TDM Signal Grouping and Package Pin Assignment for 2.5D Multi-FPGA Systems with Lookahead Placement

Jiarui Wang
School of Computer Science, School of Integrated Circuits
Peking University, Beijing, China

Runzhe Tao
School of Integrated Circuits
Peking University, Beijing, China

Jing Mai
School of Computer Science, School of Integrated Circuits
Peking University, Beijing, China

Xun Jiang
School of Integrated Circuits
Peking University, Beijing, China

Shenghua Wang
Cuiliu Yang
Haoyu Jie
Kan Huang
Richard Y. Sun
S2C Inc., Shenzhen, China

Yibo Lin*
School of Integrated Circuits, Institute of EDA
Peking University, Beijing, China
Advanced Innovation Center for Integrated Circuits, Beijing, China

Abstract

Large-scale multi-FPGA systems are widely used in modern emulation systems. As a critical part of the multi-FPGA system design flow, TDM signal grouping and package pin assignment directly impact the final placement and routing in the FPGA physical implementation. Poor pin assignments cause severe congestion and timing degradation at the logic-element level, while existing approaches lack accurate congestion modeling during system-level partitioning. This paper presents `Chimew`, a novel pin assignment methodology that leverages placement prototyping to predict logic-element-level congestion before physical implementation precisely. The proposed method co-optimizes signal grouping and pin placement through iterative refinement guided by congestion-aware cost functions derived from fast global placement. Experimental results demonstrate a 28% congestion reduction and up to 2.87ns less worst negative slack (WNS) compared to industrial tools while achieving a 100% success rate across diverse multi-FPGA benchmarks.

CCS Concepts

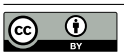
• **Hardware** → **Simulation and emulation**; **Reconfigurable logic and FPGAs**; **Placement**.

Keywords

FPGA, Multi-FPGA System, Emulation, Signal Grouping & Package Pin Assignment, FPGA Placement

ACM Reference Format:
Jiarui Wang, Runzhe Tao, Jing Mai, Xun Jiang, Shenghua Wang, Cuiliu Yang, Haoyu Jie, Kan Huang, Richard Y. Sun, and Yibo Lin. 2026. TDM Signal Grouping and Package Pin Assignment for 2.5D Multi-FPGA Systems with Lookahead Placement . In *Proceedings of the 2026 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '26), February 22–24, 2026, Seaside, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3748173.3779199

*Corresponding author



FPGA '26, Seaside, CA, USA

ACM ISBN 979-8-4007-2079-6/2026/02
https://doi.org/10.1145/3748173.3779199

1 Introduction

Field-Programmable Gate Arrays (FPGAs) have become essential components in modern emulation systems, offering reconfigurable hardware acceleration for design verification and validation. The increasing complexity of modern designs requires multi-FPGA emulation platforms, where modern 2.5D FPGAs (e.g., AMD Versal Premium VP1902 [1]) integrate multiple dies, referred to as Super Logic Regions (SLRs) in AMD/Xilinx architectures, interconnected through super long lines (SLLs) [2, 3]. As illustrated in Figure 1(a), cross-FPGA communication occurs through dedicated I/O channels between package pins. Due to limited I/O bandwidth, time-division multiplexing (TDM) [4] enables multiple signals to share a single channel within one system clock cycle, as demonstrated in Figure 1(b).

The computer-aided design (CAD) flow of multi-FPGA systems usually splits into two parts: system-level flow and logic-element-level flow. At the system level, the design is partitioned across devices, with cross-FPGA nets assigned routing topologies and TDM ratios. Recent approaches have evolved from traditional FPGA-level flows to die-level strategies, which enable more accurate performance estimation and are now widely adopted in both industrial tools like Synopsys Zebu [5] and S2C OmniDrive [6], and academic research [7, 8]. Following the TDM ratio assignment results, signals are grouped by their ratio and direction, then mapped to physical channels connecting package pins through a process known as TDM signal grouping and package pin assignment. The subsequent logic-element-level flow, typically executed by commercial tools such as AMD Vivado [9] and Altera Quartus Prime [10], performs placement and routing for logic elements (e.g., LUT, FF, DSP, BRAM, etc.) to generate final bitstreams.

The TDM signal grouping and package pin assignment process is a critical part of the multi-FPGA CAD process, as it directly builds a bridge between system-level flow and logic-element-level flow. The results of such a process directly assign the I/O placement on each

Email: jiaruiwang@pku.edu.cn, yibolin@pku.edu.cn

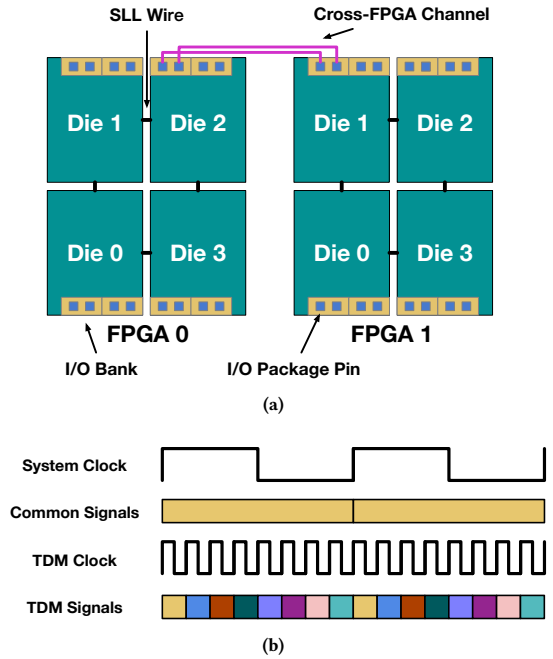


Figure 1: (a) An example of a multi-FPGA system with 2 FPGAs, and each FPGA has 4 dies. There are cross-FPGA channels between die 2 of FPGA 0 and die 1 of FPGA 1. (b) The clock waveforms of the system clock and the TDM clock.

FPGA and impact the intra-FPGA placement and routing process. However, prior works mainly focus on optimizing performance metrics at the system-level while ignoring logic-element-level. For example, [11] takes the FPGA-level routing results as inputs, and uses min-cost-flow for signal grouping and integer-linear-programming (ILP) for the TDM pin assignment process to optimize the total SLL crossing. Given that modern FPGAs accommodate large-scale design partitions on each device, limited consideration at the TDM stage can lead to severe placement and routing congestion or timing violations during subsequent physical implementation.

Considering the logic-element-level implementation at the TDM signal grouping and package pin assignment stage requires a lookahead logic-element-level placement. With the placement results, we can group TDM signals with closer positions to the same group, and assign each TDM signal group or common signal to closer package pins. Recent open-source FPGA CAD frameworks [12] can quickly generate global placement results, which creates opportunities to estimate routing congestion and guide system-level design flows. On the infrastructure side, FPGA Interchange Format [13] and RapidWright [14] have enabled broader access to modern large-scale commercial FPGA architectures. On the algorithmic side, many modern open-source FPGA placers have shown the ability to do placement on modern commercial FPGAs. Representative works include robust multi-electrostatics-based methods for handling cascaded macro groups and fence regions [12], multi-die-aware algorithms optimizing inter-die connections [15], and tools with native support for FPGA Interchange Format [16].

However, there are still three challenges to consider placement lookahead for the TDM signal grouping and package pin assignment stage. The first is that we need to consider both the system-level and the logic-element-level optimization targets, which challenge the effectiveness of the TDM signal grouping and package pin assignment algorithms. The second is that the placement results shall be highly correlated with the logic-element-level placement result of the commercial FPGA CAD tools, which challenges the quality of the placement results. The third is that the scale of FPGA devices used for modern emulation tasks is extremely huge compared to the scale of FPGA devices considered by academic FPGA placers, which challenges the efficiency of the FPGA placer.

In this paper, we propose `Chimew`, a TDM signal grouping and pin assignment framework considering logic-element-level intra-FPGA global placement results. Our framework supports the latest AMD Versal Premium 1902 FPGA [1], a large-scale FPGA with 2×2 dies (i.e., dies arranged in a 2-row-by-2-column configuration) and a capacity of 10^7 logic-elements. We load the die-level routing result and netlist as inputs, and leverage OpenPARF 3.0 [12], an open-source FPGA CAD tool with a GPU-accelerated global placement engine, as our basic placement platform. We develop novel algorithms for TDM signal grouping and pin assignment considering the placement results. To the best of our knowledge, this is the first work to consider placement lookahead at the TDM signal grouping and pin assignment stage. Our contribution is summarized as follows:

- We propose a system-level TDM signal grouping and pin assignment framework for multi-FPGA system design flow, considering logic-element-level placement impacts.
- We propose a novel placement-aware TDM signal grouping algorithm to effectively reduce the die crossing for cross-FPGA signals and a min-cost-flow model for TDM pin assignment based on the lookahead placement results.
- Compared to the industrial emulation tool [6], our framework acquires 28% less congestion and fixes an up to 2.87ns WNS with a 100% logic-element-level flow success rate, while the industrial tool fails on 3 of 6 cases due to the negative slack.

The rest of this paper is organized as follows. Section 2 describes the architecture of multi-FPGA systems and the problem formulation of the TDM signal grouping and pin assignment routing problem. Section 3 demonstrates the algorithm flow of our algorithm. Section 4 validates our routing algorithm with experimental results. Section 5 concludes the paper.

2 Preliminaries

In this section, we first introduce the background of modern emulation systems and the modern multi-FPGA system design flow. We then introduce the background of FPGA global placement and congestion estimation. Next, we demonstrate the design rules and the formulation of the TDM signal grouping and pin assignment problem.

2.1 Emulation System based on 2.5D Multi-FPGA Systems

2.1.1 Modern emualtion system with 2.5D multi-FPGA system Emulation is vital in chip design for pre-silicon verification, using

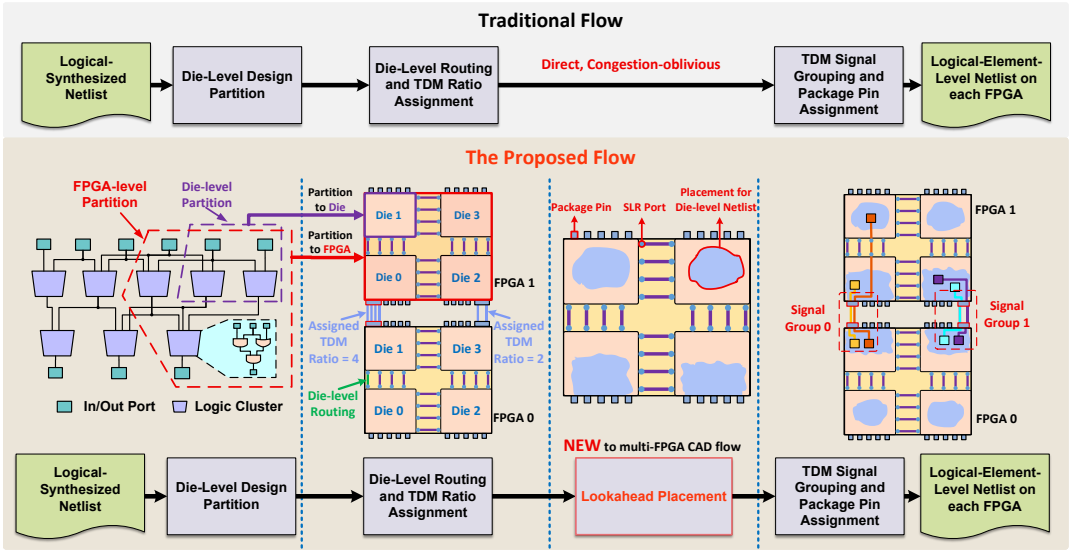


Figure 2: Modern multi-FPGA system CAD flow at die-level, with our work having a lookahead placement before TDM signal grouping and pin assignment.

hardware models to achieve near-real-time execution speeds. As design complexity grows, multi-FPGA systems with modern large-scale FPGAs have become the standard emulation platform.

Modern large-scale FPGAs employ multi-die architectures for increased capacity. While earlier designs like AMD VU19P [17] use a linear (1×4) die arrangement, recent FPGAs such as AMD VP1902 [1] adopt a 2×2 layout, improving inter-die routability through additional SLL connections between neighboring dies, as shown in Figure 1(a).

In multi-FPGA systems, inter-FPGA connections are established via I/O channels between package pins on different FPGAs. These channels support two signal types: TDM signals and common signals. TDM channels transmit multiple signals per system clock cycle with higher latency, whereas common-signal channels carry only one signal per cycle with lower latency.

2.1.2 System-level design flow of multi-FPGA systems To precisely estimate the performance of designs and effectively optimize the system frequency, modern system-level multi-FPGA CAD flows typically employ a die-level flow rather than the conventional FPGA-level flow. As shown in Figure 2, unlike the traditional FPGA-level flow, which partitions the netlist to different FPGAs, the input design is first fine-grained partitioned onto each die of different FPGAs in the die-level flow. For those die-crossing nets, system-level routing and TDM ratio assignment are then performed to assign the die-level routing topology and TDM ratios. After that, each cross-FPGA signal needs to be grouped by its TDM ratio and direction and assigned to an I/O channel between the two dies it connects in the typical die-level flow. In this work, we do an additional step to lookahead a global placement result before the TDM signal grouping process to estimate the logic-element-level placement and routing congestion and guide the TDM signal grouping and package pin assignment process, which is novel to typical system-level design flows.

2.2 Multi-Electrostatics Nonlinear FPGA Placement with Fence Region Constraints

Placement is crucial in the FPGA CAD flow, assigning logic elements to physical sites and significantly impacting final circuit performance. The process involves three stages: global placement (GP), legalization (LG), and detailed placement (DP). GP generates near-legal positions for logic elements, minimizing wirelength and reducing overlap, while LG and DP assign elements to legal sites and refine the solution. As the foundation of physical design, GP's quality ultimately determines routability and timing closure.

State-of-the-art nonlinear placers [12, 18–20] address density overflow in GP by modeling it as an electrostatic equilibrium problem. Logic elements are treated as charged particles, with the solver seeking a low-energy state that disperses cells and alleviates congestion. While effective for ASICs, this approach is less suitable for FPGAs due to their heterogeneous primitives (e.g., LUTs, FFs, DSPs, BRAMs), which require type-specific density models. To adapt it, elfPlace [21] extends the electrostatic formulation with multiple field solvers—one per resource type—capturing heterogeneity at the cost of increased computational complexity.

Additionally, modern FPGA placers must handle fence-region constraints, which partition the fabric into designer-specified regions to support hierarchical design and timing closure in large-scale designs. To enforce these constraints, OpenPARF 3.0 [12] enhances its electrostatic optimization with fence-aware auxiliary fields for each element type within every region. This force-based model guides primitives to their designated areas while preventing boundary violations, maintaining placement objectives without disrupting convergence.

2.3 Routing Congestion Estimation with RUDY

Routing congestion occurs when local routing demand exceeds available physical resources defined by the underlying architectural fabric, compromising design routability despite achieving

other timing and area targets. Early congestion estimation enables CAD flows to identify congestion-prone regions, guide logic elements toward routability-friendly spreading, and eliminate infeasible placement solutions before invoking computationally expensive routing. We employ the Rectangular Uniform wire DensitY (RUDY) model [22] to estimate routing utilization by uniformly distributing each multi-pin net's demand across its bounding box.

Consider a net n with its axis-aligned bounding box $\mathcal{B}_n$ positioned at lower-left corner $(x_n^{\text{ll}}, y_n^{\text{ll}})$ with width w_n and height h_n. The bounding box area is $\mathcal{A}_n = w_n h_n$. To estimate the consumption of routing resources, we compute the effective wire area as $\Omega_n = L_n p$, where L_n represents the estimated routing length (typically half-perimeter wire length, HPWL) and $p = \bar{p}/\ell$ captures the average wire pitch $\bar{p}$ distributed between the available routing layers ℓ.

RUDY models routing demand by assuming uniform wire distribution within each net's bounding box, yielding a constant density: $d_n = \frac{\Omega_n}{\mathcal{A}_n} = \frac{L_n p}{w_n h_n}$. This uniform distribution is mathematically expressed through an indicator function:

$$R_n(x,y) = \begin{cases} 1, & \text{if } (x,y) \in \mathcal{B}_n \\ 0, & \text{otherwise} \end{cases}$$

where $(x,y) \in \mathcal{B}_n$ means $x_n^{\text{ll}} \le x \le x_n^{\text{ll}} + w_n$ and $y_n^{\text{ll}} \le y \le y_n^{\text{ll}} + h_n$.

The global routing demand map aggregates contributions from all nets: $D_{\text{dem}}(x,y) = \sum_{n \in \mathcal{N}} d_n R_n(x,y)$. For congestion analysis, we integrate this demand over each bin Ω_g to obtain the expected load $\rho_g = \int_{\Omega_g} D_{\text{dem}}(x,y)\, dA$. By comparing ρ_g against the directional routing capacity of each bin, we can identify potential congestion hotspots before routing. The RUDY method is widely adopted because it is router-independent and avoids enumerating Steiner trees, providing a fast yet predictive congestion metric that integrates seamlessly into our multi-FPGA placement flow.

2.4 Design Rules

The TDM signal grouping and pin assignment process shall follow the following design rules:

TDM signal grouping rule. The cross-FPGA signals within a TDM signal group shall have the same TDM ratio and the same TDM direction. Also, the number of signals within a TDM signal group shall not exceed the TDM ratio of the group.

I/O channel direction rule. When assigning signals to an I/O bank, TDM signals shall be assigned first, while common signals shall be assigned after the TDM signals. Moreover, if there are two clusters of TDM signal groups with different directions assigned to the same I/O bank, groups with the same direction shall be assigned to nearby channels. Figure 3 shows an example of the difference between legal and illegal TDM pin assignment results.

2.5 Problem Formulation

We formally define the die-level TDM signal grouping and pin assignment problem as follows:

Problem. *Given the die-level system routing and TDM ratio assignment results for a design on a multi-FPGA system, assign each TDM signal a TDM signal group, and assign each TDM signal group and each common cross-FPGA signal an I/O channel and two package pins within two FPGAs, following the design rules above, and optimize the performance after the logic-element-level process (delay, congestion, etc.).*

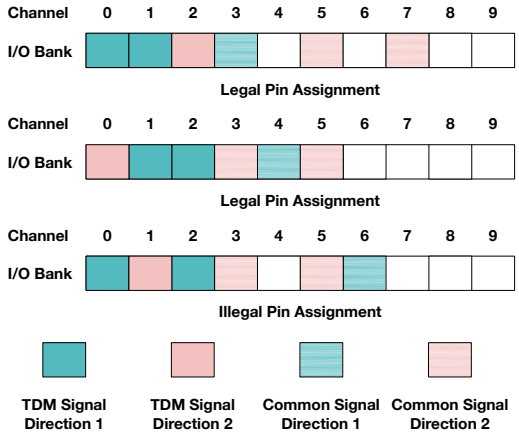


Figure 3: An example showing the TDM channel direction rule. The first 2 assignments are legal as they assign the TDM signal groups direction by direction, the last one is illegal as there is a TDM signal group by direction 2 between two TDM signals by direction 1.

3 Algorithm

In this section, we first introduce the overall flow of our TDM signal grouping and pin assignment algorithm. Then we demonstrate how we do the global placement for die-level netlists. Next, we explain our TDM signal grouping and pin assignment algorithms.

3.1 Overall Flow

As shown in Figure 4, our framework mainly consists of three parts: 1) logic-element-level global placement, 2) TDM signal grouping, and 3) package pin assignment.

The target of our logic-element-level global placement is to lookahead global placement results at the early system-level design stage. The result can not only be used to guide the following TDM signal grouping and pin assignment process, but can also be used to estimate the congestion level of the partitioned design. In this work, we use RUDY [22] to estimate the design congestion, and if the congestion level is larger than a certain threshold, we terminate the current CAD flow and re-partition the design.

As mentioned in the previous section, our TDM signal grouping and pin assignment process is to group all the cross-FPGA TDM signals and then assign all the cross-FPGA signals to package pins. In this work, we use the global placement results as guidance to enhance the process.

3.2 Speed-Boosted Lookahead Placement

Our logic-element-level global placement takes die-partitioned netlists as input and outputs placement lookahead results for each FPGA. This lookahead offers two key advantages in the multi-FPGA CAD flow: first, it identifies cross-FPGA signals with similar logic positions (e.g., fanins of a logic element tend to cluster together);

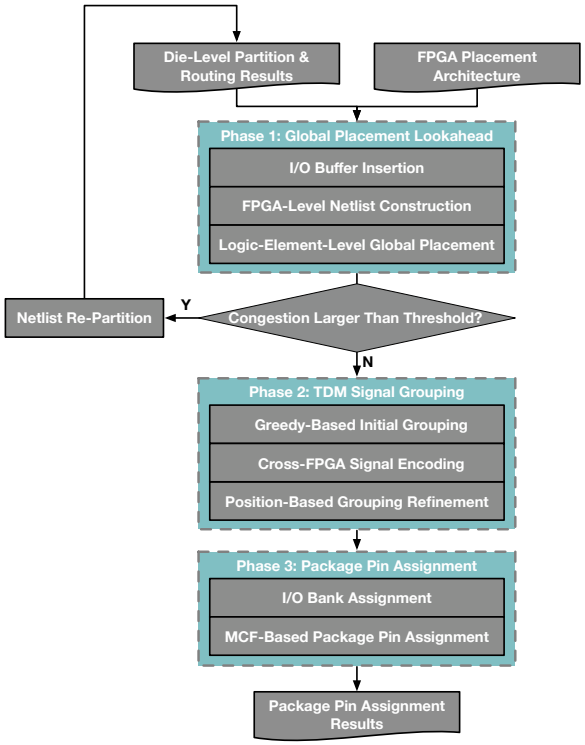


Figure 4: The overall flow of our TDM signal grouping and package pin assignment process.

second, it detects potential congestion early, enabling designers to terminate highly congested designs before further investment.

As shown in Figure 4, our global placement consists of three steps. Since the partitioned netlists contain only core logic without I/O buffers, we first invoke Vivado's *link_design* command to map each sub-netlist to target FPGA devices and insert I/O buffers. Next, we load all netlists along with system-level routing and TDM ratio assignments to generate complete netlists for each FPGA in the system. Finally, we perform global placement using our accelerated placer, which incorporates optimizations to the original OpenPARF framework to handle the large-scale designs typical of emulation tasks.

3.2.1 FPGA-level netlist construction FPGA placers take the netlist of the FPGA-level and the FPGA placement architecture as their inputs. As our input netlist is at the die-level, we need to construct an FPGA-level netlist before we call our placer. The FPGA-level netlist is merged from each die-level netlist with the following modification from the system-level routing results.

Fence region for each die. To restrict that each logic element is placed on the die it is partitioned into, we regard each die as a fence region and use fence region constraints to restrict the die each logic element shall be placed on.

Cross-FPGA signals. We call Vivado to add input/output buffers at each input/output port of each die-level netlist. For those cross-FPGA signals, as shown in Figure 5(a), we move their corresponding I/O buffers to the die that the signal inputs to/outputs from the FPGA based on the result of system-level routing.

Cross-die signals. The previous Vivado process also adds I/O buffers to the ports corresponding to die-crossing signals. To precisely estimate the die-crossing congestion, as shown in Figure 5(b), we remove those I/O buffers and add die-crossing nets to those signals.

Bypass signals. An FPGA in a multi-FPGA system can also be used to bypass signals between two FPGAs. Die-level netlists do not have information about such signals. Thus, as shown in Figure 5(c), we add an input buffer to the die that the signal enters, and add output buffers to the dies that the signal exits for bypass signals following the system-level routing results.

3.2.2 Global placement acceleration Existing academic open-source placers are designed for single-die FPGAs, which have far fewer resources than modern multi-die platforms. For instance, the 4-die AMD VP1902 [1] FPGA is roughly 20 times larger than the single-die AMD XCVU3P [23] FPGA used in MLCAD 2023 benchmarks [24]. This expanded capacity also leads to much larger netlists, imposing significant computational demands on placement algorithms. To efficiently generate placement results for downstream tasks such as TDM signal grouping and pin assignment, we propose the following enhancements to the baseline OpenPARF 3.0 placer [12].

Accelerated placer engine. To address runtime challenges in large-scale netlists, we implement a hardware-adaptive acceleration mechanism in the core placement engine. Our approach implements optimized density and wirelength kernels that leverage hardware-level vectorization for enhanced computational efficiency. Combined with adaptive lambda scheduling and shared density-map caching, this hardware-level optimization enables faster electrostatic descent convergence while preserving placement quality across large-scale multi-die density maps. We elaborate on these techniques in the following subsections.

The adaptive lambda scheduling strategy introduces two key enhancements. First, we adjust the initial lambda factor between wirelength and density objectives, deliberately biasing the optimization toward density constraints during early placement stages. Second, we accelerate the lambda update rate to expedite the transition between optimization phases. This configuration targets rapid lookahead scenarios by prioritizing density convergence while maintaining guidance quality for subsequent TDM signal grouping and package pin assignment workflows.

The shared density-map caching mechanism further enhances computational efficiency by identifying and eliminating redundant calculations within the baseline density map computation pipeline. Our caching approach pre-computes and reuses density maps for frequently accessed spatial regions, achieving measurable speedup in multi-die placement scenarios where regions are repeatedly evaluated.

Optimized data marshalling. The heterogeneous C++/Python framework we base on [12] for large-scale FPGA design flows exhibits a critical data marshalling bottleneck. The bottleneck requires a careful rethinking of memory management strategies. To address this challenge, we propose a selective zero-copy data bridge that directly exposes C++ data structures as shared memory tensors, eliminating redundant serialization overhead. This approach employs a Structure of Arrays (SoA) memory layout that maximizes spatial locality and enhances cache utilization patterns, which is

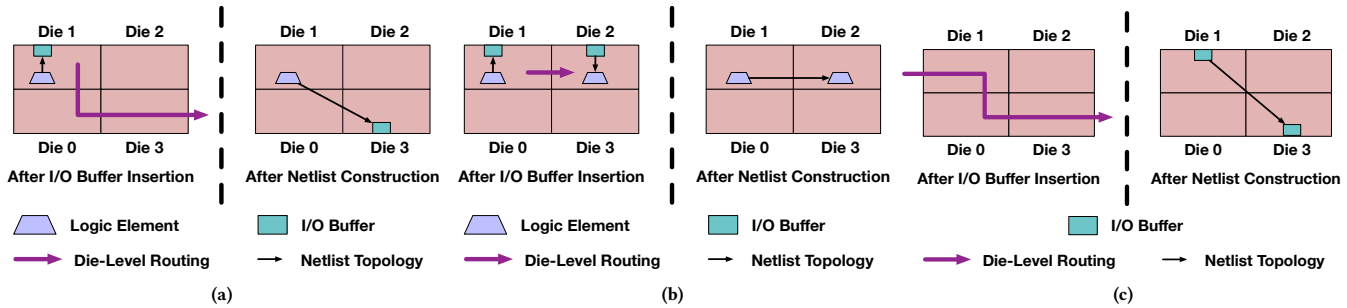


Figure 5: Examples showing how we process the special signals during FPGA-level netlist construction. (a) Cross-FPGA signals. (b) Cross-die signals. (c) Bypass signals.

especially useful for accessing sparse connectivity matrices common in FPGA netlists. This optimized memory organization reduces database initialization time by two orders of magnitude for designs exceeding 100K logic elements. This zero-copy method is particularly effective for placement tasks that frequently access the same data structures.

Placement layout approximation. To handle the large-scale multi-die FPGA layout, we made assumptions about the placement layout. We first modify the I/O capacity and area to deal with large-scale I/O buffers. With the TDM technique, the number of input/output ports of each sub-design on each FPGA is much more than the number of I/O buffers on each FPGA. As the TDM IP is inserted after the package pin assignment process, we need to approximate the I/O architecture before global placement. To quickly generate a global placement result, we increase the capacity of each I/O tile and decrease the area demand of each I/O buffer. We also modify the capacity of CLE tiles. In modern AMD VP1902 FPGA devices [1], there are a SLICEL and a SLICEM within a CLE tile. The LUTs in SLICEM can be used to implement LUTRAM while the LUTs in SLICEL do not have such an ability. Although OpenPARF [25] has the ability to deal with the differences between SLICEL and SLICEM by constructing different electrostatic fields, it takes much more time to get the placement results. As the number of LUTRAM within a netlist is much less than the number of common LUTs, we ignore the difference between SLICEL and a SLICEM, and assign each LUTRAM an area demand as the number of LUT5 it uses to quickly generate the global placement results. Such approximations allow us to obtain the global placement results with high accuracy quickly.

3.3 Placement-Aware TDM Signal Grouping Algorithm

The target of our TDM signal grouping process is to assign each TDM signal a signal group following the TDM signal grouping rule. The TDM signals within a TDM signal group will be transmitted by the same package pin channel.

As shown in Figure 6(a) and Figure 6(b), if a TDM signal group crosses more dies in the FPGA-to-FPGA pair, it will lead to a large SLL usage and may cause congestion during the logic-element-level place and route process. Thus, we need to optimize the total group die crossing during the TDM signal grouping process. To deal with this problem, we propose a novel encoding for the cross-FPGA signals and use an encoding-based algorithm to resolve the TDM signal grouping problem. Also, to reduce the congestion and delay, we consider the global placement lookahead results to further optimize the process.

3.3.1 Cross-FPGA signal encoding and problem formulation To further optimize the TDM signal grouping results, we propose a binary encoding for each cross-FPGA TDM signal. The encoding of each signal can help us quickly identify signals crossing the same SLLs, which can be used to avoid the circumstances in Figure 6(b). As shown in Figure 6(c), we encode each cross-FPGA signal as an 8-bit binary number representing the SLLs crossed by the signal. The lower 4 bits of the encoding represent the SLLs the signal crosses on the source FPGA (FPGA 0 in Figure 6(c)), and the upper 4 bits represent the SLLs on the sink FPGA (FPGA 1 in Figure 6(c)). If a bit of the encoding is 1, it means that the signal crosses the SLL that the bit represents.

With the encoding of a TDM signal, we can calculate its die crossing by the bits set to 1 in the encoding. For example, signal 1 in Figure 6(c) has 2 die crossings (die 0 to die 3 in FPGA 0, and die 0 to die 1 in FPGA 2), and there are 2 bits of 1 in its encoding.

In the TDM signal grouping process, if we group two TDM signals together, the encoding of the TDM signal group will be the result of `OR` of the two signals. For example, in Figure 6(c), if we group signal 1 and signal 2 together, the encoding of the signal group will be `00100001`. With the encoding of a TDM signal group, we can calculate its die crossing by the bits of 1 in the encoding. For example, the group of signal 1 and signal 2 in Figure 6(c) crosses 2 SLLs, and there are 2 bits of 1 in its encoding.

We hope each group crosses the least number of dies as possible. As the number of signals within a TDM signal group cannot exceed the TDM ratio, and the number of FPGA-to-FPGA channels is limited, we conclude the problem formulation as follows:

Given a set S of TDM signals with the same direction and the same TDM ratio r, group the signals into $\lceil \frac{\|S\|}{r} \rceil$ groups:

$$\begin{aligned} &\min_{x_{s,g}} \sum_{g \in G} \mathrm{ONES}\Big(\bigvee_{s \in S} \mathrm{encode}(s) \cdot x_{s,g}\Big), \\ &s.t. \sum_{s \in S} x_{s,g} \le r \qquad\qquad \forall g \in G, \end{aligned} \tag{1}$$

where G is the set of the signal groups, $x_{s,g}$ is the binary variable representing whether signal s is grouped into group g. ONES represents the bits of 1 in a binary number, and encode represents our signal encoding function.

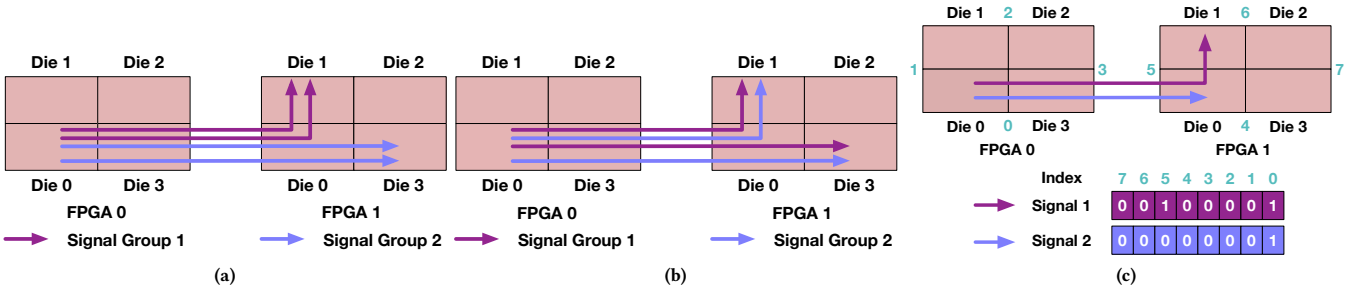


Figure 6: An example showing how TDM signal routing results can impact the SLL congestion. (a) A good TDM signal grouping with small and balanced die crossing for each TDM signal group. (b) A TDM signal grouping result with a large die crossings. (c) An example showing the signal encoding of two cross-FPGA TDM signals.

3.3.2 Encoding-based grouping algorithm It is difficult to directly find an optimal solution to the problem (1), thus, we propose a greedy-based algorithm to quickly find an approximate solution. Note that the number of die crossing of a TDM signal group is dominated by the TDM signal with the maximum die crossing. For example, in Figure 6(c), if we have signal 1 in a signal group, the signal group will cross at least 2 SLLs. However, if we group signal 1 and signal 2 in Figure 6(c) together, the number of the die crossing will not increase, as the encoding of signal 1 can cover the encoding of signal 2. Based on such properties, we list our encoding-based algorithm in Algorithm 1.

Algorithm 1: TDM Signal Grouping for Signals with the Same Direction and TDM Ratio

Input: Set S of TDM signals, TDM ratio r
Output: TDM signal groups G

1 Calculate the signal encoding.
2 Sort S based on the number of die crossings in decreasing order.
3 **while** *Exists a not grouped signal* s **do**
4 Initialize a new group g.
5 $t \leftarrow \text{encode}(s)$.
6 **while** $|g| < r$ ***and*** *have remaining signals* **do**
7 select signal s' with the nearest encoding to t.
8 $t \leftarrow t \vee \text{encode}(s')$.
9 add s' to g.

Based on the TDM signal grouping rule, the TDM signals within a TDM signal group shall have the same direction and TDM ratio. Thus, we extract a set S of TDM signals between 2 dies on different FPGAs with the same TDM ratio r and direction, and calculate their signal encoding first (line 1). As those signals with a large number of die crossing dominate the number of die crossing in the final results, we sort all the signals based on the bits of 1 in their encodings in decreasing order (line 2). If two signals have the same bits of 1 in their signal encodings, signals with higher bits of 1 will have a smaller index.

Our TDM signal grouping algorithm builds the signal groups iteratively. As the signals crossing more dies dominate the final results, for an empty group, we set the target die crossing as the die crossing of the first remaining sorted signal. We then iteratively add the signals with the nearest encoding to the signal group until the size of the signal group reaches the TDM ratio r, or all the signals have been grouped.

We then demonstrate how we find the nearest signal in line 7. For a target signal encoding, we first select those signals with the same encoding and add them to the signal group. After all the signals with the same encoding have been grouped, we then check signals with the encodings that can be totally covered by the target encoding. To decrease the number of die crossing in the final results, we add the signals with more die crossing and fewer signals with the same encoding to the group first. For example, if the current target encoding is `00111001`, and there are 3 signals with encoding `00011001`, 5 signals with encoding `00111000`, and 3 signals with encoding `00011000`. We will first add those signals with encoding `00011001` to the group, then those signals with encoding `00011100`, and finally those `00011000` signals. After all the signals whose encoding can be covered by the target encoding are inserted into the signal group, if the size of the group does not reach the TDM ratio, we then increase the bits of 1 in the target encoding. We check all the remaining signals to choose those signals with the least different bits of 1 in their encodings and add them to the signal group to fulfill the signal group.

We then refine the initial signal grouping results based on the positions of each logic element in the global placement lookahead results. Our refinement process shall not increase the die crossing of each TDM signal group, and shall make those nearby signals cluster into the same group.

To refine the initial grouping results, we sort all the TDM signals with the same encoding by the increasing order of the y-axis of the source logic element of each TDM signal. Our TDM signal grouping refinement process swaps the signals within each group of the initial grouping results, and makes groups with similar y-axis in the same group. Such an assignment can make the final package pin assignment results have less SLL congestion.

3.4 Min-Cost-Flow-based Package Pin Assignment

The target of our package pin assignment process is to assign each cross-FPGA common signal or TDM signal group a channel consisting of 2 package pins on different FPGAs. To handle the I/O channel direction rule, we split the package pin assignment process into 2 stages. We first assign each signal to a pair of I/O banks where the

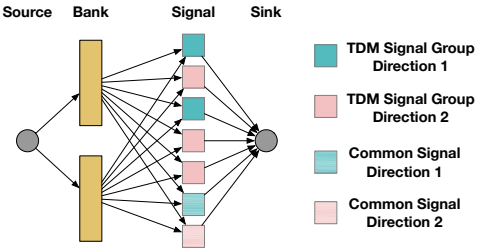


Figure 7: The min-cost-flow model of our bank assignment process

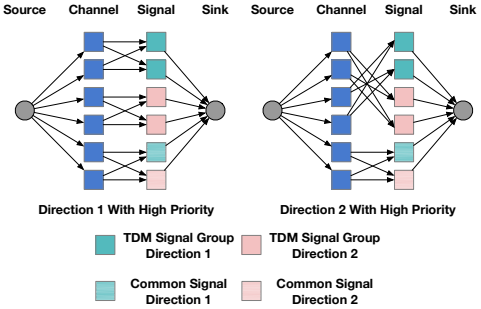


Figure 8: The min-cost-flow model of our pin assignment process.

package pins are located. Next, we assign each signal a cross-FPGA channel bank by bank. We use the min-cost flow model for the two steps to obtain the package pin assignment results.

3.4.1 I/O bank assignment As shown in Figure 7, our I/O bank assignment process is based on a min-cost flow model. Besides the source vertex and sink vertex, there are two kinds of vertices in our model. The first kind of vertices represent the banks (bank vertices), while the second kind of vertices represent the TDM signal groups or common signals (signal vertices). There are three kinds of edges in the model. The edges from the source vertex to the bank vertices have a capacity of the number of available channels within each bank pair and have zero cost. The edges from the bank vertices to the signal vertices have a capacity of 1, and the cost of those edges is calculated by the methods described in Section 3.4.3. The edges from the signal vertices to the sink vertex have a capacity of 1 and have no edge cost. After solving the min-cost-flow model, each signal vertex will have exactly one input edge with flow demand, and the signal or signal group will be assigned to the bank from which the edge comes.

3.4.2 Package pin assignment Our package pin assignment process is shown in Figure 8. To handle the I/O channel direction rule, we construct two minimum-cost-flow models for a bank-to-bank pair during this process. In each model, we select a direction of TDM signal groups and give them a higher priority. After solving the two min-cost flow problems, we check the cost of the two solutions and choose the solution with the smaller cost as the package pin assignment results.

The min-cost-flow model for package pin assignment is similar to the model of I/O bank assignment. There are also two kinds of vertices besides the source vertex and sink vertex. Channel vertices are used to represent each channel between two package pins within the bank pair, and signal vertices are used to represent common signals or TDM signal groups assigned to the bank pair. The edges from the source vertex to the channel vertices and the edges from the signal vertices to the sink vertex have a capacity of 1 and no edge cost. To satisfy the I/O channel direction rule, there are only edges from each channel vertex to available signal vertices. For example, suppose we have direction 1 with high priority. In that case, we have the first few channel vertices connected to signal vertices representing TDM signal groups by direction 1, and the next few channel vertices connected to signal vertices representing TDM signal groups by direction 2. The other channel vertices are connected to the vertices representing the common signals. Those edges have a capacity of 1, and their edge costs are also calculated by the algorithms in Section 3.4.3.

Note that the min-cost-flow model described in this section is actually a maximum-weighted bipartite-graph matching problem. Thus, we use the K-M algorithm [26] to get the solution and assign each signal or signal group to the package pin channel they are matched to in the solution.

3.4.3 Edge Cost Calculation We list how we calculate the cost of the edges to the signal vertices in our min-cost-flow models in Algorithm 2. Note that in Algorithm 2, we regard a common signal as a signal group with a single signal. The edge cost function is based on the global placement lookahead results and the locations of the I/O banks and the package pins on the FPGA layout.

Algorithm 2: Edge Cost Calculation

Input: Signal group g, I/O bank pair or package pin pair p
Output: Edge cost $c_{g,p}$

1 $c_{out} \leftarrow 0, c_{in} \leftarrow 0$.
2 **foreach** *Signal* $s \in g$ **do**
3 $\quad c_{out}$ += $dist(s_{\text{fanout}}, p_{out})$.
4 $\quad N_{\text{fanin}} \leftarrow 0, d_{\text{fanin}} \leftarrow 0$.
5 $\quad$ **foreach** *fanin logic-element* s_{fanin} *of* s **do**
6 $\quad\quad d_{\text{fanin}}$ += $dist(s_{\text{fanin}}, p_{in})$.
7 $\quad\quad N_{\text{fanin}}$ += 1.
8 $\quad c_{in}$ += $\frac{d_{\text{fanin}}}{N_{\text{fanin}}}$.
9 $c_{g,p} \leftarrow w_{out} c_{out} + w_{in} c_{in}$.

As a cross-FPGA signal connects two FPGAs, we need to consider the placement results synergistically. Thus, the edge cost is a weighted sum of the output cost c_{out} and the input cost c_{in} (line 9). In this work, we set w_{out} as 1 and w_{in} as the reciprocal of the number of fanin logic elements. A cross-FPGA signal can have one fanout logic-element and several fanin logic-elements. The output cost c_{out} is the sum of the Manhattan distance between each fanout logic-element s_{fanout} and the output bank or package pin p_{out} (line 3). To deal with the imbalanced distribution of the number of fanin logic-elements of each signal, the input cost is the sum of the average of the Manhattan distance between each fanin logic-element and the input bank or package pin (lines 4-8).

4 Experimental Results

In this section, we first introduce our experimental setups. Then, we validate the effectiveness of our TDM signal grouping and package pin assignment algorithms. We also present experiments on our

Table 1: The number of FPGAs (#FPGAs), the number of dies (#Dies), the number of cross-FPGA signals (#Conns), and the number of logic-elements (#Bels) of our benchmarks.

Design	#FPGAs	#Dies	#Conns	#Bels
IND01	4	16	59.73K	5.53M
IND02	4	16	47.92K	17.55M
IND03	4	16	31.61K	23.41M
IND04	4	16	53.49K	23.46M
IND05	16	64	66.99K	30.21M
IND06	8	32	53.9K	54.06M

placer to demonstrate the efficacy and necessity of global placement lookahead at the early system-level design stage.

4.1 Experimental Setups

We implement our TDM signal grouping and package pin assignment algorithms in C++, while our placer is implemented based on OpenPARF 3.0 [12] in C++, CUDA, and Python. Our experiments were conducted on a 2.10 GHz Intel Xeon Gold 6230 CPU platform with 512 GB of memory and an NVIDIA A40 GPU. We do the I/O buffer insertion process using Vivado 2025.1 [9] and generate the FPGA placement architecture file using the FPGA Interchange Format [13] and RapidWright [14].

We collect 6 industrial designs as our benchmarks from an industrial vendor [27], and their statistics are listed in Table 1. The FPGA devices used in all 6 cases are the AMD Versal Premium 1902 [1] device. The number of FPGA devices can be up to 16, with each FPGA device having 4 dies. The number of logic-elements can be at most 5×10^7. As the partition for each device is not balanced, there can be at most around 10^7 logic-element-level cells within an FPGA-level netlist. The scale of each FPGA-level netlist is way much larger than common academic benchmarks like ISPD 16&17 benchmarks [28, 29] and MLCAD 23 benchmarks [24], where the scales of netlists in those benchmarks are at most 10^6.

4.2 Validations on TDM Signal Grouping and Package Pin Assignment

We collect the reports of Vivado, and list the results of the SLL routing congestion, post-routing delay, and whether the logic-element-level flow of Vivado reaches the 24-hour time limit in Table 2. We also collect the industrial emulation tool from the same industrial vendor. We use the die-level routing and TDM ratio assignment results of the industrial tool as the input of our framework, and generate the package pin assignment results. We also use the industrial tool to generate the package pin assignment results using the same die-level partition results. We collect both sets of results and use Vivado 2025.1 to finish the intra-FPGA logic-element-level design flow by running the default logic-element-level flow of the industrial tool. Our logic-element-level flow is run on the cloud computing clusters provided by the industrial vendor, and we run the flow of each FPGA in parallel. Normally, the runtime of the logic-element-level flow will be around 8 to 16 hours, as the scale of the design is very large, and there are some additional steps, like inserting the TDM IP in the industrial flow. There is a 24-hour runtime limitation for the logic-element-level flow, as the default Vivado setting of the industrial tool will try to fix the negative slack by significantly more rip-up and reroute iterations.

Our average maximum placement congestion level comes from the post-placement congestion estimation by Vivado. Vivado estimates the SLL congestion at the beginning of the routing phase, and reports the SLL congestion regions with a utilization threshold of 1.000 for each SLR cut. We collect the average overflow utilization of each SLR cut of each FPGA, and list them in Table 2. As there are multiple clocks within a multi-FPGA design, to estimate the post-routing delay of each design, we collect the WNS and the total negative slack (TNS) reported by Vivado. The WNS is the smallest WNS of all the FPGA-level designs reported by Vivado, and the TNS is the summary of all the TNS of different FPGA-level designs. If the WNS is 0, it means that there are no slack violations in the routing results. Additionally, due to the significant fluctuations in the offload of cloud computing clusters, the runtime of the logic-element-level flow of both the results of the industrial tool and our framework can vary by hours. Thus, we cannot collect an exact runtime of our logic-element-level flow. Usually, the runtime of the logic-element-level flow can be 8 to 16 hours using both the system-level results of the industrial tool and our frameworks.

As shown in Table 2, we achieve a 28% reduction in SLL routing congestion and up to 2.87ns WNS reduction compared to the industrial tool. Furthermore, with the default setting of the industrial tool, the results generated by the industrial tool have negative slack on 3 of 6 benchmarks, and our results do not have any negative slack, which shows the effectiveness of our TDM signal grouping and pin assignment process. Note that although the SLL congestion of the results of the industrial tool in design IND05 is smaller than our results, the industrial tool fails to generate a legal result with no negative slack on that design.

Note that [11] also does the TDM signal grouping and package pin assignment process. We do not compare with their work for two reasons. The first is that their work is based on the FPGA-level multi-FPGA system CAD flow, which is totally different from the modern die-level design flow. Also, the FPGA device used in their works has a 1×4 die shape, which is different from the 2×2 die shape of the AMD VP1902 device. The different die shapes of FPGA devices will make it impossible to use the methods in [11] to calculate the die crossing of TDM signal grouping. Thus, we do not compare our algorithms with theirs in this paper.

We also collect the runtime of the different steps in our framework, and the total runtime of our framework, and list them in Table 3. As shown in Table 3, our framework takes at most 69 minutes to get the results. Most of our runtime is used for data initialization and global placement lookahead, which take 40% and 58% of the total runtime. As the logic-element-level flow for each FPGA is around 8 to 16 hours, it is reasonable to use the runtime to do global placement for finding out the possible congestion at the early design stage. Note that we do the global placement for each FPGA serially. As the task of global placement for each FPGA is independent of the others, the global placement process can be further accelerated. Furthermore, as the TDM signal grouping and package pin assignment process in the industrial tool does not have the global placement step, and it is difficult to extract the exact runtime of the process in industrial tools, we do not compare our runtime with the industrial tool in this paper.

Table 2: The SLL routing congestion (SLL Cong.), the post routing worst negative slack (WNS, ns), total negative slack (TNS, ns), and whether the runtime of the logic-element-level flow exceeds the 24-hour time limitation (TLE) on the industrial benchmarks between the industrial tool and our framework.

Design	Industrial Tool					Ours				
	SLL Cong.		Delay		TLE	SLL Cong.		Delay		TLE
	Overflow	Ratio	WNS	TNS		Overflow	Ratio	WNS	TNS	
IND01	1.50%	1.36	**0.000**	**0.000**	**N**	**1.10%**	**1.00**	**0.000**	**0.000**	**N**
IND02	1.88%	2.04	**0.000**	**0.000**	**N**	**0.92%**	**1.00**	**0.000**	**0.000**	**N**
IND03	0.65%	1.02	2.870	15.878	Y	**0.64%**	**1.00**	**0.000**	**0.000**	**N**
IND04	1.32%	1.31	**0.000**	**0.000**	**N**	**1.01%**	**1.00**	**0.000**	**0.000**	**N**
IND05	**1.80%**	**0.87**	1.244	3.677	Y	2.08%	1.00	**0.000**	**0.000**	**N**
IND06	1.35%	1.07	0.845	11.913	Y	**1.26%**	**1.00**	**0.000**	**0.000**	**N**
Norm.	-	1.28	-	-	3/6	-	**1.00**	-	-	**0/6**

Table 3: The runtime (s) of the data initialization (INIT), the global placement (GP), and the TDM signal grouping and package pin assignment (SG & PA) process of our framework, and the total runtime (s) of our framework.

Design	INIT	GP	SG & PA	Total
IND01	166	387	6	561
IND02	534	692	24	1250
IND03	674	923	32	1629
IND04	683	915	32	1630
IND05	966	1484	43	2493
IND06	1885	2168	87	4140
Norm.	0.40	0.58	0.02	1.00

4.3 Validations on Lookahead Placement

The target of the experiments about global placement lookahead is to prove the effectiveness and efficiency of our global placer. To prove the effectiveness of our global placer, we collect the post-placement wirelength reported by Vivado and the Half-Perimeter WireLength (HPWL) of our global placement results on our benchmarks, and show their correlations in Figure 9. As shown in Figure 9, the wirelength reported by Vivado and our placer are highly correlated, which shows the effectiveness of our global placer. Note that the wirelength unit and model used in Vivado and our placer are different, resulting in a gap of two orders of magnitude between the wirelength reported by Vivado and our placer.

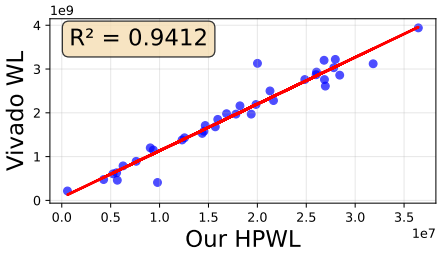


Figure 9: The correlation between the HPWL reported by our placer and the post-placement wirelength reported by Vivado.

To show the efficiency of our modification to the original OpenPARF 3.0 [12], we list the global placement runtime of the original OpenPARF 3.0 and our placer in Table 4. Due to the extremely large scale of the FPGA layout and netlist, the original OpenPARF 3.0 fails to generate the global placement results for 10 FPGA-level netlists within 4 designs, and the modified version successfully generates all the global placement results. We list the average runtime and the maximum runtime of each design, and as there are system scheduling runtimes, the product of the average runtime of our placer and the number of FPGAs within each design is smaller than the placer runtime reported in Table 3. As Table 4 shows, our speed-boosted global placer has a more than 4× speedup compared to the original OpenPARF 3.0 and shows a high efficiency.

Table 4: The number of failed cases and the average (AVG) and maximum (MAX) global placement runtime (s) for each FPGA-level design of the original OpenPARF 3.0 [12] and our speed-boosted global placer on our benchmarks.

Design	Original OpenPARF 3.0 [12]			Ours		
	FAILED	AVG	MAX	FAILED	AVG	MAX
IND01	**0/4**	389	649	**0/4**	**77**	**104**
IND02	2/4	430	525	**0/4**	**115**	**175**
IND03	1/4	643	766	**0/4**	**157**	**211**
IND04	**0/4**	667	805	**0/4**	**152**	**228**
IND05	1/16	396	744	**0/16**	**86**	**156**
IND06	6/8	671	764	**0/8**	**174**	**229**
Norm.	10/40	4.29	4.08	**0/40**	**1.00**	**1.00**

5 Conclusion

In this paper, we propose `Chimew`, a TDM signal grouping and package pin assignment framework for modern large-scale multi-FPGA systems. We conduct speed-boosted global placement lookahead at the early system-level design stage and utilize the global placement results to optimize the TDM signal grouping and package pin assignment processes. We propose a novel TDM signal grouping formulation and a min-cost-flow-based package pin assignment algorithm. Compared to the industrial tool, our framework can achieve a 28% less congestion and fix at most 2.87ns WNS with a 100% successful rate logic-element-level placement and routing flow.

Acknowledgements

This work is supported in part by the Natural Science Foundation of Beijing, China (Grant No. Z230002), and the 111 Project (B18001).

References

[1] AMD, "Versal Premium VP1902 Adaptive SoC," 2025. https://www.amd.com/en/products/adaptive-socs-and-fpgas/versal/premium-series/vp1902.html.

[2] R. Raikar and D. Stroobandt, "Multi-die heterogeneous FPGAs: How balanced should netlist partitioning be?," SLIP '22, 2023.

[3] C. Ravishankar, D. Gaitonde, and T. Bauer, "Placement strategies for 2.5D FPGA fabric architectures," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 16–164, 2018.

[4] J. Babb, R. Tessier, M. Dahl, S. Hanono, D. Hoki, and A. Agarwal, "Logic emulation with virtual wires," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 6, pp. 609–626, 1997.

[5] Synopsys, "Emulation Systems | System Verification," 2024. https://www.synopsys.com/verification/emulation.

[6] X. Zhang, "How do logic simulation, emulation, and FPGA prototyping work?," 2023. https://www.s2cinc.com/resources/lit/en/wp/s2c-how-do-logic-simulation-emulation-and-fpga-prototyping-work.pdf.

[7] J. Wang, Y. Liu, and Y. Lin, "Synergistic die-level router for multi-FPGA system with time-division multiplexing optimization," in *2025 62nd ACM/IEEE Design Automation Conference (DAC)*, pp. 1–7, 2025.

[8] C. Huang, P. Chu, S. Bi, R. Sun, and H. You, "System routing and TDM assignment optimization in multi-2.5D FPGA-based prototyping systems," in *2024 2nd International Symposium of Electronics Design Automation (ISEDA)*, pp. 324–331, 2024.

[9] AMD, "AMD Vivado Design Suite," 2025. https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vivado.html.

[10] Altrea, "Quartus® Prime Design Software | Altera® FPGA," 2025. https://www.altera.com/products/development-tools/quartus.

[11] Y.-C. Liao and W.-K. Mak, "Pin assignment optimization for multi-2.5D FPGA-based systems with time-multiplexed I/Os," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 3, pp. 494–506, 2021.

[12] J. Mai, J. Wang, Y. Chen, Z. Guo, X. Jiang, Y. Liang, and Y. Lin, "OpenPARF 3.0: Robust multi-electrostatics based FPGA macro placement considering cascaded macros groups and fence regions," in *2024 2nd International Symposium of Electronics Design Automation (ISEDA)*, pp. 374–379, 2024.

[13] Chips-Alliance, "FPGA-Interchange-Schema," 2025. https://github.com/chipsalliance/fpga-interchange-schema.

[14] C. Lavin and A. Kaviani, "RapidWright: Enabling custom crafted implementations for FPGAs," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 133–140, April 2018.

[15] Z. Di, R. Tao, J. Mai, L. Chen, and Y. Lin, "LEAPS: Topological-layout-adaptable multi-die FPGA placement for super long line minimization," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 71, no. 3, pp. 1259–1272, 2024.

[16] R. S. Rajarathnam, M. B. Alawieh, Z. Jiang, M. Iyer, and D. Z. Pan, "DREAMPlaceFPGA: An open-source analytical placer for large scale heterogeneous FPGAs using deep-learning toolkit," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 300–306, 2022.

[17] AMD, "Virtex UltraScale+ VU19P FPGAs," 2025. https://www.amd.com/en/products/adaptive-socs-and-fpgas/fpga/virtex-ultrascale-plus-vu19p.html.

[18] J. Lu, P. Chen, C.-C. Chang, L. Sha, D. J.-H. Huang, C.-C. Teng, and C.-K. Cheng, "ePlace: Electrostatics based placement using Nesterov's method," in *Proceedings of the 51st Annual Design Automation Conference (DAC)*, (New York, NY, USA), pp. 1–6, ACM, 2014.

[19] C.-K. Cheng, A. B. Kahng, I. Kang, and L. Wang, "RePlAce: Advancing solution quality and routability validation in global placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 38, no. 9, pp. 1717–1730, 2019.

[20] Y. Lin, Z. Jiang, J. Gu, W. Li, S. Dhar, H. Ren, B. Khailany, and D. Z. Pan, "DREAMPlace: Deep learning toolkit-enabled GPU acceleration for modern VLSI placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 4, pp. 748–761, 2021.

[21] W. Li, Y. Lin, and D. Z. Pan, "elfplace: Electrostatics-based placement for large-scale heterogeneous FPGAs," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, 2019.

[22] P. Spindler and F. M. Johannes, "Fast and accurate routing demand estimation for efficient routability-driven placement," in *2007 Design, Automation Test in Europe Conference Exhibition*, pp. 1–6, 2007.

[23] AMD, "Virtex UltraScale+ FPGAs," 2025. https://www.amd.com/en/products/adaptive-socs-and-fpgas/fpga/virtex-ultrascale-plus.html.

[24] 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)*, pp. 1–6, 2023.

[25] J. Mai, J. Wang, Z. Di, and Y. Lin, "Multielectrostatic FPGA placement considering SLICEL–SLICEM heterogeneity, clock feasibility, and timing optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 2, pp. 641–653, 2024.

[26] H. W. Kuhn, *The Hungarian Method for the Assignment Problem*, pp. 29–47. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[27] S2C, "S2C prototyping: FPGA ASIC SoC IP verification, validation, emulation," 2024. https://www.s2cinc.com.

[28] "ISPD16 contest." http://www.ispd.cc/contests/16.

[29] "ISPD17 contest." http://www.ispd.cc/contests/17.