajhans, B. H. Krogh, S. Yaldiz, X. Li, and L. Pileggi. Formal verification of Phase Locked Loops using reachability analysis and continuization. In *ICCAD '10: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 659–666, 2010.

[8] S. Steinhorst and L. Hedrich. Trajectory-directed discrete state space modeling for formal verification of nonlinear analog circuits. In *ICCAD '12: Proceedings of the International Conference on Computer-Aided Design*, pages 202–209, 2012.

[9] T. A. Henzinger, P. H. Ho, and H. Wong-Toi. HyTech: A model checker for hybrid systems. *International Journal on Software Tools for Technology Transfer*, 1(1):110–122, 1997.

[10] E. Asarin, T. Dang, and O. Maler. d/dt: A verification tool for hybrid systems. In *CDC '01: Proceedings of the 40th IEEE Conference on Decision and Control*, pages 2893–2898, 2001.

[11] A. Chutinan and B. H. Krogh. Computational techniques for hybrid system verification. *IEEE Transactions on Automatic Control*, 48(1):64–75, 2003.

[12] G. Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. *International Journal on Software Tools for Technology Transfer*, 10(3):263–279, 2008.

[13] C. Tomlin, I. Mitchell, A. M. Bayen, and M. Oishi. Computational techniques for the verification of hybrid systems. *Proceedings of the IEEE*, 91(7):986–1001, 2003.

[14] M. Althoff, O. Stursberg, and M. Buss. Computing reachable sets of hybrid systems using a combination of zonotopes and polytopes. *Nonlinear Analysis: Hybrid Systems*, 4(2):233–249, 2010.

[15] A. V. Karthik and J. Roychowdhury. ABCD-L: approximating continuous linear systems using Boolean models. In *DAC '13: Proceedings of the 50th Design Automation Conference*, pages 63:1–63:9, 2013.

[16] C. Gu and J. Roychowdhury. FSM model abstraction for analog/mixed-signal circuits by learning from I/O trajectories. In *ASP-DAC '11: Proceedings of the 16th Asia and South Pacific Design Automation Conference*, pages 7–12, 2011.

[17] K. V. Aadithya and J. Roychowdhury. DAE2FSM: Automatic generation of accurate discrete-time logical abstractions for continuous-time circuit dynamics. In *DAC '12: Proceedings of the 49th Design Automation Conference*, pages 311–316, 2012.

[18] http://ptm.asu.edu/modelcard/HP/22nm_HP.pm.

[19] http://www.analog.com/Analog_Root/static/techSupport/designTools/spiceModels/license/spice_general.html?cir=AD8079A.cir.

[20] http://ltspice.linear.com/software/LTC.zip.