A high-level overview of "Accurate Booleanization of Continuous Dynamics for Analog/Mixed-Signal Design"

Abdus Salam Azad

January 9, 2018

1 Introduction

This report inspects the dissertation titled "Accurate Booleanization of Continuous Dynamics for Analog/Mixed-Signal Design". The dissertation investigates the problem of modelling Analog/Mixed-Signal (AMS) systems for BUG-free AMS Design. One of the core challenges in designing AMS systems is to choose a model that can accurately capture the properties of the underlying system and is also amenable to analysis and fast verification. Traditionally, the analog and digital circuits have been modelled accurately using DAEs and FSMs (or, similar discrete representations) respectively—two quite different languages which do not mix well. However, the existing modelling techniques for the AMS systems (which contain both analog and digital components closely interacting with each other) lack the desired accuracy (for simplifying assumptions) and scalability (limited to only a few continuous variables) for real-world practical AMS systems. The dissertation presents a method called "Booleanization" that can accurately model AMS systems with purely Boolean representation (e.g., FSMs). Boolean representations/models have a clear benefit as they allow the AMS systems to be verified by existing state-of-the-art Boolean verification tools, which can verify large industry level real-world digital systems. The dissertation proposes three different techniques of "Booleanization", among which ABCD-NL is the most general and powerful.

The rest of this report is organized as follows. Section 2 provides an overview of ABCD-NL, while, Section 3 investigates its time and space complexity. Section 4 discusses how machine learning can be used as a more scalable alternative to model AMS systems. At last, Section 5 concludes the report.

2 Overview of ABCD-NL

ABCD-NL is a "Booleanization" technique, that takes an AMS system as an input (as a SPICE netlist or, a system of DAEs) and constructs an FSM as a Boolean approximation of the underlying system to accurately capture its I/O behavior. As the first step of the Booleanization technique, ABCD-NL first discritizes the given circuits inputs and outputs with a finite number of bits. In the next step, the FSM is constructed. ABCD-NL works under the assumption that, a DC input to the given system should eventually result in a DC output (DC-DC assumption). Hence, upon a change of input the system first shows a transient behavior for a while and eventually settles into a constant DC voltage. To accurately capture the behavior of both the DC and transient behavior of the underlying system, the ABCD-NL-constructed FSMs feature two distinct type of states: 1) DC states and 2) Transient states (between each pair of DC states). Each DC state correspond to a distinct combination of the input symbols and represent the DC operating point of the system for that input. When the input changes and kept fixed (constant) for a 'reasonable' amount of time, the FSM moves from one DC (or, transient too [1]) state into another DC state and converges [2]. It is on the path to the DC states, the transient states are visited and the appropriate transient output behavior is exhibited.

The transition between different states is triggered by a change of input (both in DC and transient states) or, by the advancement of time (while in a transient state). These transitions are represented by three different types of arcs: 1) self-looping arcs of the DC states, 2) the outbound

[1] If an input is changed, while the underlying system is still in a transient state due to a previous change in input, the transition to a DC state can start from a transient state, too.

[2] In case of a constant input kept fixed for a 'reasonable' amount of time, the corresponding FSM converges into one of the DC states, similar to the corresponding DAEs it is approximating. This 'convergence' in the FSM is achieved by a self-loop in the DC states. The self-loop is labeled by the corresponding constant input.

arcs of the transient states leading them from one DC state to another, and 3) the outbound arcs of the transient states corresponding to input changes during the transient state of the system. ABCD-NL utilizes SPICE simulations of the underlying AMS systems to determine the output labels of the arcs to correctly portray the desired I/O behavior. Determining the output-labels of the self-looping arcs to the DC states are straight-forward. The input value corresponding to the arc (from the input label) is fed to the SPICE simulator and the discretized DC (steady) output is used as the label. For labeling the arcs of the transient states between two DC states e.g., DC1 and DC2, ABCD-NL simulates the transient behavior of the system in SPICE by feeding it a step input—a sharp transition from the DC level of DC1 to the DC level of DC2. Afterwards, a number of transient states (the number is determined using the estimated settling time and FSM time step) are inserted between the two DC states and their output labels are determined from the trajectory of the simulation output. For constructing and appropriately labeling the third kind of arcs, 'Jumping' heuristics are used. A jumping heuristic finds a 'similar' state for the transient state in consideration. The next state and output label of the 'similar' state are used by the transient state.

3 A Closer Look on ABCD-NL

Accuracy and scalability are two of the most important aspects of AMS modelling. While, ABCD-NL can accurately capture the I/O behavior of the underlying AMS systems, the issue of scalability can still arise. Consider an AMS system with 2 inputs and we want to discretize each input with 16 bits. The number of DC states in the FSM will be $2^{2\times16}$. In general, if there are n input signals and we discretize each with b bits, the total number of DC states in the corresponding FSM is 2^{nb}, i.e., the number of DC states is exponential to the number of bits required to discretize the input signals. The number of the transient states are also exponential to the number of input bits. As there are 2^{nb} distinct DC states, there are ${}^{2^{nb}}P_2$ (i.e., $(2^n)^2 - (2^n)$) distinct transient paths. If we consider t as the average number of states in the transient paths, the total number of transient states is $t \times {}^{2^{nb}}P_2$. Hence, the size of the FSM is exponential to the number of input bits nb and storing the model in memory may become infeasible for a moderate number of inputs and discretization level.

Now, the algorithm ABCD-NL proposes to construct the FSMs, takes each ordered pair of the DC states (i.e., the nested loop between Line 4 to 14 of Algorithm 1) to construct the transient paths between them. The number of iterations is hence, $(2^n)^2 - (2^n)$. For simplicity, if we assume that each such transient path takes a constant time to be constructed, the nested loop is of exponential order i.e., $O((2^n)^2 - (2^n))$ or, simply $O(4^n)$. The other nested loop that the algorithm features between Line 15 to 19, is also of exponential order, i.e., $O((t \times (2^n)^2 - (2^n)) \times 2^{nb})$ or, simply $O(2^{n(2+b)})$. Hence, the overall time complexity of the algorithm is of exponential order, which limits the capability of ABCD-NL by limiting the number of inputs and discretization levels it can work with.

4 A Machine Learning Based Approach

The ABCD-NL algorithm, in essence, 'saves' snapshots of the output values (e.g., voltages) of the underlying system at a number of 'interesting' states and the transition among these 'interesting' states capture the dynamics of the system. However, this process is similar to memorizing information into a knowledge base without any generalization. An alternative way to model the AMS systems can be using machine learning techniques, which can generalize the underlying I/O behavior within compact representations. This report now briefly outlines an approach, which is believed to be worth exploring, to serve this purpose.

In this preliminary approach, the DC-DC assumption is made similar to ABCD-NL. The model will learn the function, $\vec{y_t} = f(\vec{x_{t_{0-}}}, \vec{x_{t_{0+}}}, \vec{y_{0-}}, t)$ to approximate the I/O behavior. Here, t_0 marks the time when the last input change had occurred and t is the time elapsed after t_0. $\vec{x_{t_{0-}}}$ and $\vec{x_{t_{0+}}}$ are the previous and current input, respectively. $\vec{y_{0-}}$ is the value of output just before the input change had occurred. The inclusion of $\vec{y_{0-}}$ as an input enables the function to capture the appropriate behavior when an input signal is changed during transient states. $\vec{y_t}$ is the output at time t, which the model will predict. With this formulation we can capture both the DC and transient behavior of the system. The DC behavior of the circuit is portrayed at $t = \infty$, while the transient behavior is shown at t within $[0, \infty)$. In simple words, the model will predict the output of the system, at a certain time t after a change in the input, using the previous input, current

input, and previous output. Now, there are a number of machine learning techniques to learn this function. Among them Artificial Neural Networks (ANN) may be a well suited technique for this task. Some of the appealing properties of this proposed direction are discussed below:

- ANN does not need to save every possible input combination. Rather, it will need one input node per input signal. In addition to that, ANNs can handle both discrete and continuous-valued inputs together— a suitable property for AMS systems[3]. However, it is also possible to discretize all the continuous-valued input signals if necessary. While discretizing the inputs before the training phase is straight-forward, training the model with continuous values but using discrete values during prediction phase can be interesting. On the other hand, the output is typically continuous, which can also be discretized if needed. It is also worth mentioning that, even if we use one single input node per input signal (e.g, one input node portraying 16 discrete values between 0 to 5 volt, instead of 4 binary input nodes), the models can still be verified by boolean verification techniques. For example, we can start with boolean inputs and convert them appropriately before feeding into the network.
- Theoretically, feed-forward neural nets are universal approximators [1]. With a single, sufficiently large hidden layer, it is possible to represent any continuous function of the inputs with arbitrary accuracy. However, the required number of hidden units will grow exponentially with the number of inputs. In practice, ANNs have shown to closely approximate complex linear and non-linear functions.
- The training samples for the ANN can easily be generated from SPICE simulations.
- Each evaluation of the function can be done with one forward pass, which takes constant time.

Recurrent Neural Networks(RNN), a special kind of ANNs, are also worth exploring as a model for the AMS systems. RNNs has a notion of memory or state, unlike the traditional ANNs, which allow them to 'remember' information about what it has calculated so far to predict about the new input. Theoretically, with enough sigmoidal hidden units, RNNs can approximate any non-linear dynamical system to any accuracy, with no restrictions on the compactness of the state space [2]. The analogy between RNNs and FSMs has also been explored in the literature([3] and [4]).

5 Conclusion

To summarise, this report takes its view on ABCD-NL—an accurate modelling technique for AMS systems. ABCD-NL constructs an FSM to accurately capture the underlying I/O behaviour. However, it suffers in terms of scalability, another important property of the AMS models. The report eventually proposes neural networks as a modelling alternative. Proper investigation of neural networks may result in an accurate and scalable AMS modelling technique.

References

[1] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[2] A. M. Schäfer and H. G. Zimmermann, "Recurrent neural networks are universal approximators," in *International Conference on Artificial Neural Networks*, pp. 632–640, Springer, 2006.

[3] P. Tino, B. G. Horne, and C. L. Giles, "Finite state machines and recurrent neural networks–automata and dynamical systems approaches," tech. rep., 1998.

[4] M. Casey, "The dynamics of discrete-time computation, with application to recurrent neural networks and finite state machine extraction," *Neural Computation*, vol. 8, pp. 1135–1178, Aug 1996.

[3]The inputs will still be discretized in time.