Mixed Signal Design Validation Using Reinforcement Learning Guided Stimulus Generation for Behavior Discovery

Barry Muldrey, Suvadeep Banerjee, and Abhijit Chatterjee
Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA
[bmuldrey3, sbanerjee49]@gatech.edu, chat@ece.gatech.edu

***Abstract*—High operating speeds and use of aggressive fabrication technologies necessitate validation of mixed-signal electronic systems at every stage of top-down design: behavioral to netlist to physical design to silicon. At each step, design validation establishes the equivalence of lower level design descriptions against their higher level specifications. Prior research has leveraged state reachability analysis, nonconvex optimization, or performance specifications in order to generate tests. In contrast, we reformulate the systems under validation as a Markov decision process and examine the use of reinforcement-learning to provide a globally convergent solution, a means of "storing" the valuable information created during stimulus generation, and low-cost iterated generation. The integration of the proposed design validation methodology with deep-Q learning software and the suite of Cadence simulation tools is presented, validation results for selected design bugs in representative designs are analyzed, and the quality and efficiency of the proposed design validation methodology is discussed.**

I. Introduction

In hierarchical system design (SoCs, SoPs), models for mixed-signal components (e.g. regulators, data converters, I/O, RF) are described at the behavioral level (Simulink or MATLAB) with the complete system described as an interconnection of these models. The design strategy is typically top-down with bottom-up verification of synthesized netlists and physical layout. The design is incrementally advanced and verified with the intent of ensuring design correctness through each iteration of the design process from high level design specification to physical layout to silicon. This prevents expensive correction of design bugs that percolate from early stages of the design process to later steps including fabrication of silicon. To facilitate rapid turnaround design, it is necessary to verify behavioral equivalence between a higher level specification of the design (e.g. AHDL) and its lower level implementation (e.g. netlist) as early as possible.

While the analog specifications of modules at two different levels of the design can be checked for equivalence, such a check does not guarantee behavioral equivalence across the entire space of input stimuli because the set of specifications used to check for equivalence may itself be *incomplete* [3], [4]. For similar reasons, formal methods for design equivalence checking may fail because designer inserted assertions may not cover all input-output behaviors and are generally myopic and constrained to specific input conditions.

II. Prior Work

The use of rapidly exploring random trees (RRT) for analog circuit test generation was presented in [5], [6]. The idea was to quickly explore reachable points in the state space of the analog circuit for verification purposes. In [7], an efficient discretized state space guided test stimulus generation approach is proposed with the goal of equivalence property checking. The methodology combines formal methods with circuit simulation techniques. In a similar vein, the equivalence between a behavioral model and its transistor level design over a highly likely input stimulus space is discussed in [8]. The discrepancy between the two design descriptions over the space of possible input stimuli is maximized to detect design errors. The work of [9]–[11] was among the first to combine formal verification methods with simulation driven techniques to explore the limits to which analog design specifications can be stressed, but assumes tests that are derived from manually crafted specifications. Central challenges going forward include eliminating up-front assumptions of input distribution (i.e. "kinds of input") and assumptions of completeness of any set of provided specifications.

III. Motivation

The purpose of this work is to present a new reinforcement learning (RL) driven test generation and anomaly-model generation algorithm that does not require any a-priori knowledge about: stimuli to be used for design verification/validation, the device specifications, or properties to be verified. Additionally, the RL techniques used have been shown to be convergent upon the global optimum under certain conditions and internally contain a useful distillation of all previous information (they have memory) and so can be reused and retrained many times and in many contexts at higher efficiency than optimization alone [12].

In Section V, we briefly review the subject of RL and its relationship to other machine learning (ML) techniques. In Section VI-A, we present our formulation of the analog and mixed-signal (AMS) system validation problem as a Markov decision process (MDP). In Section VI-B, we illustrate the direct applicability of contemporary RL techniques to validation and explore the possible benefits over existing techniques. In Section VII, we detail our implementation of deep Q-learning and discuss our experimental setups. We present experimental

978-1-7281-1170-4/19/$31.00 ©2019 IEEE

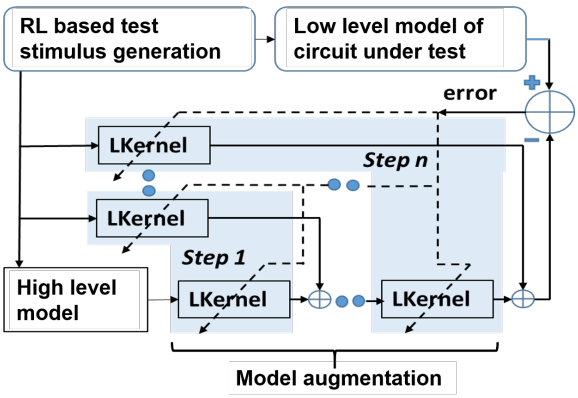

Fig. 1. Methodology overview

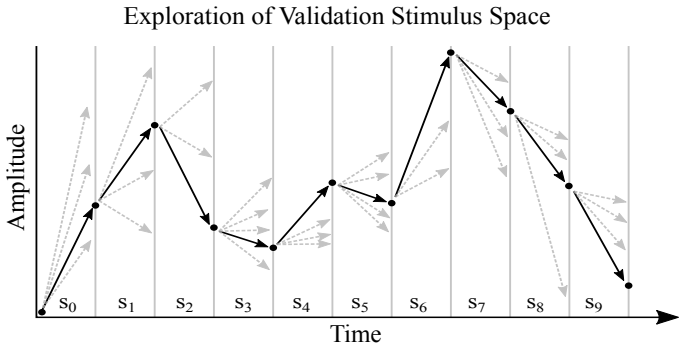

Fig. 2. Validation stimulus built as a succession of actions taken, each leading the system into a different state, S_i

results and provide a summary analysis before concluding the paper in Section VIII.

IV. Overview of Validation Methodology

For design validation it is necessary to attempt to excite differences between the high level design and its low level design description. We propose a directed stochastic test generation approach based on reinforcement learning which excites both the high level model as well as its low level description with the same stimulus while observing the difference (or error) signal between the two. While the focus of this paper is on test stimulus optimization, the end application is in behavior learning. This involves several test generation runs. After each run, the high (or low) level models are augmented with learning kernels in such a way that the error between them is minimized. After model augmentation, another round of test generation and model augmentation is performed (without undoing the augmentation performed earlier). The learning kernels in each round of augmentation are arranged in a "boosting" fashion [13]. This process is repeated until no further differential behaviors can be excited by the test stimulus generator. Figure 1 describes the above process. In the past, sparse wiener networks have been used as learning kernels [14]; recurrent neural nets can also be used as earning kernels. The ensemble of kernels collectively describes behaviors induced by logical and electrical bugs in the design (electrical bugs are harder to model in simulation but can be excited in fabricated silicon). The behaviors induced by multiple design bugs may be characterized in this manner and passed to the circuit designer for further analysis.

V. Review of Machine- and Reinforcement-Learning

"Machine Learning" refers broadly to the study of statistical methods that leverage the power of computing hardware to predict attributes of future observations by making inferences from past observations, the "learner's" ability to do so is predicated on exposing the learner to a volume of historical data which is representative of future observations. The categories of tools that comprise machine learning are divided into two main classes: those that are "supervised" and those that are "unsupervised." A supervised approach refers to the requirement for human intervention in the preparation of training data. By contrast, "Unsupervised" algorithms do not require human intervention. Classically, unsupervised techniques consist mainly of data clustering algorithms.

A. Reinforcement Learning

In the early 1990s, work on a third class of machine learning algorithms called "reinforcement learning" began to circulate. [15]. In RL, rather than require human intervention on the very granular datum-to-datum scale, one provides a means for the machine to evaluate its own performance and guide its own learning. The full potential of RL became evident with the advancement of parallel computation tools, and the work of Minh et. al. ([16]) wherein a RL learner was trained to play the Atari video game console with skill surpassing that of humans put RL squarely into the spotlight. Google's DeepMind team has recently enraptured the public once again with their exhibition of a self-taught RL player which has handedly taken the crown from the reigning champion, Stockfish [17]. The overarching goal of RL is for a machine to learn to be "good" at a task in a self-directed fashion through feedback about its performance received via the reward signal (discussed in Section VI-B).

B. Basics of (Deep/Double) Q-Learning

Q-Learning is so named because it centers around the approximation of a "quality" function,

$$q_{i+1} = Q(s_i, a_i) \tag{1}$$

which predicts the "quality" resulting from taking a certain action, a_i, given that the system is in state s_i. It requires that every intermediate action, a_i, receive a corresponding reward signal, r_i. In this work, circuit state is defined by a vector of node voltages and branch currents. Individual actions are defined as ramp inputs to the system which take the inputs of the circuit under test from an initial value, a_i, to a_{i+1} as shown in Figure 2. Starting from the state s_i, the RL takes an action (choice of input ramp) which will lead the system

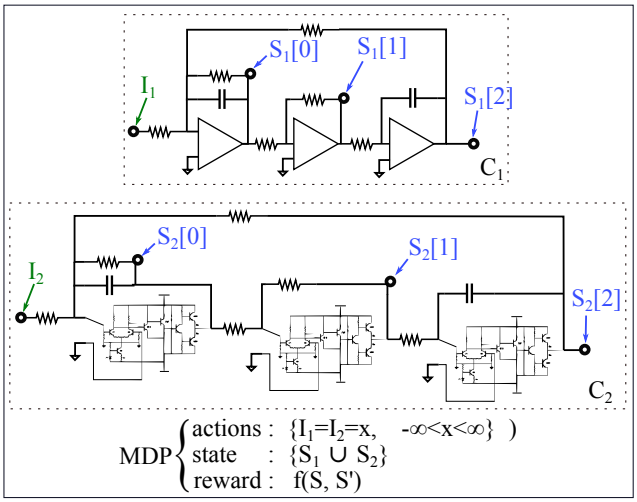

Fig. 3. Example of two circuits combined to create a validation MDP

into state s_{i+1}. We also define Q^*, the discounted sum of all n incremental rewards received during an episode:

$$Q^*(a_0, \ldots, a_n) = \sum_{i=0}^{i=n} \sum_{i}^{n} \gamma^i r_i \tag{2}$$

where γ is the factor of time-discounting, which takes a value on the range [0,1] (and is usually close to 1).

The goal of Q-learning is to accurately predict Q at each step so that optimal actions are selected, resulting in maximization of the expectation of Q^*. In [12], the algorithm is shown to be convergent to a global optimum. In order to correctly model Q however, one must explore the entire state-space of the system which can be infeasible even for modestly sized digital systems. So instead, Q-learning algorithms balance exploration and exploitation in effort to deliver acceptable performance in reasonable time.

"Deep Q-learning" is simply an implementation of Q-learning which leverages a so-called "deep" neural net (one which has a number of layers in series performing sequential abstractions from observation space to inference space) to implement the Q function. Double-Q learning is another variation in which separate networks are used for action selection and Q estimation which can avoid systematic bias introduced by guiding learning with the same network that is doing the learning.

In this work, we employed DQN algorithms as described by Minh, et. al. in [16] through open source implementations provided by OpenAI in their Baselines framework [18] as well as from the "stable-baselines" package [19].

VI. Stimulus Generation for Behavior Discovery

A. Circuit-pairs as Markov Decision Processes

In this work we formulate the validation of a system as a Markov decision process (MDP): a state machine which transitions probabilistically as a function of input and state, with the transition probability functions being stationary (memoryless). As illustrated in Figure 3, we compose such a system by exercising two systems in unison and observing their corresponding states. So state sets $\mathcal{S}_1$ and $\mathcal{S}_2$ must contain at least one node voltage or branch current which corresponds behaviorally (non-null intersection). The state of the MDP, s, is then the union of all pairs of states which $\mathcal{S}_1$ and $\mathcal{S}_2$ have in common, and the action inputs to our MDP map to a circuit input shared by the systems. Let m represent the number of *pairs* of states. Figure 2 illustrates the progression of a circuit through time conceptualized as as sequence of state transitions resulting from a particular input sequence. In some cases where only a subset of circuit states are observable, previous values of input (up to the system's memory depth) can be used as a proxy for unobservable states. In such cases, a system with memory-depth N can be made to seem Markov to an observer if the previous N inputs are treated as states.

B. Validation and RL Reward

Reinforcement learning introduces to MDPs the notion of "reward." Reward is a measure of the favorability of an individual transition from one state to another. In reinforcement learning, the algorithm's goal is to exercise the MDP environment by carefully manipulating its inputs in a way that maximizes the expected accumulated reward at the end of an experimentation period, or "episode." If we can devise a reward which aligns itself with our validation goals (to prove inequivalence, for example), then we can employ reinforcement learning techniques to find stimuli which maximize the likelihood of reaching our goal.

1) Designing the Reward Function: In each time step of the learning episode we perform a short transient simulation composed of many timesteps of its own, and so every RL timesetp results in $2 \times m \times k$ data points. Choice of time discretization should be made such that observations appear **at most** weakly nonlinear within any RL time step. The reward function, therefore, must accept a $2 \times m \times k$ matrix as input and produce a scalar reward.

In validation, we are interested in the distance between the states of the two systems. There are many distance metrics which might be suitable in helping produce a scalar reward: L-2 norm, Manhattan, cosine, cross-correlation, Mahalanobis, Wasserstein, etc.; analysis of the relative merits of each is an area of active research and must be assessed on an individual basis [20]. In this work, we have chosen the L1 norm of the time-domain difference waveform as the reward metric for reinforcement learning. In experiment Section VII-A, the RL receives the reward only when its value surpasses that of any reward received up to that point in the episode (generation session); experiments Section VII-B and Section VII-C receive each reward regardless of history.

VII. Experimental Results

Brief Note on Hyperparameters: Initial prototyping was performed in Matlab using a Double-Deep-Q architecture built from scratch. The effort yielded mixed success (Figure

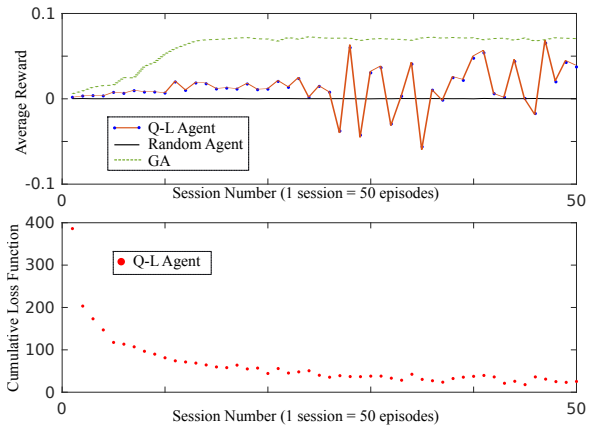

Fig. 4. Results of initial prototype experimentation illustrating hyperparameter sensitivity (learning-rate-induced instability)

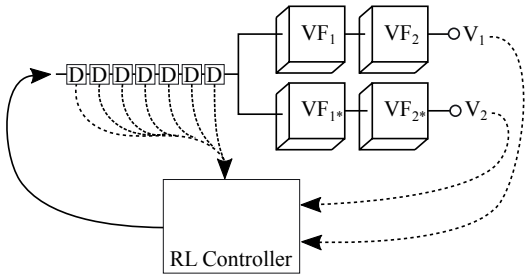

Fig. 5. Block-level schematic of the experimental setup in Section VII-A

4) mostly due to the abundance of hyperparameters in the algorithm and implementation-dependent subtleties. Because of our inability to achieve consistently good results, we turned to OpenAI Baselines, and Stable-Baselines for more robust implementations [19].

A. Volterra Models in Python

Two pairs of Volterra filters were created in a Python environment with identical coefficients drawn from a standard normal distribution. In one set of filters, two coefficients were chosen for a large binormally-distributed perturbation (+/- $0.2/\sigma$ and $\mu = 0.2/\pi$) and 10 coefficients were chosen for small perturbations (+/- $0.002/\sigma$ and $\mu = 0.002/\pi$). The coefficients of each filter were then normalized such that no coefficient had a magnitude larger than 1. We obscured the internal states of the filters from the learning algorithm, and kept the 7 most recent historical inputs as proxy states.

The stimuli were limited to 20 samples in length, corresponding to 20 steps per episode. The learning rate, α was 0.5e-4, the discounting of future value, γ, was 0.99, and the policy was ϵ-greedy with ϵ decaying exponentially from 0.1 to 0.02 over the course of 500k steps. The system was explored in batches of 32 episodes constituting a session. After each session, the network (sizes varying from 8 neurons to two layers of 64 neurons each) was retrained to approximate the updated Q values from the most recent 50k sessions.

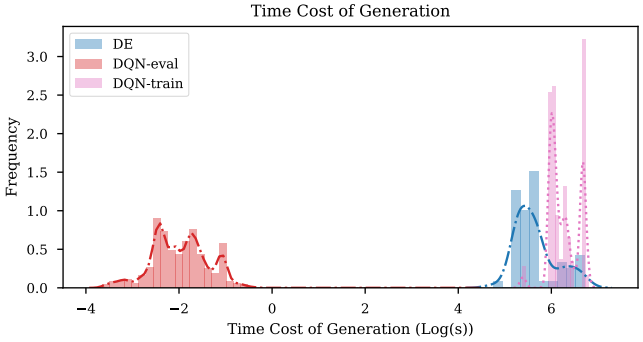

Fig. 6. Comparison of Time Costs

TABLE I
DQN V. DE: SELECTED STATISTICS

Mean DE generation time	316.4s
Mean DE reward	8458.2
Mean DQN training time	404.1s
Mean DQN evaluation time	0.17s
Mean DQN reward	3347.4

Figure 6 shows a comparative study of q-learning implementation against a differential evolution implementation provided by the Scipy library.

It can be seen that the DQN stimulus generation algorithm can repeatedly synthesize stimuli at a fraction of the cost of DE once it's trained.

B. Programmable Gain Amplifier in Spectre/OpenAI

Our second experimental platform was built in Pythton 3.5 and Tensorflow 1.5 and utilized the OpenAI Baselines library [18]. The implementation of DQN was based on [21].

The systems under validation were Cadence Spectre simulations. Each circuit was implemented in its own netlist, and instances of the Spectre binary were run in parallel and interacted with dynamically through our own "circuitgym," "pyspectre," and "libpsf" python libraries [22]–[24]. In this fashion, the circuits' inputs are manipulated and short time step transient simulations are performed.

The system under test is a 2-bit DAC feeding directly into a programmable gain amplifier with a 2-bit gain control. One version utilizes a more detailed and less idealized AHDL model for the amplifier core. Inputs are updated at 333ns intervals, and transient simulations of 333ns duration occur between each RL time step. After each time step, Spectre returns transient waveforms for all the transitional dynamics that are observable.

We exposed all 102 internal states which the two models had in common. This meant that the network must have an input dimension of (102,1). The deep Q-learning model implemented a discrete action space, and so each full input vector was assigned a probability of inducing a particular action. The circuit has 4 discrete input bits, corresponding to two 2-bit DACs. The interface between the Q-learner and

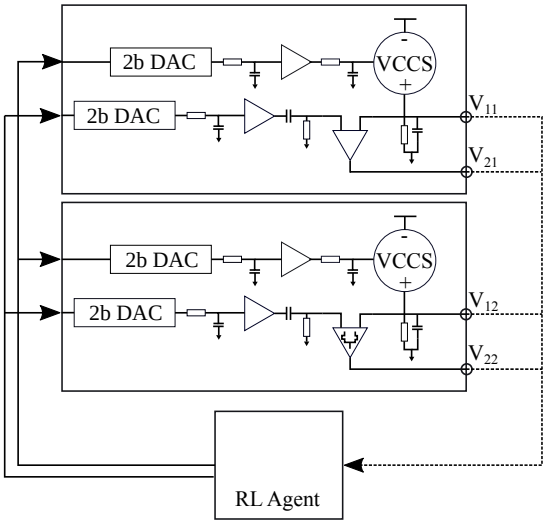

Fig. 7. Block-level schematic of the system under validation in Section VII-B

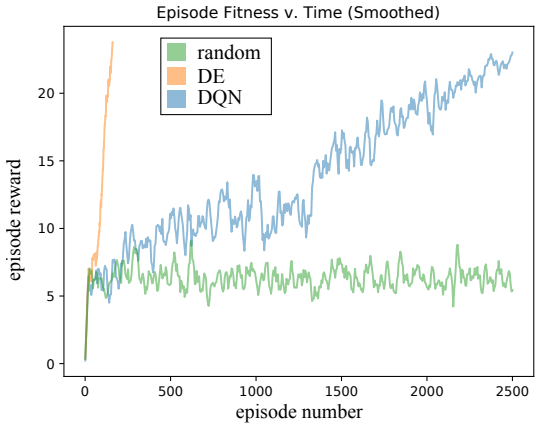

Fig. 8. Evolution of rewards from different actors in PGA environment (Section VII-B)

the circuit was made by considering each full 4-bit vector to be a discrete action. Thus the output layer of the circuit required 16 neurons, corresponding to the 16 possible inputs. The network was configured to have two hidden layers of 64 neurons each; This configuration lead to a total of 11,793 trainable parameters in the neural model.

Three test benches were built for this circuit: a random actor, an off-the-shelf differential-evolution algorithm (DE) and our DQN algorithm. In each, a maximum of number of allowed steps was set to 12,500, corresponding to 2500 episodes, each of length 5. The results of the random actor exercise over time are shown in Figure 8.

A more in-depth look at how the off-the-shelf DE algorithm progresses reveals a pitfall; independent trials of the DE algorithm, though the DE algorithms may sometimes converge and terminated very quickly, result in high variance in performance. A majority of the solutions proposed by the DE algorithm are poorer in performance than that from the DQN

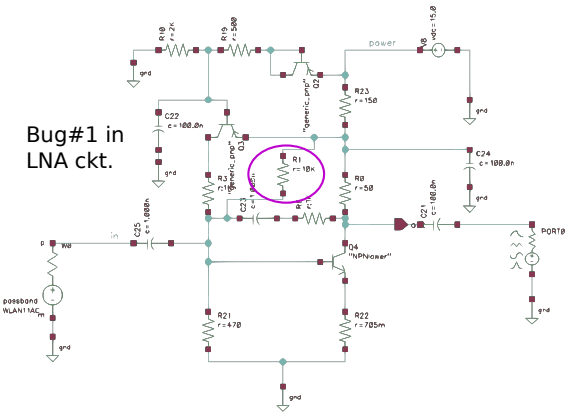

Fig. 9. Fault-injected LNA

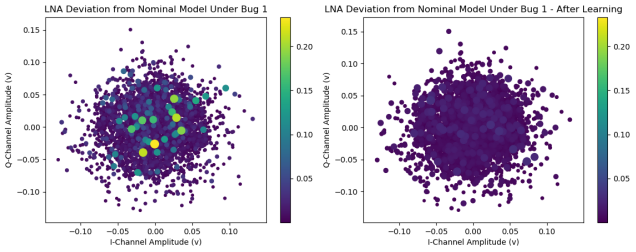

Fig. 10. Buggy model error before and after augmentation. Lighter colors represent higher-discrepancy observations

algorithm. This is a consequence of the possibility that the algorithm converges in a local solution. The DQN algorithm does not suffer from this risk because it has been analytically demonstrated to be globally optimal in the asymptotic case [12].

C. LNA Model Augmentation

In order to validate the model augmentation portion of the framework, we implemented a buggy low-noise amplifier in Cadence spectre by injecting a 10k-ohm resistor to couple two nodes which should not be directly bridged. Augmentation was performed by iteratively creating, training, and injecting SVM regressors into the bug-free model in order to reduce the observed behavioral difference. Like experiment Section VII-B, the experiment was built in Pythton 3.5 and Tensorflow 1.5 and utilized the OpenAI Baselines library [18].

The LNA circuit is shown in Figure 9; lna model discrepancies (buggy v. bug-free), before and after 20 rounds of augmentation are shown in Figure 10 with light color indicating observations of high-discrepancy. Figure 11 plots the evolution of the mean and kurtosis of model discrepancy over all observations over the course of iterated augmentation, indicating that not only is overall error reduced, but the number of spurious, relatively high-error events is also reduced.

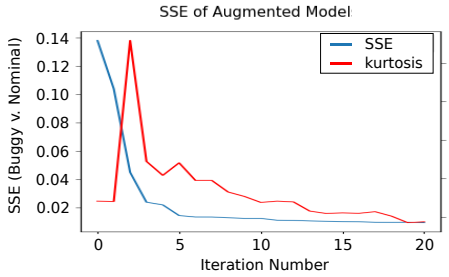

Fig. 11. Evolution of LNA augmentation while capturing buggy behavior.

VIII. Conclusion

We have presented the details of our state-of-the-art reinforcement learning algorithm which serves as the stimulus-generation subsystem in a design validation framework. We have designed and conducted several experiments which leverage Q-learning to generate a stimulus which is capable of revealing a maximal amount of information about the differences between two systems. Our work suggests that reinforcement learning provides a very powerful complement to tools like differential evolution and Monte Carlo stimulus generation.

While DQN is able to ultimately outperform the other methods, it comes at the cost of computational time (about 10x more). It however outperforms the optimization solver from a warm-start by a factor of roughly 1800x, suggesting that if appreciable restarts are foreseen, the benefits of RL may outweigh its up-front costs.

Some validation tasks can be accomplished with traditional specification tests (i.e. two-tone test) while others might require slightly more entropic stimuli like white noise in order to excite behaviors of interest. Circuit complexity, situation, or mission-criticality may mandate running the additional number of simulations required to train a high-performance agent to reveal the global solution, something that stochastic nonconvex optimizers cannot provide. We will continue this work, looking at broader classes of circuit and varying degrees of bug magnitude and nature. Additionally, we will explore the limits of RL's warm-start advantage.

Acknowledgment

The authors would like to thank the Semiconductor Research Corporation for supporting this research under GRC Task 2712.005. This project is funded under the Texas Analog Center of Excellence (TxACE) at UT Dallas.

References

[1] J. Keshava, N. Hakim, and C. Prudvi, "Post-silicon validation challenges: How EDA and academia can help," in *47th ACM/IEEE Design Automation Conference (DAC)*, pp. 3–7.

[2] M. G. . Prologue: The 2016 Wilson Research Group Functional Verification Study Verification Horizons BLOG. [Online]. Available: https://blogs.mentor.com/verificationhorizons/blog/2016/08/08/prologue-the-2016-wilson-research-group-functional-verification-study/

[3] A. DeOrio, Q. Li, M. Burgess, and V. Bertacco, "Machine Learning-based Anomaly Detection for Post-silicon Bug Diagnosis," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '13. EDA Consortium, pp. 491–496. [Online]. Available: http://dl.acm.org/citation.cfm?id=2485288.2485411

[4] C. Gu, "Challenges in Post-silicon Validation of High-speed I/O Links," in *Proceedings of the International Conference on Computer-Aided Design*, ser. ICCAD '12. ACM, pp. 547–550. [Online]. Available: http://doi.acm.org/10.1145/2429384.2429502

[5] S. N. Ahmadyan, J. A. Kumar, and S. Vasudevan, "Goal-oriented stimulus generation for analog circuits," in *Proceedings of the 49th Annual Design Automation Conference*. ACM, pp. 1018–1023.

[6] ——, "Runtime verification of nonlinear analog circuits using incremental time-augmented RRT algorithm," in *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, pp. 21–26.

[7] S. Steinhorst and L. Hedrich, "Improving verification coverage of analog circuit blocks by state space-guided transient simulation," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 645–648.

[8] A. Singh and P. Li, "On Behavioral Model Equivalence Checking for Large Analog/Mixed Signal Systems," in *Proceedings of the International Conference on Computer-Aided Design*, ser. ICCAD '10. IEEE Press, pp. 55–61. [Online]. Available: http://dl.acm.org/citation.cfm?id=2133429.2133440

[9] H. Hu, Q. Zheng, Y. Wang, and P. Li, "HFMV: Hybridizing formal methods and machine learning for verification of analog and mixed-signal circuits," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, pp. 1–6.

[10] L. Yin, Y. Deng, and P. Li, "Verifying dynamic properties of nonlinear mixed-signal circuits via efficient SMT-based techniques," in *Proceedings of the International Conference on Computer-Aided Design*. ACM, pp. 436–442.

[11] H. Lin, P. Li, and C. J. Myers, "Verification of digitally-intensive analog circuits via kernel ridge regression and hybrid reachability analysis," in *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*. IEEE, pp. 1–6.

[12] C. J. C. H. Watkins and P. Dayan, "Q-learning," vol. 8, no. 3-4, pp. 279–292. [Online]. Available: https://link.springer.com/article/10.1007/BF00992698

[13] A. H. Mirza, "Online boosting algorithm for regression with additive and multiplicative updates," in *2018 26th Signal Processing and Communications Applications Conference (SIU)*, May 2018, pp. 1–4.

[14] B. Muldrey, S. Deyati, and A. Chatterjee, "Post-silicon validation: Automatic characterization of rf device nonidealities via iterative learning experiments on hardware," in *2017 30th International Conference on VLSI Design and 2017 16th International Conference on Embedded Systems (VLSID)*. IEEE, 2017, pp. 403–408.

[15] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, ser. Adaptive computation and machine learning. MIT Press.

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," vol. 518, no. 7540, p. 529. [Online]. Available: https://www.nature.com/articles/nature14236

[17] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm." [Online]. Available: http://arxiv.org/abs/1712.01815

[18] Dhariwal, Prafulla, Hesse, Christopher, Klimov, Oleg, Nichol, Alex, Plappert, Matthias, Radford, Alec, Schulman, John, Sidor, Szymon, and Wu, Yuhuai, "OpenAI Baselines," GitHub. [Online]. Available: https://github.com/openai/baselines

[19] A. Hill, A. Raffin, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable Baselines." [Online]. Available: https://github.com/hill-a/stable-baselines

[20] C. C. Aggarwal, A. Hinneburg, and D. A. Keim, "On the surprising behavior of distance metrics in high dimensional space," in *International conference on database theory*. Springer, 2001, pp. 420–434.

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning." [Online]. Available: http://arxiv.org/abs/1312.5602

[22] Circuitgym. [Online]. Available: https://pypi.org/project/circuitgym/

[23] Pyspectre. [Online]. Available: https://pypi.org/project/pyspectre/

[24] Libpsf. [Online]. Available: https://pypi.org/project/libpsf/