Reinforcement Learning for FPGA Placement

Shang Wang[1], **Deepak Ranganatha Sastry Mamillapalli**[1], **Qianxi Li**[1], **Tianpei Yang**[1], **and Matthew E. Taylor**[1,2]
[1]University of Alberta, Canada
[2]Alberta Machine Intelligence Institute (Amii), Canada
{shang8,mamillap,qianxi,tianpei.yang,matthew.e.taylor}@ualberta.ca

Abstract

This paper introduces the problem of learning to place blocks in Field-Programmable Gate Arrays (FPGAs) and a preliminary learning-based method. In contrast to previous FPGA placement algorithms, we depart from simulated annealing techniques and instead employ deep reinforcement learning (deep RL) for the placement task with the objective of minimizing wirelength. To facilitate the agent's decision-making, we design unique state representations including the chip-board observations and interconnections between different blocks. Additionally, we ground representation learning in the supervised task of predicting placement quality to enhance the RL policy's generalization capabilities. To the best of our knowledge, we are the first to introduce a deep RL agent for FPGA placement, with preliminary results to suggest the feasibility of our approach. We hope that this paper will attract more attention to using RL in FPGAs by electronic design automation engineers.

1 Introduction

The rapid growth in the scale of integrated circuits has heightened interest in electronic design automation. Chip placement, a crucial yet time-consuming step in design, involves mapping netlist components onto the chipboard. Recently, AI has gained traction for accelerating chip design, as seen in Mirhoseini et al.'s RL-based ASIC macro placement [7] and DeepPlace's reinforcement learning success [2]. However, these advances have mainly focused on ASICs, while FPGA design deserves greater attention, as FPGA placement represents a more constrained problem than ASIC placement. Simulated annealing has traditionally been a backbone algorithm in FPGA placement methods due to its natural handling of placement constraints and routing delays. Notably, RLplace [3], the current state-of-the-art approach, enhances the efficiency of simulated annealing by allowing an RL agent to choose from multiple types of directed move, such as random move or directed perturbation. However, RLplace still relies on simulated annealing, which can be slow to converge, particularly for complex FPGA designs. Its probabilistic search process is influenced by factors such as the cooling schedule, and the bandit formulation assumes that the rewards are based solely on intrinsic properties, lacking contextual information.

Solving large-scale sophisticated placement problems is likely to be NP-hard (i.e., are unlikely to be solvable in polynomial time) [6]. We believe that RL offers advantages in tackling FPGA placement problems by providing *approximate solutions*, *adaptability* to changing environments, and *generalization* to similar instances, ideally outperforming traditional placement methods.

2 Methodology

The FPGA placement problem seeks to arrange logic elements on a chip's canvas to optimize performance while meeting various constraints, including the constraint that various block types

Machine Learning for Systems Workshop at 37th NeurIPS Conference, 2023, New Orleans, LA, USA.

can only be placed within pre-fabricated locations (see Figure 1). Traditional algorithms, including RLplace, define the decision space on a chipboard that includes the complete placement of all blocks. But, as in prior work [2, 7], we have chosen to formulate the problem as a sequential Markov decision process (MDP). Our MDP consists of four key elements: **1) States** represent the possible situations the agent can encounter. Our state consists of the netlist graph, the current block to be placed, and the placement status of the current board. **2) Actions** are available decisions an agent can execute. We define actions as locations where a given block can be placed without violating hard constraints (e.g., capacity or block type limits). **3) Rewards** are numerical values that the agent receives as feedback after taking actions. Intermediate steps receive a reward of 0 until the full placement is completed, at which point the final reward is based on the true wirelength generated by VTR after routing. We use Verilog-to-Routing (VTR) [9] as our simulator. VTR is an open-source CAD tool that provides a complete suite of tools for FPGA design, which covers various stages of FPGA design, including synthesis, mapping, placement, routing, and timing analysis. **4) State transitions** define how the state changes after an action (i.e., the new board layout after placing a block in a location).

We define the state to be an image composed of four channels based on the current state of the board and information from the netlist graph about the block to be placed. The four channels are matrices of the same size as the chipboard: 1) the remaining capacity of each grid cell, 2) the number of times the placed block serves as a source/input in all nets, 3) the number of times the placed block serves as a sink/output in all nets, and 4) the total Half-Perimeter Wirelength (HPWL) contributed by each block placed on the grid to the relevant nets. HPWL [14] is defined as the half-perimeter of the bounding boxes for all nodes in the netlist, where x_y and y_b in Equation (1) are the coordinates of the endpoints of net i. In addition to the board observations, the netlist graph represents a directed graph that encodes the connections between various blocks and the nets that link them. To extract valuable information from this netlist graph, we generate a vector representation for each node by concatenating node-specific features, which encompass the node's type, index, and x and y coordinates.

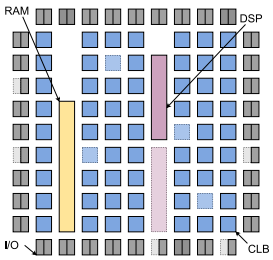


Figure 1: The FPGA board is 11×11 units in size and incorporates DSP, CLB, IO, and RAM blocks, with IO locations having a capacity of 2.

$$\mathrm{HPWL}(i) = \left(\max_{b \in i}\{x_b\} - \min_{b \in i}\{x_b\} + 1\right) + \left(\max_{b \in i}\{y_b\} - \min_{b \in i}\{y_b\} + 1\right) \tag{1}$$

We employ Proximal Policy Optimization (PPO) [10] to train a neural network as a policy model, guiding the RL agent through episodes to maximize cumulative rewards (see Figure 2). The board layout information is processed by through ResNet [5], while the netlist graph is handled by the Graph Attention Network (GAT) [13] to embed node representations. The board observation embedding and the current block embedding vectors are then concatenated to form the state embedding. PPO updates the policy by optimizing the clipped objective function (see Equation (2)), where r_t is the probability ratio of new policy to old policy and A_t is the estimated advantage at time step t.

$$L^{CLIP}(\theta) = \hat{E}_t\left[\min\left(r_t(\theta) \cdot \hat{A}_t, \mathrm{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right] \tag{2}$$

To address limited exploration in a sequential placement setting, we employ the RND framework [1], shown to significantly improve performance in ASIC placement [2]. Instead of traditional sparse rewards, we introduce intrinsic rewards R_T as $R_T = \|\hat{f}(\mathrm{s}_t;\theta) - f(\mathrm{s}_t)\|^2$. These rewards are based on the prediction error between two neural networks: a fixed, randomly initialized target network $f(\mathrm{s}_t)$ and a predictor network $\hat{f}(\mathrm{s}_t;\theta)$ trained on state information s_t. During training, we minimize the mean squared error (MSE) loss by adjusting the predictor's parameters θ. As the predictor network becomes better at approximating the target network, it reduces prediction errors, leading to intrinsic rewards that drive the agent to explore new facets of the placement environment. For extrinsic rewards, VTR processes the complete placement generated by our RL agent for routing and provides the real wirelength of the routed board as the reward score, which we normalize to $[0, 1]$.

To improve learning, we used a pretraining strategy to help the agent learn useful weights in the initial components of the network, similar to how others [3] showed pretraining in ASIC can help an agent converge to higher performance more quickly. We first created a set of 10,000 boards with all the

blocks placed. Each board was labeled with the true wirelengths. We then trained the network to predict wirelengths, with the intuition that a representation capable of predicting wirelengths would be useful for an RL agent learning a policy and a value function.

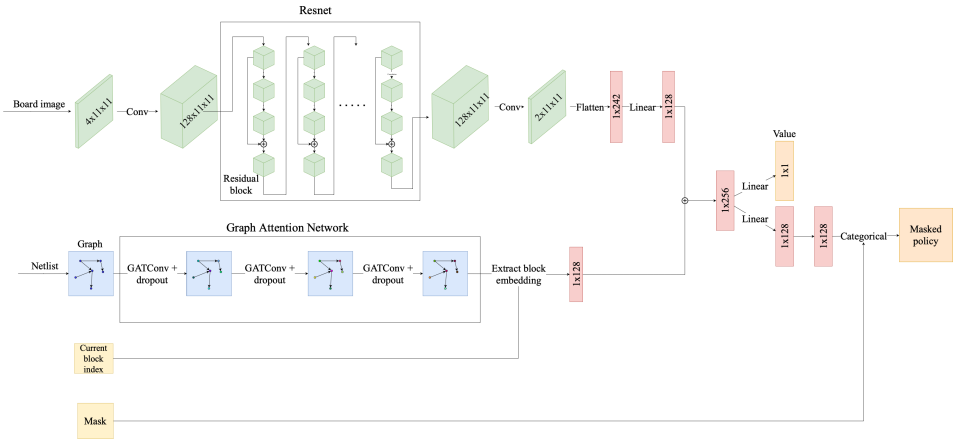


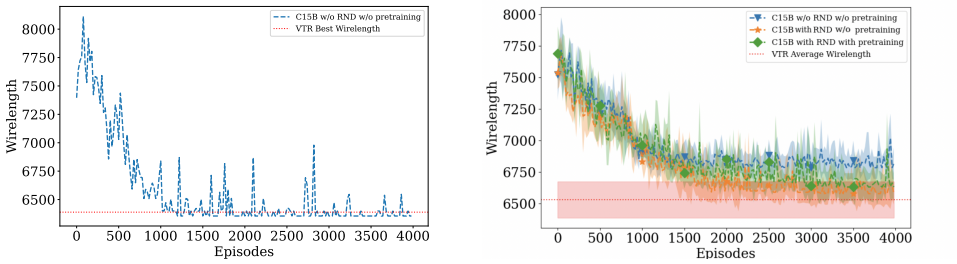
Figure 2: The network takes board observations, the netlist graph, and the current block index as input. It outputs a probability distribution over available placement locations (the policy) and an estimate of the expected reward for the current placement (the value function).

3 Experiments

This section evaluates our base RL agent and the two implemented extensions. We chose wirelength as our primary metric because it aligns directly with our reward mechanism, simplifying our assessment without the need to integrate multiple metrics. We employ the `tseng.net` netlist and `EArch.xml` architecture files as benchmarks from VTR [11]. This set-up consists of 56 CLB and 174 IO blocks. In our initial experiments, we found that the agent struggled to place all the blocks, yielding results no better than random placement. As a result, we decided to simplify the experiments.

We start by using an optimal VTR-derived placement and place a subset of blocks, beginning with a constraint of placing only 1 configurable logic block (CLB), while keeping other block placements fixed. Subsequently, we incrementally increase the number of CLBs the agent is responsible for placing in each experiment, with a special focus on the results of placing 15 CLBs. We compare our RL agent's placements to VTR's, as it is the current state-of-the-art baseline in placement and routing of FPGAs.

In analyzing our results for placements, we find that for the 15-block placement, our agent, averaging over 5 seeds, delivers wirelengths comparable to 5 seeds of VTR (see Figure 3b). Moreover, in specific instances, our agent even outperforms the placements of VTR, highlighting its potential (see Figure 3a). Figure 3b shows that including RND is critical, which improves exploration and results



(a) One example of outperforming VTR

(b) Average performance over 5 seeds

Figure 3: 15 blocks placement

in better placements compared to its counterpart, underscoring RND's key role in avoiding local optima. The difference in results between the runs including RND and excluding RND is statistically significant (a student's t-test reports that $p < 0.03$). Although pretraining does not seem to help much, further experimentation is required to assess if it would help improve sample complexity.

Each run of 4000 episodes takes roughly 4 hours to run on an NVIDIA GeForce RTX 3070 Ti GPU. Each episode takes roughly 4 seconds, which includes the time for: 1) the RL agent to finish a placement, 2) VTR to route the given circuit, and 3) updating the RL agent's policy.

4 Discussion and Future Work

An effective state representation captures the underlying structure of the environment and enables generalization by identifying similarities between different states. While ASIC has large blocks that can take on multiple shapes, FPGA placement has fixed (pre-fabricated) locations that can hold blocks of various types. We used chipboard-based observations and supplemented them with a directed graph to describe the netlist. Future work will consider how to introduce more features related to FPGA placement problems. For example, features such as critical path delay and the internal encapsulation logic of each block could be valuable additions. Compared to using stacked images as observations in Atari games [8], and MuJoCo's state-based observations [12], when only one block is placed at each step, changes between states are quite small. The high similarity in states can lead to difficulties for the policy to distinguish between different states, motivating future research on richer feature spaces.

Rewards in this MDP are particularly sparse, making it difficult to learn a policy, especially when the board size and block size increase [4]. In addition to using RND, future work could change the MDP so that actions swapped placed blocks, allowing the agent to receive the wirelength reward after every action. This change may also help to better differentiate states after each action. Nevertheless, an enormous discrete action space introduced by this MDP presents a significant challenge.[1] A good placement should not only optimize wirelength but also account for critical time delays, congestion, and area utilization. Different applications and objectives can define a "good" placement uniquely, often requiring a trade-off between various metrics. One could approach this by linearly combining different metrics or formulating the problem as a multi-objective RL task to optimize multiple conflicting objectives simultaneously.

Recognizing the balance between placement quality and processing time, our future work will prioritize optimizing processing time. Furthermore, we plan to enhance the applicability of the method by experimenting with diverse netlists, aiming for a more comprehensive understanding of its effectiveness.

5 Conclusion

This paper formulated the FPGA placement problem as an MDP and made a dedicated effort to leverage RL to learn a placement policy. To our knowledge, we are the first to introduce a deep RL model for FPGA placement problems. Much of our effort focused on constructing an appropriate state, as the state used for prior ASIC placement methods was not applicable. We employed two extensions, intrinsic rewards and pretraining, to enhance the performance of the RL agent. While our approach may not yet outperform VTR, we have achieved promising preliminary results. Our work represents a significant step forward in leveraging RL for FPGA placement and we hope others will contribute to further advancements in this exciting area.

Acknowledgments and Disclosure of Funding

This work has taken place in the Intelligent Robot Learning (IRL) Lab at the University of Alberta, which is supported by research grants from the Alberta Machine Intelligence Institute (Amii); a Canada CIFAR AI Chair, Amii; Compute Canada; Huawei; Mitacs; and NSERC.

[1] If a board is 11×11 units in size, the agent would need to select the first block from the 11×11 grid, and then select the second block from $11 \times 11 - 1$ alternatives, for a total of 14,520 possible actions (without considering placement constraints).

References

[1] Yuri Burda, Harrison Edwards, Amos J. Storkey, and Oleg Klimov. Exploration by random network distillation. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[2] Ruoyu Cheng and Junchi Yan. On joint learning for solving placement and routing in chip design. *Advances in Neural Information Processing Systems*, 34:16508–16519, 2021.

[3] Mohamed A Elgammal, Kevin E Murray, and Vaughn Betz. Rlplace: Using reinforcement learning and smart perturbations to optimize fpga placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(8):2532–2545, 2021.

[4] Jianye Hao, Tianpei Yang, Hongyao Tang, Chenjia Bai, Jinyi Liu, Zhaopeng Meng, Peng Liu, and Zhen Wang. Exploration in deep reinforcement learning: From single-agent to multiagent domain. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–21, 2023.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[6] Zhuolun He, Lu Zhang, Peiyu Liao, Yuzhe Ma, and Bei Yu. Reinforcement learning driven physical synthesis. In *2020 IEEE 15th International Conference on Solid-State & Integrated Circuit Technology (ICSICT)*, pages 1–4. IEEE, 2020.

[7] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Sungmin Bae, et al. Chip placement with deep reinforcement learning. *Nature 594, 207–212 (2021).*, 2021.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[9] Kevin E Murray, Oleg Petelin, Sheng Zhong, Jia Min Wang, Mohamed Eldafrawy, Jean-Philippe Legault, Eugene Sha, Aaron G Graham, Jean Wu, Matthew JP Walker, et al. Vtr 8: High-performance cad and customizable fpga architecture modelling. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 13(2):1–55, 2020.

[10] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[11] Verilog to Routing (VTR) Project Team. Benchmarks. `https://docs.verilogtorouting.org/en/stable/vtr/benchmarks/?highlight=Benchmarks`, 2022.

[12] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[13] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. *CoRR*, abs/1710.10903, 2017.

[14] Ming Xu, Gary Gréwal, Shawki Areibi, Charlie Obimbo, and D Banerji. Near-linear wirelength estimation for fpga placement. *Canadian Journal of Electrical and Computer Engineering*, 34(3):125–132, 2009.