# CSC2515 Research Project Report

Guanglei(Gabriel) Zhou, Junxi(Stuart) Xu
University of Toronto

December 16, 2020

**Abstract**

Logic synthesis transforms a high-level description of a circuit design into an optimized gate-level representation. The sequence of different optimization techniques plays an important role in logic synthesis. However, finding the optimal combination and sequence of techniques is challenging due to the vast amount of permutations. Recent researches have applied deep learning and reinforcement learning to automate this process. In this report, we implement an advanced *Actor Critic style* Reinforcement learning algorithms to further optimize the automation process and improve the quality of results(QoR).

## 1 Introduction

In computer engineering, logic synthesis is a subject about how to represent logic circuits, how to transform and optimize them[1]. In the logic synthesis framework, there are a variety of optimization algorithms such as balancing, rewrite, refactoring, etc. All these algorithms combine together in sequence and form a large permutation space which makes the optimization process time-consuming and laborious for human experts. Therefore, machine learning techniques are investigated in this area to learn heuristics from the previous searches through the algorithm permutations to satisfy the need for designing optimal flow in complex, heterogeneous, and non-deterministic systems[2].

## 2 Related work

In recent years, some researchers have begun investigating the use of RL to determine good orders for applying optimization passes in high-level synthesis [3] and deciding chip placement[4]. These works show attempts to combine reinforcement learning techniques to improve other CAD algorithms. For logic synthesis, DRiLLs [5] framework is the first work that combines the CAD in Logic synthesis with RL agent. It uses the RL agent to decide the best order of applying the optimization commands. However, it is only focusing on the 7nm ASICs circuit. In the typical logic synthesis optimization flow, we should consider not only the ASICs mapping but the FPGA mapping as well. So there is a small space left for us to explore RL agent on FPGA mapping and observe its performance.

## 3 Environment

ABC[6], a System for Sequential Synthesis and Verification, provides the basic framework of logic synthesis CAD which allows us to read verilog file and perform optimization using developed commands and generate the BLIF file for output.

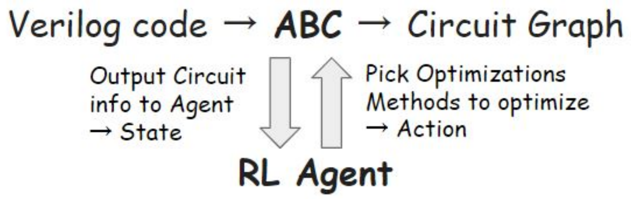
Verilog code → ABC → Circuit Graph
Output Circuit info to Agent → State
Pick Optimizations Methods to optimize → Action
RL Agent

Figure 1: Flow of how the RL agent interact with Logic Synthesis CAD

We use circuits in EPFL Combinational Benchmark Suite[7] as verilog Code input, which are the standard benchmarks for the logic synthesis community. Due to the time limitations of this project, we only go through a limited number of circuits (adder. v, max. v). RL agent's state is customized for the FPGA circuit. which is the number(#) of nodes, # of edges, # of LUT levels and # of LUT count. Action space is using the 7 optimization commands in the ABC framework, resub, resub -z, rewrite, rewrite -z, refactor, refactor -z, balance.

## 4 Models

We investigated several actor-critic style RL algorithms. we first implemented *proximal policy optimization* (PPO)[8]. It uses both "actor" and "critic" to combine policy-based and value-based methods and allow more frequent updates[9]. The advantage (also called TD error) function Eqn.1 evaluates how much "advantage" is obtained after taking a specific action by comparing the next state to the current state. Critic is updated after certain steps to maximize this advantage function using gradient descent. This advantage function is applied while updating the actor. Eqn.2 shows the loss function of the actor. Since the policy update range should be limited, the actor loss function includes the ratio between the new and old policies. Meanwhile, a KL divergence term is added to compute the difference between the new and old policies. This KL divergence acts as a penalty term that penalizes the difference between policies. It is to some extent similar to regularization in linear regression. Eqn.3 shows a modified version of the actor loss function. The objective of this function is the lower bound on the unclipped objective which prevents fast updates.

This method is easier to implement and performs better than the KL method. For this model, we ran the baseline test but failed to integrate it with the ABC environment. We discovered it takes much time to build the environment and the short duration of this course project may not allow us to implement that.

$$\hat{A}_t = \sum_{t'>t} \gamma^{t'-t} r_{t'} - V_\phi(s_t) \tag{1}$$

$$L^{KLPEN}(\theta) = \hat{\mathbb{E}}_t \left[ \frac{\pi_\theta(a_t \mid s_t)}{\pi_{old}(a_t \mid s_t)} \hat{A}_t - \beta KL\left[\pi_{old}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)\right] \right] \tag{2}$$

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[ min\left( r_t(\theta)\hat{A}_t, clip\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t \right) \right] \tag{3}$$

Therefore, we choose to improve the DRiLLs framework for FPGA mapping and use its available environment with ABC. For this model, we use another RL algorithm called advantage actor-critic (A2C)[9]. It is a degraded version of PPO. It also uses both the "actor" and "critic" but does not have the KL divergence to control the update steps. However, despite the model limitation, the A2C method still outperforms other approaches in logic synthesis CAD environment as shown in result section.

## 5 Results

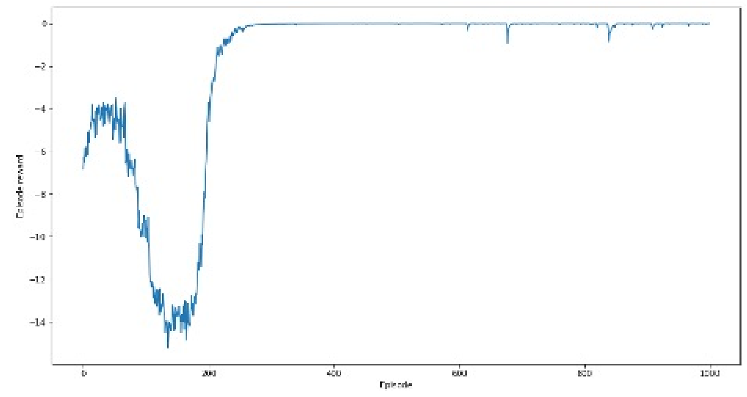

Figure 2: Reward vs Episodes in gym environment "MountainCar-v0"

Figure 2 shows the change of reward with episodes in the gym baseline environment MountainCar-v0 (continuous action space). Although the reward plunges in about 180 episodes, it rapidly increases to the peak after 200 episodes and remains steady. Other baseline environments such as Pendulum-v0 are also applicable to PPO and perform well. For the logic synthesis environment, Table 1 shows the performance of both RL agent and average case optimizations. It

Table 1: Comparison Result between RL agent with average case optimization (resyn2 command)

| | RL Agent | | Average Opt: | | Original: | |
|---|---|---|---|---|---|---|
| Name | LUT Count | Level | LUT Count | Level | LUT Count | Level |
| Max.v | 754 | 42 | 777 | 41 | 842 | 56 |
| Adder.v | 243 | 51 | 257 | 51 | 257 | 51 |

Table 2: Result of Greedy Exploration

| | Best Area: | | Best Level: | |
|---|---|---|---|---|
| Name | LUT Count | Level | LUT Count | Level |
| Max.v | 768 | 41 | 816 | 35 |
| Adder.v | 243 | 51 | 366 | 34 |

can be observed that the RL agent outperforms the average case optimization in both two benchmark circuits. Table 2 lists the comparison results using the Greedy algorithm where at any state, it tries all optimization commands and picks the current best based on the circuit performance. This result is obtained by running Greedy algorithm for 50 iterations. Our RL agent performs better than the Greedy algorithm in terms of the Best Area.

## 6 Discussion

Conclusively, actor-critic Style RL algorithms maintain a good performance in a continuous action space environment. Additionally, it is observed that this model also performs well on discrete action space in our logic synthesis environment. We successfully demonstrate that RL techniques can be integrated with current CAD algorithms and achieve a more adaptive optimization process.
The limitation of this project is also listed below. Firstly, the training of the RL agent is pretty long. Currently, the training data is generated by only one agent. It takes around 40min to train the agent for the Max circuit and 35min for the adder circuit which is much longer than any logic synthesis optimization algorithms. An alternative approach is to allow multiple agents to run the ABC environment to generate the data points which facilitates a better convergence. This is left for future work. Secondly, our experiments just involve two circuit file so that we cannot simply conclude that our RL agent always outperforms other approaches (Greedy, resyn2) for doing logic synthesis optimization. To address this issue, we need to run our model for more circuits which will be left for our future work. Another future work also includes building up the environment with PPO and see if the performance can improve when the update step is unconstrained.

## Attributions

Attributions: All group members contributed equally.
Division of project:
Guanglei(Gabriel) Zhou: Created logic synthesis environment, tested the model on the environment.
Junxi(Stuart) Xu: Constructed the model, tested the model on the gym baseline environment

## References

[1] Jie-Hong (Roland) Jiang and Srinivas Devadas. Logic synthesis in a nutshell. *Electronic Design Automation*, page 299–404, 2009.

[2] H. Leather and C. Cummins. Machine learning in compilers: Past, present and future, Sep 2020.

[3] Ameer Haj Ali, Qijing Huang, William Moses, John Xiang, Ion Stoica, Krste Asanovic, and John Wawrzynek. Autophase: Compiler phase-ordering for high level synthesis with deep reinforcement learning. *CoRR*, abs/1901.04615, 2019.

[4] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Sungmin Bae, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Anand Babu, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter, and Jeff Dean. Chip placement with deep reinforcement learning, 2020.

[5] A. Hosny, S. Hashemi, M. Shalan, and S. Reda. Drills: Deep reinforcement learning for logic synthesis, 2020.

[6] A. Mishchenko et al. Abc: A system for sequential synthesis and verification, 2007.

[7] L. Amarù, Pierre-Emmanuel Gaillardon, and G. D. Micheli. The epfl combinational benchmark suite. 2015.

[8] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, Aug 2017.

[9] Alvaro Durán Tovar. Advantage actor critic (a2c) implementation - deep learning made easy - medium, Dec 2019.