TRPlaceFPGA-MP: A Two-stage Reinforcement Learning Framework for Fast FPGA Macro Placer

Qin Luo[1], Xinshi Zang[1], Evangeline F.Y. Young[1], Martin D.F. Wong[2]
[1]The Chinese University of Hong Kong, [2]Hong Kong Baptist University
{qluo22, xszang, fyyoung}@cse.cuhk.edu.hk, mdfwong@hkbu.edu.hk

***Abstract*—Reinforcement learning (RL)-based macro placement has garnered significant interest in both the fields of artificial intelligence and electronic design automation (EDA), due to its excellent potential for achieving better performance, power and area optimization compared to analytical methods. However, existing techniques are restricted in the ASIC and ignore the other hardware architectures like FPGA. Neglecting the intrinsic characters of FPGA structures, conventional RL-based methods for ASICs may result in a large exploration space and low sample efficiency. In this work, we propose TRPlaceFPGA-MP, a two-stage RL-based macro placement framework for Ultrascale FPGAs. Leveraging the columnar architecture, we first train a tiny RL model to determine the candidate columns for each macro in the first stage. With the pruned searching space, a more sophisticated RL model is then trained in the second stage to determine the ultimate positions of the macros. Experimental results on the MLCAD2023 contest benchmark demonstrate that TRPlaceFPGA-MP still maintains superior placement performance compared with Vivado and DreamplaceFPGA-MP. Furthermore, it improves the convergence rate by 2.28x and accelerates the exploration process by 1.61x compared to the one-stage RL approach.**

I. Introduction

Modern circuit designs heavily rely on macros, which are pre-defined and pre-verified functional blocks aimed at reducing design complexity [1]–[3]. In the design flow of field-programmable gate arrays (FPGAs), macro placement refers to determining the sites of the memory blocks and digital signal processors [4], [5]. The placement of macros plays a crucial role in overall routability and timing. In recent years, reinforcement learning (RL)-based macro placement has emerged as a popular research topic. Notably, research conducted by the Google Brain Team [6] has revealed that RL-based macro placement methods can achieve comparable or even superior performance when compared to human-designed layouts. RL-based methods offer distinct advantages over analytical approaches [7]–[9] in handling non-differentiable objectives such as Half-Perimeter Wirelength (HPWL), cell density, and timing violations without the need for approximations. Recent research works have further advanced RL-based macro placement techniques by integrating the DreamPlace [8] as a standard cells placer [10]–[12], employing dense rewards [13], [14], exploring the offline RL scheme [15], or considering the cell-shifting approach [16].

Existing research on RL-based macro placement has primarily focused on application-specific integrated circuits (ASICs), while limited attention is given to another equally important architecture: FPGAs. FPGA macro placement introduces additional challenges due to their heterogeneity and limited resources for the place and route process. While there have been works such as [17], [18] that explore the use of RL methods as auxiliary techniques to search for the best directed moves in simulated annealing-based FPGA placers, these approaches heavily rely on heuristics and lack flexibility with the predefined directed moves.

In addition, existing RL-based macro placers suffer from the limitation of requiring a large number of iterations and significant time to achieve convergence in exploring better placement solutions. Even for the Chipformer [15] that is based on the offline RL methods, additional refinement based on the time-consuming online RL is still required to improve the placement results. Research conducted by Andrew B. Kahng's group [19] demonstrates that online RL-based macro placement can take more than 10 hours to obtain a policy network with satisfactory performance. Moreover, certain RL-based macro placers place a subset of macros based on the guidance from the initial placement generated by the analytical placer or the commercial tool [13], [20], [21], with the number of a few hundred. However, in FPGA macro placement, the number of macros to be placed can reach more than thousands, posing additional challenges in achieving fast convergence and reducing exploration time. The slow convergence rate and the long exploration time can be attributed to the large search space in the exploration. Therefore, proper pruning of the search space is highly significant in RL-based FPGA macro placement.

In our paper, we introduce TRPlaceFPGA-MP, a two-stage RL-based macro placer specifically designed for FPGA architectures. We emphasize two key features that address the unique characteristics of FPGAs:

- **New architecture:** our RL-based macro placer is specifically designed to leverage the columnar architecture and heterogeneity of FPGA boards.
- **Fast convergence:** a two-stage decomposition approach is proposed to address the challenge of slow convergence and long exploration time. In the first stage, candidate columns are determined for each macro. In the second stage, the placement location is determined within the search space restricted by these candidate columns. By exploring a significantly smaller search space at these two stages, our RL agent achieves faster convergence and reduces exploration time.

To demonstrate the effectiveness of our method, we compare the macro placement performance like wirelength, with the commercial tool Vivado and the leading academic FPGA macro placer DreamplaceFPGA-MP, using the MLCAD2023 contest benchmark. We also conduct experiments comparing the convergence speed and exploration time with MaskPlace^f, a one-stage approach that adapts MaskPlace [13] specifically for FPGAs.

II. Preliminary

A. FPGA Architecture

Our macro placer targets at the Xilinx Ultrascale series FPGAs [22]. These FPGAs are characterized by a column-wise structure, as depicted in Figure 1. The architecture of these FPGAs incorporates five common site types that allow for flexible configuration: configurable logic block (CLB), digital signal processor (DSP), block RAM (BRAM), input/output buffer (IO), and switchbox. Each CLB can accommodate multiple look-up tables (LUTs) and flip flops (FFs). Within the same column, all sites have the same types. The switchboxes serve the purpose of interconnecting different sites using prefabricated wires.

During the macro placement, the BRAMs and DSPs are considered as the macros. These macros exhibit heterogeneous sizes, ranging from single macros to cascaded macros that comprise multiple BRAMs and DSPs [23], [24].

B. Problem Formulation

When applying online RL technique to FPGA macro placement, several fundamental elements can be defined:

- State s_t - the state encodes both the partial FPGA macro placement solution, the features of the macro to be placed and the global information of the netlist. It can be represented as a gridmap that reflects the columns placing macros and IOs on the FPGA board. For example, the Xilinx Ultrascale+ xcvu30 FPGA board consists of 33 columns placing macros as well as IOs, with a height of 300. Each BRAM and DSP site occupies 5 grids and 2.5 grids respectively. Then the size of the gridmap is 33x300. For cascaded macros with n units, the occupation of grids in the grid map is the accumulation of the size of each unit within the cascaded macro.
- Action a_t - the action can be represented as a probability map, corresponding to the grid map representing the available sites on the FPGA board.
- State transition $T(s_{t+1}|s_t, a_t)$ - it is defined by the probability distribution over the next state s_{t+1} given the current state s_t and action a_t.
- Rewards R_t - it can be defined as the negative increment of the Half-Perimeter Wirelength (HPWL) for the macros and fixed IOs, which we'll refer to as MacroHPWL:

$$MacroHPWL = \sum_{n \in N} [(\max_{i \in S(n)} x_i - \min_{i \in S(n)} x_i) + (\max_{i \in S(n)} y_i - \min_{i \in S(n)} y_i)] \quad (1)$$

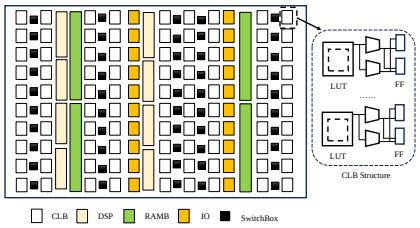


Fig. 1: The layout of the Xilinx Ultrascale board.

where N denotes the set of all the nets, and $S(n)$ represents the set of the IO ports and the macros that are connected to the net n and have already been placed. The reward R_t can be densely calculated to provide immediate feedback after each macro is placed.

In our RL-based FPGA macro placement problem, the objective is to train a policy network that sequentially places the macros on the gridmap while considering the following factors:

a) Producing the valid macro placement solution. The valid placement solution means ensuring that each macro should be placed on the corresponding site.

b) Maximizing the reward (which also minimizes the $MacroHPWL$)

III. Methodology

The overall framework of TRPlaceFPGA-MP is depicted in Figure 2. In this framework, macros are placed sequentially onto valid positions on the FPGA board, and it consists of two distinct stages. The first stage involves predicting candidate columns, while the second stage determines the precise position within the chosen columns.

To achieve this, both stages utilize the actor-critic framework, incorporating policy and value networks. Initially, three masks are computed to represent the column information or the space information of the partial placement. Meanwhile, the embeddings are extracted from the netlist through a pretrained Variational Graph AutoEncoder (VGAE) [25] to serve as the global structural information. Furthermore, the features of the macros to be placed are concatenated with these masks and the netlist embeddings, forming the input for the policy network. The policy network analyzes this concatenated input and generates a probability map that indicates the likelihood of placing the macro in each column or at specific grid locations on the FPGA board. The candidate columns or placement locations are sampled from this action probability map. Additionally, the concatenated features are also utilized as input for the value network. The value network plays a role in enhancing the training process of the RL algorithm by providing supplementary guidance.

A. Motivation

The primary challenge faced by online RL-based macro placers is the time overhead associated with the exploration, particularly when dealing with a larger number of macros and expansive search space. Suppose the number of the macros is

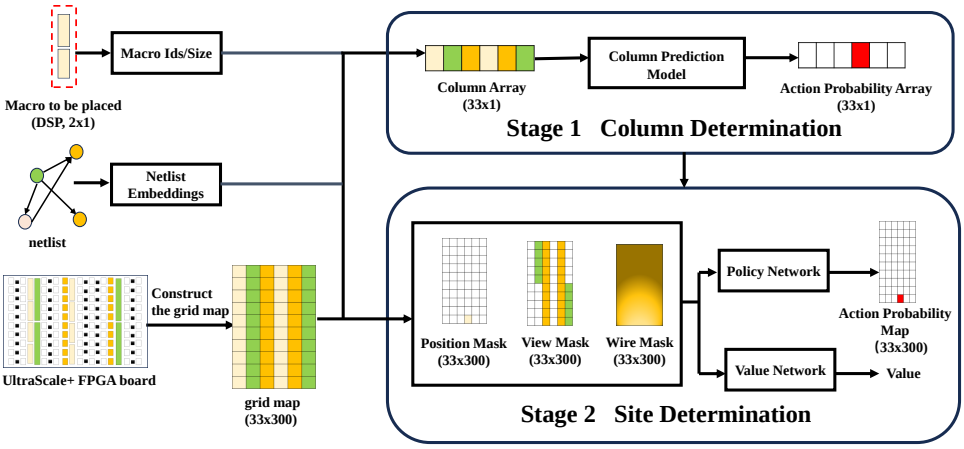


Fig. 2: The overall framework for TRPlaceFPGA-MP.

N, the number of the grids occupied by the macro i is a_i and the total number of sites is a_T. The size of the search space, denoted as c_s can be calculated as follows:

$$c_s = \prod_{i=1}^{N} \left(a_T - \sum_{j=1}^{i} a_i \right) \approx (a_T)^N \tag{2}$$

Previous research in RL-based ASIC macro placement typically involves an initial placement using an analytical placer, followed by the selection of a subset of macros (often with $N < 1000$) for further refinement using online RL. However, in the case of FPGA designs, the number of macros is significantly higher. For example, in the MLCAD2023 contest benchmark [4], there are approximately two thousand macros. Furthermore, the macros to be placed in later stages of FPGA designs exhibit more flexibility and options due to their smaller size compared to the macros in ASIC. The increased number of macros and the larger search space in FPGA designs result in the need for more extensive exploration and slower convergence rates for RL-based macro placement. The slower convergence rate contributes to increased time expenses on the online training.

To enhance the practicality and reduce online exploration time in RL-based macro placement, the TRPlaceFPGA-MP introduces a two-stage decomposition approach and leverages the column structure of the FPGA board. In the first stage to determine the candidate columns, the size of the search space c_{s1} is calculated as:

$$c_{s1} = (n\_col)^N \tag{3}$$

where n_col is the number of columns in the gridmap. In the second stage to determine the specific sites within the candidate columns, the size of the search space c_{s2} is:

$$c_{s2} = (k * \frac{a_T}{n\_col})^N \tag{4}$$

where k is the number of the candidate columns selected for each macro. The first policy model explores n_col columns to identify the candidates, thereby helping to narrow down the search space and reduce the exploration required in subsequent stages. We divide the search space of the two-stage RL-based macro placer with that of the one-stage macro placer:

$$\frac{c_{s_1} + c_{s_2}}{c_s} \approx (\frac{n\_col}{a_T})^N + (\frac{k}{n\_col})^N \tag{5}$$

Under the circumstances that the size of the gridmap is 33x300 and top 5 columns are selected as candidates, the ratio is approximately calculated as:

$$(\frac{n\_col}{a_T})^N + (\frac{k}{n\_col})^N \approx (\frac{1}{300})^N + (\frac{5}{33})^N < 1 \tag{6}$$

which verifies the reduction of the search space when adopting the two-stage RL-based macro placer.

B. Stage 1: Column Determination

1) Feature Construction: To determine the suitable columns for macro placement, the problem is modeled as a bin packing problem with the objective of minimizing the increase in MacroHPWL while ensuring sufficient available sites to accommodate the macros. Let $\{n^i_{sites}\}_{i=1}^{n\_col}$ represent the number of sites in each column, and $\{n^i_{occ}\}_{i=1}^{n\_col}$ denote the number of occupied sites in each column. The condition for placing the macro with m BRAMs/DSPs in a column is as follows:

$$n^i_{occ} + m \leq \alpha n^i_{sites} \tag{7}$$

where α is the coefficient that controls the macro density in one column. In this case, we set α as 0.9. Meanwhile, the wirelength at the x-direction needs to be minimized:

$$MacroHPWL_x = \sum_{n \in N} (\max_{i \in S(n)} x_i - \min_{i \in S(n)} x_i) \tag{8}$$

In TRPlaceFPGA-MP, we develop and train the column determination model using online RL to identify the optimal columns that minimize the wirelength increase in the x-direction and prevent column overflow. The overall flow is presented in Figure 3, which involves utilizing three masks in one direction as input and generating a probability array

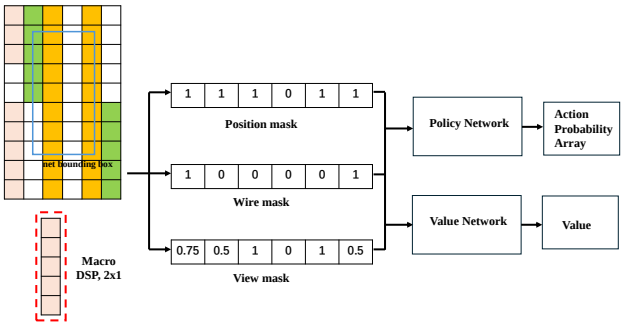


Fig. 3: Column prediction model

as output. The construction of the three masks in the column prediction model is as follows:

View mask. The view mask is constructed by calculating the occupation ratio of the sites within a single column. Each component of the view mask representing this ratio is determined as follows:

$$pos\_mask_i = \frac{n^i_{occ}}{n^i_{sites}} \tag{9}$$

In Figure 3, the 6x10 gridmap has 2 sites available for placing BRAMs and 4 sites available for placing DSPs in each column. Let's consider the occupation of the sites in the first two columns as examples. In the first column, a macro consisting of two cascaded DSPs takes up 2 sites and a single DSP macro takes up 1 site. The first element in the view mask is 0.75. In the second column, a BRAM with a height of 5 grids occupies one site in this column. Therefore, the second element in the view mask is 0.5.

Position Mask. It is a binary array that indicates whether an overflow would occur when placing a macro in a specific column. For columns that match the macro type and satisfy $n^i_{occ} + m \leq n^i_{sites}$, the corresponding elements in the position mask are marked as 0. Otherwise, they are marked as 1. In Figure 3, if a cascaded DSP macro that occupies 5 grids is to be placed next, then the fourth element in position mask in marked as 0 because only it satisfies the conditions.

Wire Mask. Each element in the wire mask represents the increase in wirelength in the x-direction. Referring to Figure 3, the net bounding box is depicted in blue and spans the middle three columns. In this case, if the macro is placed in the first or last column, the increase in wirelength in the x-direction would be 1. However, if the macro is placed in any other column within the net bounding box, the increase in wirelength would be 0.

The netilist structural information is crucial in determining the position for each macro. Therefore, the netlist embeddings are generated with the Variational Generative Autoencoder (VGAE) [25] as a meta data shown in Figure 4. The graph containing the the macros as vertices and the connections between them as edges is created from the netlists. We also define the an $N \times N$ adjacency matrix A and an $N \times 5$ matrix X as the vertice features, where N is the number of the vertices in the graph. The vertice features contain the geometric information of macros like width, height and area, as well as the categorical information (BRAM or DSP macros). The inference model takes the adjacency matrix and vertice features as the inputs and outputs the mean and variance of the embedding distribution for each vertices. The generative model samples each node embedding and reconstructs the connections between the macros. The model is pretrained via reducing the loss between the real and reconstructed adjacency matrices. The netlist embeddings NE are calculated as the average of the sampled embeddings for each vertice:

$$NE = \frac{1}{N}\sum_{i=1}^{N} Z_i \tag{10}$$

where Z is the sampled embeddings for vertices.

The macro information includes the order index and the corresponding size. The order of macro placement is determined by two factors: macro size and degree of connectivity. We first sort all the macros in descending order based on their sizes. This ensures that larger macros are considered first during placement. Next, we prioritize the macros based on their degree of connectivity, which represents the number of connections to IOs and other macros. Macros with a higher degree of connectivity are given higher order priority.

After obtaining the masks, netlist embeddings and macro features, they are input into the policy network in Stage 1. The policy network outputs the probability of placing the macro on each column. From these probabilities, we select the top k columns with the highest probabilities as candidate columns. These candidate columns represent potential locations for macro placement. In Stage 2, the search space is restricted within the candidate columns.

2) Loss Function: The Proximal Policy Optimization (PPO) algorithm [26] is employed to optimize the policy and value networks. In training the policy network, the loss function is formulated as follows:

$$L_{policy}(\theta) = \mathbb{E}[\min(r_t(\theta)A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)] \tag{11}$$

where the ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ and $A_t = G_t - V_t$ denotes the advantage function. Here we adopt the cumulative discounted reward $G_t = \sum_{k=1}^{V-t-1} \gamma^k r_{t+k+1}$. V_t is the estimated value produced by the value network. The value network is updated by optimizing the square error between the cumulative reward G_t and the estimated value V_t, which is defined as:

$$L_{value}(\phi) = \mathbb{E}[(G_t - V_t)^2] \tag{12}$$

C. Stage 2: Site Determination

In the Stage 2 of TRPlaceFPGA-MP, three types of pixel-level feature maps with two dimensions, still namely position mask, wire mask, and view mask, provide important information for the placement process. These three masks are discussed below:

Position Mask. The position mask is represented as a binary matrix. In this matrix, a value of "0" indicates a valid position to place a macro, while a value of "1" signifies an infeasible

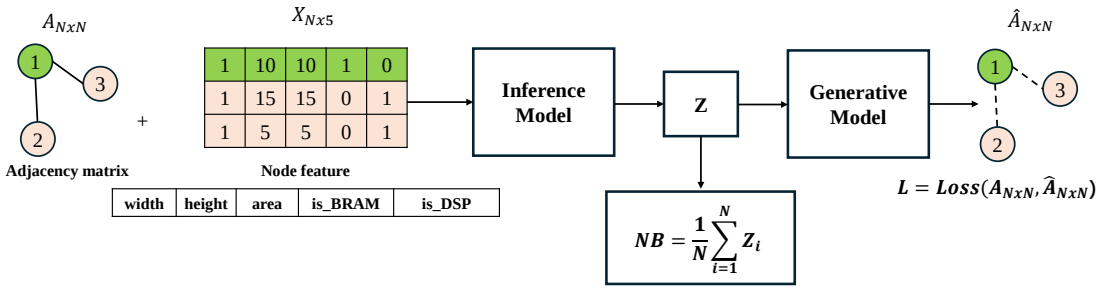


Fig. 4: The generation of the netlist embeddings through VGAE.

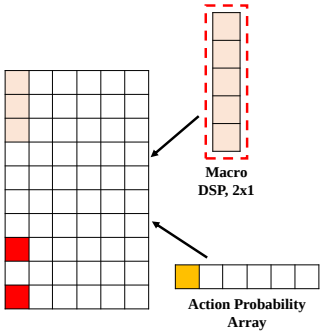


Fig. 5: The example of calculating the placement mask (Pink color: the DSP macros, orange color: the candidate columns from Stage 1, red color: the valid sites to place macros)

position. The position mask takes into account various constraints such as overlap, boundary constraints, heterogeneous nature of the FPGA, and column restrictions from Stage 1. Firstly, only the top k columns that match the type of the macro to be placed in the Stage 1 are considered. Furthermore, specific positions within a column are determined based on the macro type. For example, every 5 grid positions are considered feasible for placing BRAMs, while every 2.5 grid positions are considered feasible for placing DSPs within a column.

Figure 5 illustrates an example of calculating the position mask for positioning a cascaded macro with 2 DSPs. Let's assume that one DSP has already been placed on the gridmap. The action probability array from Stage 1 indicates that only the first column is eligible for macro placement. It can be observed that only 2 sites are valid options for macro placement. These valid sites are all located within the first column. Notably, within the same column, the valid sites are separated by a gap equivalent to the height of the DSP, which amounts to 2.5 grids.

Wire Mask. The wire mask is a matrix consisting of continuous elements ranging between 0 and 1. Its purpose is to record the increase in MacroHPWL when placing the macro in each grid. The MacroHPWL increase can be further decomposed into increments in the x-direction and y-direction. Suppose the bounding box of the net n which involves the macros and IOs is denoted as $(x_{dl}, y_{dl}, x_{ur}, y_{ur})$. Here (x_{dl}, y_{dl}) represents the coordinate of the left-down corner of the bounding box, while (x_{ur}, y_{ur}) represents the coordinate of the right-up corner of the bounding box on the FPGA board. The increase of the MacroHPWL of the net n in the x-direction when placing macros at (x_m, y_m) can be calculated as:

$$\Delta MacroHPWL_x(x_m, y_m, n) = \begin{cases} x_{dl} - x_m, & x_m \leq x_{dl} \\ 0, & x_{dl} < x_m < x_{ur} \\ x_m - x_{ur}, & x_m \geq x_{ur} \end{cases} \tag{13}$$

The calculation for the MacroHPWL increase in the y-direction follows a similar approach. Additionally, during the wire mask calculation, it is crucial to consider the conversion from the coordinates in the gridmap to the coordinates in the original FPGA layout. This conversion ensures that the wire mask accurately reflects the increase in MacroHPWL in the context of the original FPGA layout.

View Mask. The view mask, denoted as $f_v \in \{0,1\}^{33\times300}$, provides a global observation of the FPGA placement. In this mask, a value of "1" indicates that a grid is occupied by a macro, while a value of "0" indicates that the grid is empty. For cascaded macros that contain n BRAMs, they cover a span of $5n$ grids. Similarly for the cascaded macros with m DSPs, they cover a span of $2.5m$ grids.

The policy network in Stage 2 takes the masks along with the macro features and the netlist embeddings as the inputs. It processes this input information and outputs the probability of placing the macro on each grid location. The site to place the macro is sampled from the probability distribution. The PPO algorithm and the same loss function in section III-B are used in the training of the policy and the value networks in Stage 2.

D. Search Space Transition

Column restrictions can indeed improve the efficiency of searching for optimal macro placement sites. However, they also introduce the risk of getting stuck in local optima due to the limited search space. To overcome this challenge, a strategy is employed that initially utilizes the column restriction to narrow down the search space and identify potential optimal solutions. Subsequently, the search space is expanded to cover the entire gridmap, increasing the chances of finding an improved solution.

In this strategy, the learning rate is increased as the search space expands. By increasing the learning rate as the search

TABLE I: The information for sample designs in MLCAD2023 contest benchmark

	Basic Information				Macros/IOs		
	# Nodes	# Nets	# Macros	# Placement Nets	# BRAMs	# DSPs	# IOs
Design_182	566055	615920	2320	37173	1824	576	456
Design_185	575626	633913	2048	31623	1824	576	
Design_190	576050	634364	2048	27875	1824	576	
Design_197	594824	645901	2380	34013	1870	590	
Design_207	622234	674462	2440	34959	1915	605	
Design_220	651076	712876	2228	29502	1961	619	
Design_232	697886	753658	2620	42517	2052	648	
Design_240	707070	771230	2348	29201	2052	648	

space expands, the algorithm can overcome the confinement in the vicinity of local optimal solutions and enables it to approach the global optimum more effectively.

IV. Experiment

A. Setups

In this section, we evaluate TRPlaceFPGA-MP in terms of macro placement performance. We conduct the evaluation using eight representative designs from the MLCAD2023 contest benchmark [4]. These designs feature cascaded macros and do not have any regional constraints. Table I provides basic statistics for the designs. The number of macros to be placed on the FPGA board ranges from 2048 to 2620. The placement nets refer to the nets which simply connect to the macros and the IOs, and the number ranges from 27k to 37k. All the macros are placed on the Xilinx Ultrascale+ xcvu3p FPGA board [22] whose size is 206x300. We extract the gridmap with the size of 33x300 from the FPGA board. The number of BRAM and DSP sites is 2280 and 720, and the utilization is relatively high with the ratios between 0.8 to 0.9. The positions of the IOs are fixed in the designs. All the experiments are performed on a server equipped with an Intel(R) Xeon(R) Gold 6326 CPU and an NVIDIA A800-SXM4-80GB GPU. To construct FPGA macro placement environment, we utilize the Gym library [27] developed by OpenAI.

Little literature has explored the direct usage of RL in FPGA macro placement. In this study, we compare our macro placer with the commercial tool Vivado [28], the analytical macro placer DreamplaceFPGA-MP [29] and a self-modified RL-based FPGA macro placement model called MaskPlace^f. The followings are the details of the baselines:

- **Vivado** - here we use the version ML 2021.1. It adopts the SimPL algorithm for the macro placement.
- **DreamplaceFPGA-MP** - it is a non-linear FPGA macro placer with modeling the cell density as the electrostatics system and also the winner of the MLCAD 2023 FPGA macro placement contest. We adopt its CPU version in our experiments.
- **MaskPlace^f** - it is a one-stage RL-based macro placer that builds upon the MaskPlace [13] by adapting it to the specific characteristics of FPGAs. It shares similarities with Stage-2 in TRPlaceFPGA-MP. In the implementation of TRPlaceFPGA-MP, multiple linear layers are utilized to construct the policy and value networks in Stage 1. In Stage 2, the network structure is similar to that in MaskPlace^f with an encoder based on ResNet-18 [30] and multiple convolution layers as the decoder.

The training process for the policy and value networks is performed over a total of 300 epochs. For the column determination network, the training duration is set to 200 epochs. The learning rate is selected from the set $\{2.5e^{-2}, 2.5e^{-3}\}$. However, for TRPlaceFPGA-MP, which includes the column determination process, the search space is initially limited to candidate columns for the first 50 epochs and then expanded to cover the entire FPGA board. During the search space transition, the learning rate is adjusted to $2.5e^{-2}$, which is subsequently reduced to $2.5e^{-3}$ after 100 epochs. Throughout the online RL process, all transitions, consisting of state-action-reward pairs, are stored in a buffer for model updates.

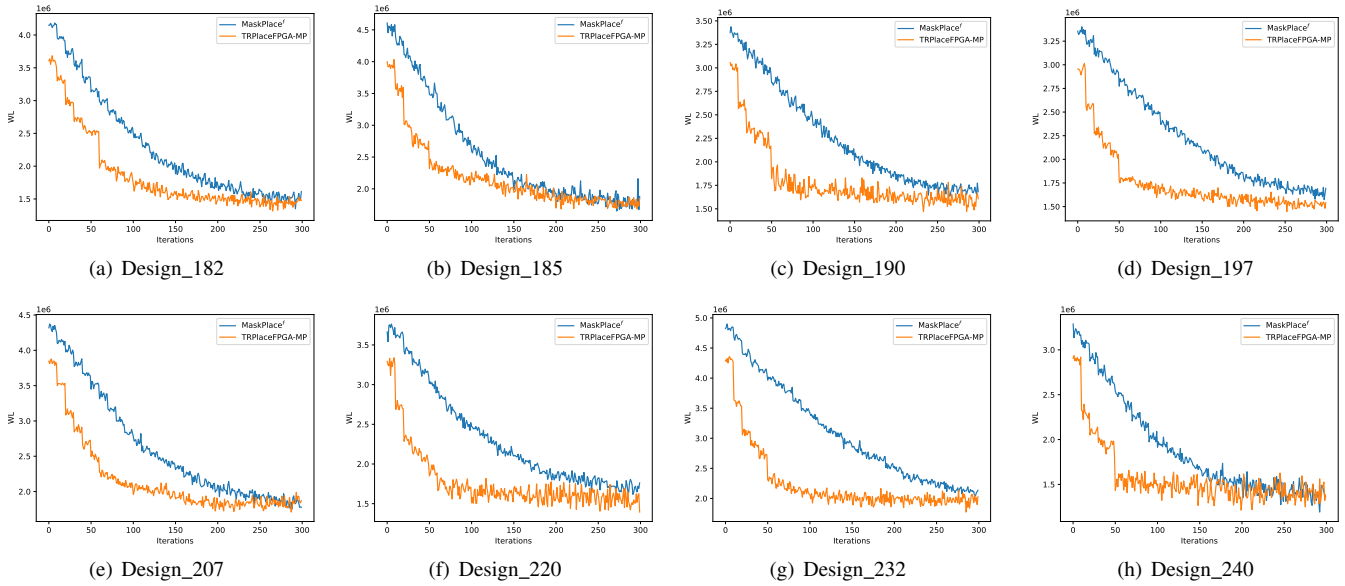
B. Macro placement performance comparison

We use the wirelength (WL) and runtime (RT) as the evaluation metrics for the macro placement solution. The wirelength is calculated as the total HPWL for the nets connecting the macros and the IOs at the epoch reaching convergence. For the RL-based macro placers TRPlaceFPGA-MP and MaskPlace^f, the convergence is determined when the standard deviation of the wirelength, after reaching convergence, falls below 5% of the average wirelength. Smaller wirelength can result in fewer demands on the routing resources and better routability for the designs. The runtime simply considers the macro placement process, whose record stops when it reaches convergence.

Table II demonstrates the comparison results. As the *place_design* command in Vivado involves both the macro and standard cell placement, the runtime for Vivado is not reported. TRPlaceFPGA-MP can reduce the wirelength by 9%, 14% and 1% compared with Vivado, DreamplaceFPGA-MP and MaskPlace^f, maintaining the superiority in the wirelength optimization for the RL-based macro placers. Furthermore, TRPlaceFPGA-MP significantly reduces the runtime for exploration by 61% when compared with the MaskPlace^f. The comparison results verify that TRPlaceFPGA-MP can search for a better macro placement solution with more reduction in the exploration time.

C. More discussions on convergence and runtime

1) Convergence Analysis: The training curves about WL for TRPlaceFPGA-MP and the MaskPlace^f, are demonstrated in Figure 6. It is observed that both TRPlaceFPGA-MP and MaskPlace^f can converge to a similar level of WL within



(a) Design_182 (b) Design_185 (c) Design_190 (d) Design_197

(e) Design_207 (f) Design_220 (g) Design_232 (h) Design_240

Fig. 6: The training curves concerning the wirelength (WL) for TRPlaceFPGA-MP and MaskPlace^f

TABLE II: Comparison of the macro placement performance between different FPGA macro placers (WL: wirelength when reaching convergence, R: ratio of the WL over TRPlaceFPGA-MP, RT: the time for macro placement before convergence)

	Vivado		DreamplaceFPGA-MP			MaskPlace^f			TRPlaceFPGA-MP		
	WL	R	WL	R	RT (min)	WL	R	RT (min)	WL	R	RT (min)
Design_182	1523942	1.08	1660745	1.18	3.33	1427266	1.01	217.07	1412026	1	160.45
Design_185	1618835	1.09	1641812	1.1	3.56	1495516	1	275.4	1489328	1	244.93
Design_190	1791538	1.09	1790675	1.09	3.53	1666130	1.01	192.07	1648314	1	85.53
Design_197	1795939	1.12	1891286	1.18	3.46	1570223	0.98	261.41	1602263	1	125.48
Design_207	1916483	1.09	1958940	1.11	3.59	1782329	1.01	212.17	1763164	1	143.89
Design_220	1673995	1.06	1834076	1.16	3.6	1656815	1.05	250.42	1580075	1	128.22
Design_232	2034491	1.09	2176743	1.16	3.64	1892076	1.01	333.41	1871731	1	128.2
Design_240	1447351	1.08	1503430	1.12	3.81	1361072	1.01	339.49	1341562	1	150.25
Geomean	1715427		1796661			1597739			1580079		
Ratio	1.09		1.14			1.01			1		

TABLE III: Comparison of the exploration time for TRPlaceFPGA-MP and MaskPlace^f (n_c: the number of iterations to achieve convergence, t_{CD}: the time for the column determination, t_{SD}: the time for site determination, RT: the time for the overall macro placement before convergence, the unit for the time is minute).

Designs	TRPlaceFPGA-MP				MaskPlace^f		Designs	TRPlaceFPGA-MP				MaskPlace^f	
	n_c	t_{CD}	t_{SD}	RT	n_c	RT		n_c	t_{CD}	t_{SD}	RT	n_c	RT
Design_182	125	21.22	139.23	160.45	207	217.07	Design_207	100	23.34	120.55	143.89	186	212.17
Design_185	159	22.15	222.78	244.93	230	275.4	Design_220	100	19.58	108.64	128.22	174	250.42
Design_190	53	19.15	66.38	85.53	170	192.07	Design_232	58	24.63	103.57	128.2	204	333.41
Design_197	61	24.32	101.16	125.48	179	261.41	Design_240	78	22.05	128.2	150.25	220	339.49
Geomean								85.69	21.97	117.45	140.27	195.16	225.31
Ratio								1	1	1	1	2.28	1.61

300 epochs, but TRPlaceFPGA-MP achieves convergence with significantly fewer iterations. Table III demonstrates that TRPlaceFPGA-MP can converge within 86 iterations on average, while MaskPlace^f needs to take 196 iterations to converge on average. The speedup in the convergence is 2.28x. Notably, in Design_190 and Design_232, the acceleration in convergence reaches as high as 3.20x and 3.51x, respectively.

The acceleration of convergence in TRPlaceFPGA-MP compared to MaskPlace^f can be attributed to two factors. Firstly, TRPlaceFPGA-MP exhibits a smaller WL at epoch 0, indicating a better initial placement facilitated by two-stage framework. Secondly, TRPlaceFPGA-MP demonstrates a faster rate of WL reduction in the initial 50 epochs. This increased speed can be attributed to the appropriate restriction of the search space.

2) Runtime Decomposition and Analysis: We also analyze the exploration time for the TRPlaceFPGA-MP and the MaskPlace^f before convergence. The comparison of the

exploration time is presented in Table III. The exploration time for TRPlaceFPGA-MP can be decomposed into the time required for both column determination t_{CD} and the time for the site determination t_{SD} . For MaskPlace^f, it solely involves the time for site determination t_{SD} using online RL. The results in Table III demonstrate that TRPlaceFPGA-MP reduces the exploration time from an average of 3.75 hours to 2.3 hours. The acceleration is mainly attributed to the column restriction in stage 2 for TRPlaceFPGA-MP. Without considering the time expense of the column determination t_{CD} , the speedup is 1.91x.

V. Conclusion

In this paper, we propose an FPGA macro placer using online reinforcement learning and a two-stage decomposition to accelerate convergence. Our approach can still reduce the wirelength by 9% and 14% when compared with Vivado and the state-of-art academic FPGA placer DreamplaceFPGA-MP. Meanwhile, a remarkable 2.28x acceleration in convergence rate and a 61% decrease in exploration time is achieved compared to one-stage RL-based FPGA placers. The reduced exploration time brings RL-based FPGA placers closer to more practical application scenarios with maintenance of the superiority in placement performance optimization.

Acknowledgement

The work described in this paper was partially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CUHK 14209624). We sincerely appreciate Professor Tsung Yi Ho for the permission of Nvidia A800 GPU usage.

References

[1] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, and B. Hutchings, "Hmflow: Accelerating fpga compilation with hard macros for rapid prototyping," in *2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2011, pp. 117–124.

[2] M. Nemani and V. Tiwari, "Macro-driven circuit design methodology for high-performance datapaths," in *Proceedings of the 37th Annual Design Automation Conference*, 2000, pp. 661–666.

[3] W.-K. Mak and E. F. Young, "Temporal logic replication for dynamically reconfigurable fpga partitioning," in *Proceedings of the 2002 international symposium on Physical design*, 2002, pp. 190–195.

[4] I. Bustany, G. Gasparyan, A. Gupta, A. B. Kahng, M. Kalase, W. Li, and B. Pramanik, "The 2023 mlcad fpga macro placement benchmark design suite and contest results," in *2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*. IEEE, 2023, pp. 1–6.

[5] C. Xu, W. Zhang, and G. Luo, "Analyzing the impact of heterogeneous blocks on fpga placement quality," in *2014 International Conference on Field-Programmable Technology (FPT)*. IEEE, 2014, pp. 36–43.

[6] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi *et al.*, "A graph placement methodology for fast chip design," *Nature*, vol. 594, no. 7862, pp. 207–212, 2021.

[7] X. He, T. Huang, L. Xiao, H. Tian, and E. F. Young, "Ripple: A robust and effective routability-driven placer," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 10, pp. 1546–1556, 2013.

[8] Y. Lin, S. Dhar, W. Li, H. Ren, B. Khailany, and D. Z. Pan, "Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[9] Q. Luo, X. Zang, Q. Wang, F. Wang, E. F. Young, and M. D. Wong, "A routability-driven ultrascale fpga macro placer with complex design constraints," in *2024 IEEE 32nd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2024, pp. 1–7.

[10] R. Cheng and J. Yan, "On joint learning for solving placement and routing in chip design," *Advances in Neural Information Processing Systems*, vol. 34, pp. 16 508–16 519, 2021.

[11] R. Cheng, X. Lyu, Y. Li, J. Ye, J. Hao, and J. Yan, "The policy-gradient placement and generative routing neural networks for chip design," *Advances in Neural Information Processing Systems*, vol. 35, pp. 26 350–26 362, 2022.

[12] Z. Jiang, E. Songhori, S. Wang, A. Goldie, A. Mirhoseini, J. Jiang, Y.-J. Lee, and D. Z. Pan, "Delving into macro placement with reinforcement learning," in *2021 ACM/IEEE 3rd Workshop on Machine Learning for CAD (MLCAD)*. IEEE, 2021, pp. 1–3.

[13] Y. Lai, Y. Mu, and P. Luo, "Maskplace: Fast chip placement via reinforced visual representation learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 019–24 030, 2022.

[14] Z. Tan and Y. Mu, "Hierarchical reinforcement learning for chip-macro placement in integrated circuit," *Pattern Recognition Letters*, 2024.

[15] Y. Lai, J. Liu, Z. Tang, B. Wang, J. Hao, and P. Luo, "Chipformer: Transferable chip placement via offline decision transformer," in *International Conference on Machine Learning*. PMLR, 2023, pp. 18 346–18 364.

[16] C.-Y. Chiang, Y.-H. Chiang, C.-C. Lan, Y. Hsu, C.-M. Chang, S.-C. Huang, S.-H. Wang, Y.-W. Chang, and H.-M. Chen, "Mixed-size placement prototyping based on reinforcement learning with semi-concurrent optimization," in *Proceedings of the 30th Asia and South Pacific Design Automation Conference*, 2025, pp. 893–899.

[17] M. A. Elgammal, K. E. Murray, and V. Betz, "Rlplace: Using reinforcement learning and smart perturbations to optimize fpga placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 8, pp. 2532–2545, 2021.

[18] F. Mahmoudi, M. A. Elgammal, S. G. Shahrouz, K. E. Murray, and V. Betz, "Respect the difference: Reinforcement learning for heterogeneous fpga placement," in *2023 International Conference on Field Programmable Technology (ICFPT)*. IEEE, 2023, pp. 152–160.

[19] C.-K. Cheng, A. B. Kahng, S. Kundu, Y. Wang, and Z. Wang, "Assessment of reinforcement learning for macro placement," in *Proceedings of the 2023 International Symposium on Physical Design*, 2023, pp. 158–166.

[20] H. Gu, J. Gu, K. Peng, Z. Zhu, N. Xu, X. Geng, and J. Yang, "Lamplace: Legalization-aided reinforcement learning based macro placement for mixed-size designs with preplaced blocks," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2024.

[21] S. Wang, D. R. S. Mamillapalli, T. Yang, and M. E. Taylor, "Fpga divide-and-conquer placement using deep reinforcement learning," in *2024 2nd International Symposium of Electronics Design Automation (ISEDA)*. IEEE, 2024, pp. 690–696.

[22] Xilinx, "Ultrascale architecture and product data sheet: Overview," 2022.

[23] H. Hu, D. Fang, W. Li, B. Yuan, and J. Hu, "Systolic array placement on fpgas," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[24] N. Kapre, "Implementing fpga overlay nocs using the xilinx ultrascale memory cascades," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2017, pp. 40–47.

[25] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.

[26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[27] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[28] Xilinx, "Vivado design suite properties reference guide (ug912)," 2023.

[29] Z. Xiong, R. S. Rajarathnam, Z. Jiang, H. Zhu, and D. Z. Pan, "Dreamplacefpga-mp: An open-source gpu-accelerated macro placer for modern fpgas with cascade shapes and region constraints," *arXiv preprint arXiv:2311.08582*, 2023.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.