Capstone Proposal:
Machine Learning for Functional Verification

Alberto Guasco

March 20, 2019

The project I will develop for concluding the Nanodegree concerns the usage of machine learning techniques for improving functional verification. My goal is to understand if a highly dimensional system such as a generic piece of hardware can be verified more efficiently using intelligent stimulus generation.

Background In the RTL design flow, functional verification is a key step because it requires considerable efforts and ensures the manufacturer to have a product that respects all the specifications and functionalities designed. The verification is considered completed if we can estimate that on the device under testing (hereafter DUT) all checkers have not failed and the DUT has been fully stimulated. Some years ago it was feasible to detail all possible DUT's states and then check one-by-one if they have been reached almost once, or precompute which series of stimuli are needed for reaching a certain state. Right now for more complex DUT (such as modern microprocessors) these scenarios are far from reality. The complexity of DUT is increasing (following Moore's Law), it's really difficult to list all possible states and find an efficient way to stimulate them in order to complete the verification in the shortest possible time. Differently from the problems seen during the Nanodegree, this one doesn't describe any daily issue. Anyway it really motivates me as I worked on this domain: I experienced this problem and I'm sure that machine learning will become a standard for solving it. Moreover it will be interesting to see if the results of this work can be applied on other kind of problems.

Problem Statement Given a known DUT and given the list of states that the DUT has to reach, we propose to use machine learning for stimulate the DUT in order to reach all states in the shortest possible time. The DUT is an ensemble of sequential and combinatorial logic: we force some binary values in the inputs, the internal registers change their states and the outputs are the results of operations done on the registers values. In figure 1 is shown a generic model of a DUT. Another way for generalizing it is a Moore's Machine (as in figure 2): for a given DUT we can group all sequential logic (called flip-flop or registers) on a single state, and all the combinatorial logic as the functions for moving between states. Note that in our problem we do not focus on the output: we are not trying to predict a certain output but we want to learn how to visit all states of the DUT in the shortest possible time.

Datasets and Inputs Functional verification is normally done using Electronic Design Automation (EDA) software for performing verification and for measuring DUT's coverage. For this project we can't use this approach because these softwares are under license and because a single test requires a significant amount of time, due to the modelization of the hardware DUT. For overcoming this problem we will generate a generic and parametrizable model of a DUT: we will design a python function taking as input the number of states, the number of inputs, and will

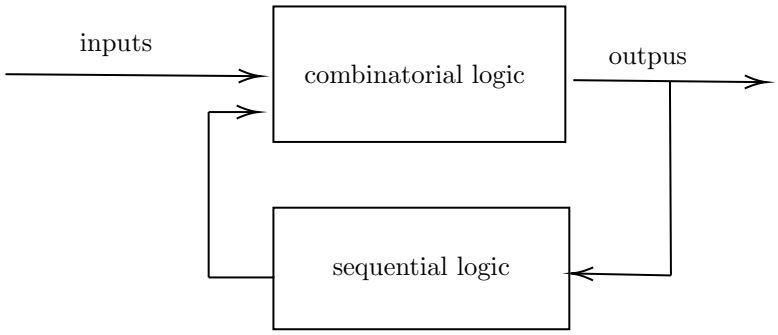

Figure 1: DUT generalization

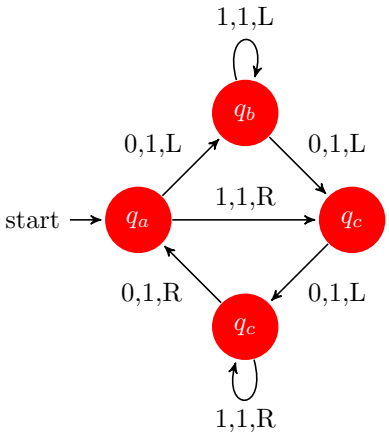

Figure 2: Moore's Machine example

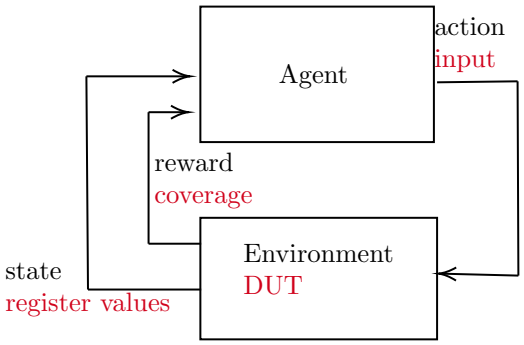

Figure 3: DUT as part of Reinforcement Learning

return the corresponding Moore's Machine (quite similar to an oriented graph). This approach will give us also the opportunity to progressively increase the number of states of the DUT, and then understand how our solution is able to solve the problem.

Solution Statement In the schematic in figure 3 we can see how I'm approaching the problem: my goal will be to design an Agent capable of forcing some input (actions) on the DUT (Environment), based on the register values (state). The reward will be a function of the coverage, which is an indicator of states explored. As previously said, the problem will be solved using reinforcement learning: the episode will be a test, and a step of the episode corresponds to a test's cycle (in the schematic is not shown, but registers have clock input). In particular the actor-critic method as in the DDPG paper will be employed. I'm very willing to master this method because it's very new and seems to be perfect for solving this problem, as it may have a really big number of states. My goal is to try different configurations of DUT, varying the input size between 2 and 100, and the states number between 10 and 1000.

Benchmark Model When performing functional verification we ideally want to reach 100% coverage: this will be our ideal target, which normally is obtained merging the coverage of thousands of random tests. The state of the art of machine learning applied to functional coverage is represented from the work done at IBM. Another scientific reference used for background knowledge is this paper: an important outcome are the coverage results. From these tables we can deduct that 80% of coverage maybe considered as really good.

Evaluation Metrics We have different metrics for evaluating the agent. The more important one is the coverage, which returns us the percentage of stimulated states of the DUT. Another one is the number of cycles needed to reach the 80% coverage; these results have particular relevance when compared with respect to the results obtained using a completely random generator. Finally we can underline how the complexity of agent (actor-critic neural networks weights) scale with the increasing of DUT's complexity.

Project Design The starting point will be the code obtained form the project "Tech a Quadcopter How to Fly", and then reuse the actor-critic approach. The first step of the project is to generate the parametrizable DUT model, which will be used as the environment of the problem. Then a series of events will be generated and the goal will be to generate a test (or a series of tests) capable of learning how to cover the DUT. The project will be hosted on github in this repository.