Synergistic Die-Level Router for Multi-FPGA System with Time-Division Multiplexing Optimization

Jiarui Wang[1,2], Yanjing Liu[2,3], Yibo Lin[2,4,5*]

[1]School of Computer Science, Peking University, Beijing, China [2]School of Integrated Circuits, Peking University, Bejing, China

[3]School of Software & Microelectronics, Peking University, Bejing, China [4]Institute of EDA, Peking University, Wuxi, China

[5]Beijing Advanced Innovation Center for Integrated Circuits, Beijing, China

jiaruiwang@pku.edu.cn, liuyanjing@stu.pku.edu.cn, yibolin@pku.edu.cn

***Abstract*—Modern multi-FPGA systems featuring multi-die devices connected through time-division multiplexing (TDM) techniques have become increasingly common as the scale of designs increases rapidly. FPGA designs are meticulously partitioned at die-level for prototyping in modern emulation systems. Conventional FPGA-level routers often result in a large critical connection delay that impacts the whole design's frequency. Additionally, the excessive use of super long lines (SLLs) between neighboring dies leads to substantial routing congestion, causing the failure of the routing progress. To tackle these issues, we propose an effective and efficient die-level router for multi-FPGA systems, optimizing routing topology and the TDM ratio. Experimental results on the benchmarks from the die-level routing contest 2023 demonstrate 7.6% better critical connection delay with a 5.761× speed-up compared to the state-of-the-art router.**

I. Introduction

Multi-FPGA systems with multiple dies within each FPGA device are widely applied to implement large-scale circuit designs as the scale of designs increases rapidly. As shown in Fig. 1(a), a single multi-die FPGA device consists of several dies, each interconnected by super long lines (SLLs) [1, 2]. To address the challenge of limited I/O pins in the FPGA devices, the time-division multiplexing (TDM) technique [3] has been widely applied to enable efficient communication between different FPGAs. Fig. 1(b) and (c) illustrate the TDM I/O structure driven by the TDM clock, whose frequency is much higher than that of the system clock. Each physical TDM wire is assigned a TDM ratio, which specifies the number of nets it transmits within one system clock period.

There are two primary approaches to deploying a design to multi-FPGA systems: FPGA-level and die-level flows. FPGA-level flows partition a design into different FPGAs and let FPGA CAD tools like Vivado [4] finish the physical implementation of each FPGA. In contrast, die-level flows perform more fine-grained partitioning, dividing the design into dies (a.k.a SLRs in Xilinx FPGA) of FPGAs and only letting FPGA CAD tools finish the physical implementation of each die. FPGA-level flow can quickly generate the partitioning and FPGA-level routing topology, but it can lead to congestion during physical implementation for each FPGA device as it cannot precisely estimate the congestion level between and inside each die. Thus, fine-grained die-level flow is commonly used in many emulation systems like Synopsys ZeBu [5] and S2C OmniArk [6] to precisely estimate the design performance and control congestion at the early stage of the design process.

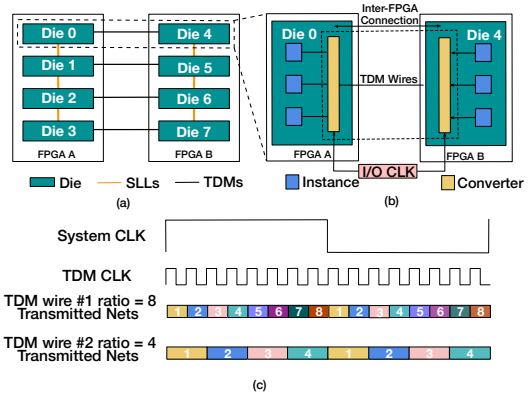


Fig. 1: (a) A multi-FPGA system with 2 FPGAs and 8 dies, with each die-to-die edge having several physical wires, for example, 100 wires. (b) The TDM edge between two dies on different FPGAs. (c) The waveforms of the system clock and the TDM clock.

System routing is an essential step in the multi-FPGA system design flow to generate inter-FPGA and inter-die routing topologies and TDM ratios in FPGA-level flow and die-level flow, respectively. Compared to FPGA-level routing, die-level routing provides more precise control and requires a more synergistic routing algorithm. A die-level router must simultaneously consider the different timing behaviors of SLL and TDM edges. It shall also synergistically avoid violating complex design rules or overlapping on SLL edges, as well as increasing design frequency by minimizing the maximum delay of all net connections (critical connection delay).

Prior academic works on multi-FPGA system routing mainly focus on FPGA-level [7] while ignoring die-level routing. They usually consist of two steps: initial routing to generate the routing topology, and TDM ratio assignment to determine TDM ratios. [8] and [9] propose SMT-based initial routing algorithms by finding minimum Steiner trees while dynamically adjusting routing costs. [10] applies maze routing [11, 12] and spanning tree algorithm [13] for nets with different criticality. However, as routing resources within different dies differ, FPGA-level initial routing cannot precisely estimate the design's frequency and can lead to insufficient SLL resources when the design's scale increases. To do the

*Corresponding author

TDM ratio assignment, many academic FPGA-level routers apply criticality-based algorithms [8, 10, 14] or Lagrangian-relaxation-based algorithms [15, 16]. Besides those works, [17] also considers the TDM capacity and direction rule during the TDM ratio assignment process. However, they ignore the different timing behavior between SLL and TDM wires, leading to an imprecise timing estimation.

Recently, [18] considers die-level routing by combining the minimum Steiner tree algorithm and maze routing algorithm for the initial routing and applying dynamic programming to assign the TDM ratio for each physical TDM wire. Their initial routing algorithm focuses on minimizing the total usage of SLL and TDM edges, which can lead to a large delay on critical connections. Their dynamic programming does not scale with design sizes either.

In this paper, we propose a synergistic die-level router for multi-FPGA systems to minimize the critical connection delay. The major contributions of this paper are summarized as follows.

- We propose a synergistic die-level router for multi-FPGA systems, optimizing both routing topologies and TDM ratios at die-level.
- We propose a balanced initial routing algorithm that synergistically optimizes connection delay and demands on SLL and TDM edges.
- We propose a multi-threaded Lagrangian-relaxation-based initial TDM ratio assignment algorithm with a margin-aware TDM ratio legalization and TDM wire assignment algorithm to assign TDM ratios for nets and physical TDM wires synergistically.

Compared to the state-of-the-art (SOTA) die-level multi-FPGA system routers, we achieve a 7.6% less critical connection delay with 5.761× speed-up on die-level multi-FPGA system routing contest 2023 [19] benchmarks (a problem of the 2023 Integrated Circuit EDA Elite Challenge organized by an industrial vendor [20]).

The rest of this paper is organized as follows. Section II describes a die-level multi-FPGA system and the problem formulation of the die-level multi-FPGA system routing problem. Section III demonstrates the algorithm flow of our router. Section IV validates our routing algorithm with experimental results. Section V concludes the paper.

II. Preliminaries

In this section, we introduce the background of the die-level multi-FPGA system routing problem and demonstrate our design rule constraints. The symbols and their descriptions used in this paper are listed in TABLE I.

A. Background

Modern FPGA architectures [1] usually have multiple dies. As shown in Fig. 1(a), FPGA A and FPGA B are two FPGAs, each with 4 dies. Neighboring dies are connected by SLL edges, and each SLL edge consists of several physical SLL wires. We define the number of physical SLL wires as the capacity of an SLL edge e, denoted as cap_e, which is the maximum number of nets it can route. The capacity of each SLL edge can be up to 10^4.

Multi-FPGA systems have been widely applied for large designs in recent years. As shown in Fig. 1(a), FPGA A and FPGA B form a multi-FPGA system consisting of two FPGAs. TDM edges $\mathbb{E}_{\mathrm{TDM}}$ connect different dies across

TABLE I: List of symbols.

Symbol	Description
$\mathbb{V}$	The set of FPGA dies.
$\mathbb{E}_{\mathrm{TDM}}, \mathbb{E}_{\mathrm{SLL}}$	The set of TDM edges and SLL edges.
$\mathbb{N}$	All the nets in the netlist.
$\mathbb{N}_e$	All the nets using edge e.
$\mathbb{C}$	All the connections.
S_{ce}	Binary variable representing whether edge e is used to route connection c.
n_c	The net of the connection c.
r_{ne}	The TDM ratio of net n on TDM edge e.
cap_e	The number of physical wires edge e contains.
demand_e	The number of nets using e.
d	the critical connection delay.
d_c	The delay of the connection c.
d_{SLL_c}	The delay of connection c contributed by the SLL edges.
d_{SLL}	The delay of the SLL edges.
p	TDM step.

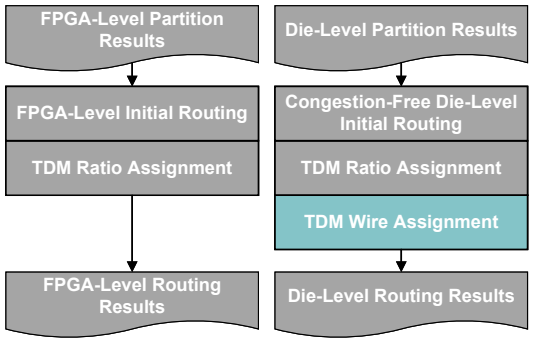


Fig. 2: The Difference between FPGA-level routing and die-level routing. (a) A typical flow of ICCAD 2019 contest [7] FPGA-level routers. (b) A typical flow of die-level routers.

different FPGAs. Each TDM edge e_{TDM} is formed by several physical TDM wires, and each physical TDM wire has a TDM ratio. The capacity of a TDM edge e is the number of physical wires it has, which is also denoted as cap_e. All the physical TDM wires within the same TDM edge share the same TDM clock but may have different TDM ratios. As shown in Fig. 1(c), the two physical wires have TDM ratios of 8 and 4, respectively.

As shown in Fig. 2, die-level routing is completely different from FPGA-level routing. Conventional FPGA-level multi-FPGA system routers (e.g. ICCAD 2019 contest routers [7]) take FPGA-level partitioning results as their inputs and assign TDM ratio between each FPGA-to-FPGA pair, which is shown in Fig. 2(a). As illustrated in Fig. 2(b), die-level routers take die-level partitioning results as inputs and output an overlapping-free routing solution with minimum critical connection delay at die-level. They shall also do TDM wire assignment to assign the TDM ratio on each physical TDM wire and assign each cross-FPGA net to physical TDM wires on the die-to-die TDM edges it passes, which is not considered by most FPGA-level routers.

B. Design rules

Unlike conventional FPGA-level routing, die-level routing deals with much more complex design rules, which are listed below.

Connectivity rule. Each connection of a net shall be routed by SLL edges or TDM edges without loops.

SLL capacity and delay rule. Each physical SLL wire can only be used to route at most one net. Therefore, the

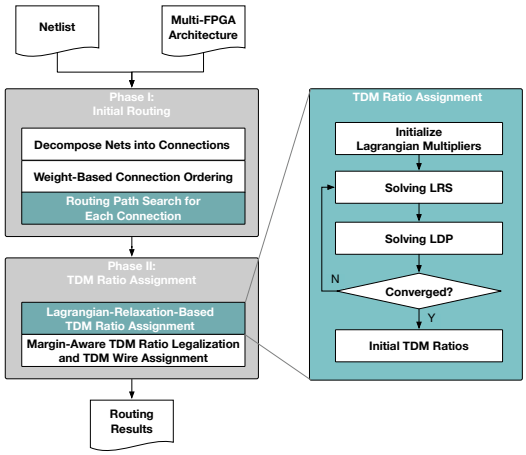


Fig. 3: The overall algorithm flow of our die-level router.

number of nets using an SLL edge e shall never exceed cap_e. The delay of all the SLL edges $\mathbb{E}_{\text{SLL}}$ is a constant d_{SLL}.

TDM wire ratio and delay rule. A physical TDM wire belonging to TDM edge e can route any number of nets. The number of nets routed by a wire is defined as its demand. The TDM ratio r of a TDM wire shall be not less than its demand, and it shall also be multiple of the TDM ratio step p. All the TDM ratios of nets using the same physical TDM wire with TDM ratio r are the same as r, and the delay of a TDM wire is defined as $d_0 + d_1 r$, where d_0 and d_1 are constants.

TDM capacity rule and TDM direction rule. For a TDM edge e, the demand for physical TDM wires shall never exceed cap_e. Moreover, although TDM edges are bi-directional, a physical TDM wire can only be used to pass signals in the same direction.

C. Die-level routing problem for multi-FPGA system

We formally define our die-level multi-FPGA system routing problem as follows.

Problem. *Given a die-level multi-FPGA system and netlist $\mathbb{N}$, find the routing paths for each net with the minimum critical connection delay following the design rules listed in Section II-B:*

$$\min \max_{c \in \mathbb{C}} \sum_{e \in \mathbb{E}_{\text{TDM}}} S_{ce}(d_0 + d_1 r_{n_c e}) + \sum_{e \in \mathbb{E}_{\text{SLL}}} S_{ce} d_{\text{SLL}}. \quad (1)$$

III. Algorithm

In this section, we introduce the algorithm of our die-level router for multi-FPGA systems.

A. Overall flow of our algorithm

The overall flow of our die-level router is shown in Fig. 3. Our router takes the netlist and the multi-FPGA system as input and outputs die-level routing paths for each net and TDM ratios for each net and TDM wire. Our router synergistically optimizes the usage of SLL edges and TDM edges by two phases: (I) initial routing to generate routing paths for each die-crossing net, and (II) TDM ratio assignment to assign TDM ratios on TDM wires to minimize the critical connection delay.

Our router first generates die-level routing paths for each net at the initial routing phase (Section III-B). We decompose each net into die-to-die connections and find a routing result while balancing the connection delay and demand on SLL and TDM edges.

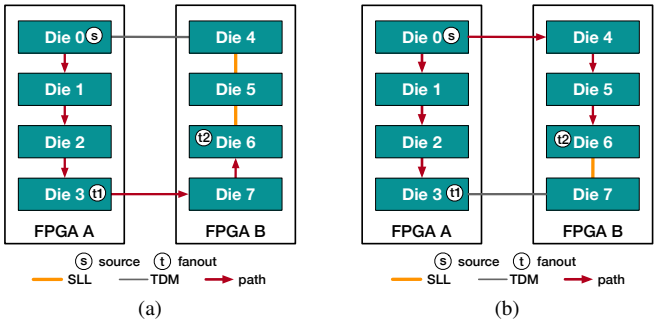


Fig. 4: (a) The minimum Steiner tree for the net connecting die 0 to die 3 and die 6 with the larger connection delay to die 5. (b) The shortest path tree for the same net with a smaller connection delay but uses more edges to route.

To further optimize the critical connection delay, our router assigns TDM ratios on each TDM edge based on our initial routing solution. We first use the Lagrangian-relaxation-based method to assign the initial TDM ratio (Section III-C). Then, we legalize the TDM ratio and assign the TDM ratio for each physical TDM wire (Section III-D).

B. Delay-demand-balanced initial routing

Our initial routing phase aims to find a routing path for each net without congestion. Some prior works [8] apply the minimum Steiner tree algorithm to get initial routing results. Those algorithms aim to minimize the total routing cost of the input design. However, as shown in Fig. 4(a), such algorithms can lead to a large connection delay when routing multi-fanout nets. An alternate idea is to find each net a shortest-path tree to minimize the connection delay. Such an idea can decrease the delay for each connection, but as shown in Fig. 4(b), it increases the usage of routing edges and can cause routing congestion on the SLL edges or lead to large TDM ratios on TDM edges. Thus, our initial routing algorithms synergistically balance the delay of each connection and the demand of each SLL and TDM edge.

To find delay-demand-balanced initial routing results for each net, our initial router decomposes each net into connections and finds each connection a routing path. The routing order of those connections is decided by sorting the routing weight by the Floyd-Warshall [21] algorithm by the decreasing order. Using the cost function listed in the following paragraphs, we then find a routing path for each connection following the negotiation-based path-finding algorithms [22]. The routing cost for each SLL and TDM edge is impacted by both edge delay and demand of each edge to synergistically consider connection delay and edge demand. As there may be congestion in the routing result since there are capacity constraints on those SLL edges, we increase the cost of those congested SLL edges and reroute those congested nets iteratively to find an overlapping-free routing result for each net.

Weight estimation and connection ordering. We assign a routing weight to each edge to calculate the routing weight of each pair of dies. Based on the number of nets in the design, SLL and TDM edges have different weights. When the number of nets on each die is less than half of the capacity of SLL edges, TDM edges are assigned a higher

weight of $\|\mathbb{V}\| + 1$, while SLL edges are given a lower weight of 1. Such a weight assignment encourages the router to explore more SLL and TDM edges for less delay. Conversely, if the number of dies exceeds this threshold, SLL edges receive a $\|\mathbb{V}\| + 1$ weight, and TDM edges are weighted at 1, which helps the router avoid congestion on SLL edges.

The routing weight of each connection is the sum of the edge weight of the shortest path between two dies, which is collected by the Floyd-Warshall algorithm [21]. Connections with higher routing weights are routed first, and connections belonging to the net with fewer fanouts have higher priority when the routing weight is the same.

Calculating routing cost. Our router iteratively applies the negotiation-based path-finding algorithms [22] to find a routing path for each connection, and we need to precisely estimate the routing cost of each SLL edge and each TDM edge to balance usage of routing edges and delay of connections. SLL and TDM edges have different routing cost functions as their timing behaviors differ. To reduce usage of the routing edges, when routing a connection of a net n, the cost of a routing edge is also impacted by whether the edge is used to route other connections of n. For SLL edges, we define their routing costs as μw_e, where w_e is the estimated edge weight, and μ is impacted by whether the SLL edge has been used to route the net the connection belongs to, which will be discussed in the following paragraph. For TDM edges, their routing costs are defined as follows:

$$\text{cost}_e = \mu(d_0 + p + \frac{\text{demand}_e}{\text{cap}_e}). \tag{2}$$

The routing cost of TDM edges increases as their demand increases, which can balance the usage of TDM edges to minimize the critical connection delay.

The parameter μ is used to balance the total usage of routing edges and the delay of each connection. When we explore edge e when routing a connection c of a net n, μ is impacted by whether e has already been used to route n. If e is not used to route n, μ is set as 1. If e is used to route other connections of n, then μ is set as a real number between 0 and 1, and we set it as $\frac{1}{2}$ in practice.

C. Lagrangian-relaxation-based initial TDM ratio assignment

It is difficult to directly assign the TDM ratio of each TDM wire and assign nets to them. Therefore, we relax the TDM wire ratio and delay rule, and assign each net a TDM ratio as a real number on each TDM edge it passes. Those initially assigned TDM ratios are legalized by the algorithms described in the next section.

The formulation of our TDM ratio assignment problem is listed as follows:

$$\begin{aligned} \min_{d, r_{ne}} \quad & d \\ s.t. \quad & \sum_{e \in \mathbb{E}_{\text{TDM}}} S_{ce} d_{ce} + d_{\text{SLL}_c} \le d, \quad \forall c \in \mathbb{C}, \\ & \sum_{n \in \mathbb{N}_e} \frac{1}{r_{ne}} \le \text{cap}_e - 1, \qquad \forall e \in \mathbb{E}_{\text{TDM}}, \end{aligned} \tag{3}$$

where

$$d_{ce} = d_0 + d_1 r_{n_c e}. \tag{4}$$

Note that to avoid a physical TDM wire being used bidirectionally, we restrict usage of each TDM edge e to no more than $\text{cap}_e - 1$, not cap_e. The usage of TDM edges is fulfilled in our TDM ratio legalization phase.

As the primal problem (PP) is difficult to solve, we apply Lagrangian relaxation (LR) to PP and get the Lagrangian relaxation subproblem (LRS) by adding Lagrangian multipliers (LMs) λ to the first set of constraints in (3), such a method is widely used in many FPGA-level TDM ratio assignment algorithms [15, 16]. LRS is much easier to solve than PP, and its optimal solution of LRS is a lower bound on PP. We list the formulation of LRS as follows:

$$\begin{aligned} \min_{d, r} \quad & L_\lambda(d, r) \\ s.t. \quad & \sum_{n \in \mathbb{N}_e} \frac{1}{r_{ne}} \le \text{cap}_e - 1, \quad \forall e \in \mathbb{E}_{\text{TDM}}. \end{aligned} \tag{5}$$

where

$$L_\lambda(d, r) = d + \sum_{c \in \mathbb{C}} \lambda_c \Big(\sum_{e \in \mathbb{E}_{\text{TDM}}} S_{ce}(d_0 + d_1 r_{n_c e}) + d_{\text{SLL}_c} - d \Big). \tag{6}$$

The optimal solution of LRS is defined as $LRS^*(\lambda)$. The maximization of $LRS^*(\lambda)$ provides the maximum lower bound of PP. We aim to maximize $LRS^*(\lambda)$ in our TDM ratio assignment phase, which is known as solving the Lagrangian dual problem (LDP). The formulation of LDP is defined as follows:

$$\begin{aligned} \max_{\lambda} \quad & LRS^*(\lambda) \\ s.t. \quad & \lambda_c \ge 0, \qquad \forall c \in \mathbb{C}. \end{aligned} \tag{7}$$

We list how we assign the initial TDM ratio and solve LDP in Algorithm 1. We init λ_c as $\frac{1}{\|\mathbb{C}\|}$ (line 1) at the beginning and iteratively update λ to maximize $LRS^*(\lambda)$ until the difference between $LRS^*(\lambda)$ and d is less than the given threshold ϵ or the number of iterations reaches the maximum iteration number (lines 2-7). In each iteration, we first solve LRS to assign TDM ratios for each net on each TDM edge it passes (line 4). Then, we calculate $LRS^*(\lambda)$ and d and update the LMs λ (lines 5-6).

Algorithm 1: Initial TDM Ratio Assignment

Input: Initial routing result, Maximum iterations $MaxIter$, Threshold ϵ
Output: Initial TDM ratios r_{ne}

1 Initialize Lagrangian Multipliers λ.
2 $Iter \leftarrow 0$.
3 **while** $\frac{d - LRS^*(\lambda)}{LRS^*(\lambda)} \ge \epsilon$ ***And*** $Iter < MaxIter$ **do**
4 Calculate r_{ne} by solving LRS.
5 Calculate $LRS^*(\lambda)$ and d.
6 Update the Lagrangian Multipliers λ.
7 $Iter \leftarrow Iter + 1$.

Finding the optimal solution of LRS. To find the optimal solution of LRS, we first apply the Karush-Kuhn-Tucker (KKT) condition:

$$\frac{\partial L_\lambda}{\partial d} = 1 - \sum_{c \in \mathbb{C}} \lambda_c = 0. \tag{8}$$

Therefore, we only need to solve the following optimization problem for each TDM edge e to minimize $L_\lambda(d, r)$:

$$\min_{r} \sum_{n \in \mathbb{N}_e} \eta_{ne} r_{ne} + \sum_{c \in \mathbb{C}} \lambda_c (d_{\text{SLL}_c} + \sum_{e \in \mathbb{E}_{\text{TDM}}} d_0 S_{ce}), \tag{9}$$

where

$$\eta_{ne} = d_1 \sum_{c \in \mathbb{C}_n} \lambda_c S_{ce}. \quad (10)$$

By applying Cauchy-Schwarz Inequality, we have:

$$(\sum_{n \in \mathbb{N}_e} \sqrt{\eta_{ne} r_{ne}}^2)(\sum_{n \in \mathbb{N}_e} \frac{1}{\sqrt{r_{ne}}^2}) \geq (\sum_{n \in \mathbb{N}_e} \sqrt{\eta_{ne}})^2. \quad (11)$$

To fully use the capacity of each TDM edge, the optimal solution of LRS must follow that $\sum_{n \in \mathbb{N}_e} \frac{1}{r_{ne}} = \text{cap}(e) - 1$. Therefore, the equality in (11) holds if and only if

$$r_{ne} = \frac{\sum_{n' \in \mathbb{N}_e} \sqrt{\eta_{n'}}}{\sqrt{\eta_n}(\text{cap}_e - 1)}. \quad (12)$$

Afterwards, we obtain the optimal solution r_{ne} for LRS.

Iteratively update solution of LDP. The target of LDP is to find λ that maximize $LRS^*(\lambda)$. Based on the fact that PP is a convex problem, which has only one optimal solution, we apply methods in [15] to update λ in each iteration by the following equation to obtain the solution of LDP quickly:

$$\lambda_c'^{i+1} = \lambda_c^i (\frac{d_c}{d})^{K_c^i}. \quad (13)$$

K_c^i is the acceleration factor, and it is updated every iteration by the method in [15]. After that, we normalize $\lambda_c'^{i+1}$ to satisify the KKT condition listed in (8).

Multi-threaded acceleration. Solving LDP usually takes many iterations, which is very time-consuming when the netlist is very large. We apply a multi-threaded technique for large-scale netlists to efficiently obtain initial TDM ratio assignment results and accelerate the process without losing quality. At the beginning of each iteration, we first calculate η_{ne} and use them to calculate r_{ne}. This progress is individual for each TDM edge and we can assign each TDM edge an independent thread to accelerate the process. Next, we need to calculate the results of LRS and LDP. To calculate them, we need to obtain the delay of each connection, which can be calculated parallelly for each connection. We assign each connection a thread to calculate its delay. To avoid the read-write conflict, we use the reduction operation [23] to parallelly calculate $LRS^*(\lambda)$ and d. We then need to update LMs λ. We first parallelly calculate $\lambda_c'^{i+1}$ and use the reduction operation to get the sum of $\lambda_c'^{i+1}$. Finally, we update λ_c for each connection individually.

D. TDM ratio legalization and margin-aware TDM wire assignment

Our initial TDM ratio assignment process assigns each net a TDM ratio on each TDM edge it passes. We then need to legalize those TDM ratios to multiples of TDM step p. The legalized TDM ratios can be used for TDM wire assignment to decide the TDM ratio of each TDM wire and assign nets to TDM wires. Such a process increases the TDM ratio of each net, leaving a margin between demand and capacity on each TDM edge. To further optimize the critical connection delay, we propose a TDM ratio refinement algorithm to resolve those margins. Our legalization and assignment process is individual for each TDM edge, and we parallelly process each edge on large designs for less runtime.

Since a physical TDM wire can only pass connections in one direction, we first assign the number of physical wires for two directions of a TDM edge. For each bi-directional TDM edge e, we first calculate the sum of $\frac{1}{r_{ne}}$ for two directions and round them up to the nearest integer as the number of physical TDM wires for each direction, which are nominated as $\text{cap}_{\overrightarrow{e}}$ and $\text{cap}_{\overleftarrow{e}}$ for two directions. Note that we restrict the sum of $\frac{1}{r_{ne}}$ less than $\text{cap}_e - 1$ in the initial assignment stage. Therefore, the sum of physical wires assigned to two directions is exact cap_e.

Algorithm 2: TDM Ratio refinement

Input: TDM Ratios $r_{n\overrightarrow{e}}$ on the directed TDM edge $\overrightarrow{e}$

Output: Refine TDM Ratios $r_{n\overrightarrow{e}}$

1 Initialize an empty priority queue q.
2 Push all the nets using $\overrightarrow{e}$ and their criticality in to q.
3 **while** $\text{cap}_{\overrightarrow{e}} - \sum \frac{1}{r_{n\overrightarrow{e}}} \geq \epsilon$ **do**
4 Pop out the most critical net n from q.
5 Decrease $r_{n\overrightarrow{e}}$ and the criticality of n.
6 Add n back to q.

To legalize TDM ratios on a TDM edge e, we first increase all the r_{ne} to the nearest multiple of TDM step p. After that, there will be a margin between demand and capacity on each TDM edge. We list how we refine TDM ratios on a directed TDM edge $\overrightarrow{e}$ in Algorithm 2. We define the criticality of a net on a TDM edge as the largest connection delay of the connections of the net passing the TDM edge. We use a priority queue to manage all the nets passing the TDM edge (line 2) and pop out the most critical net repeatedly until the margin is less than a certain threshold (lines 3-6). If the TDM ratio for the most critical net on the TDM edge is larger than TDM step p, we decrease its ratio by p and its criticality by $d_1 p$ (line 5), and push the net back into the priority queue (line 6).

After the TDM ratios are refined, we use a greedy-based method to assign the TDM ratio of physical TDM wires and assign nets to them. For each TDM wire, we assign the smallest r_{ne} of the remaining nets as its TDM ratio and then assign nets from the same direction with the top r_{ne}-smallest to use the TDM wire. After that, there will be remaining demands to assign or remaining wires not assigned a ratio. We then increase the TDM ratio of the wires with small criticality nets to assign the remaining demands or assign nets with large criticality to empty wires to resolve the remaining capacity. Finally, each net will be assigned to a TDM wire, and the TDM ratio of the wire is assigned to nets using it as their ratio on the TDM edge.

IV. Experimental Results

We implement our algorithm in C++ and use OpenMP [24] for multi-threading. Our experiments are conducted on a Linux machine with an Intel Xeon 4210-R 10-Core Processor (2.40 GHz) and 320 GB RAM. For designs with more than 200,000 nets, we use 10 threads to run phase II of our router. We use a single thread to avoid the time of multi-thread scheduling for other designs.

TABLE II: The statistics of the contest [19] benchmarks.

Design	#FPGAs	#Dies	SLL		TDM		#Nets	#Conns
			#Edges	#Wires	#Edges	#Wires		
Case #1	2	8	6	122K	2	400	5	5
Case #2	2	8	6	122K	2	400	86	155
Case #3	2	8	6	122K	2	20	84	154
Case #4	2	8	6	122K	2	40	449	577
Case #5	3	12	9	183K	3	440	5K	5K
Case #6	3	12	9	183K	14	10K	145K	281K
Case #7	4	16	12	244K	15	9K	76K	118K
Case #8	4	16	12	244K	15	7K	86K	146K
Case #9	4	16	12	244K	21	142K	871K	183K
Case #10	5	20	15	305K	19	75K	3324K	7279K

TABLE III: Critical connection delay (Delay), the number of overlapping on SLL edges (#CONF), and routing runtime (Runtime, s) on the die-level routing contest benchmarks [19] between the contest winners, [18], adapted [9] and our router.

Router \ Design		Case #1	Case #2	Case #3	Case #4	Case #5	Case #6	Case #7	Case #8	Case #9	Case #10	Norm.
1st	Delay	**6.5**	**7.5**	**11.5**	19.5	135.5	213	84.5	124	150	4657.5	1.098
	#CONF	0	0	0	0	0	0	0	0	0	0	-
	Runtime	0.01	0.01	0.01	0.03	0.17	6.73	2.43	3.11	50.81	356.70	1.557
2nd	Delay	**6.5**	**7.5**	14.5	19.5	136	260	102.5	145.5	196.5	4745	1.238
	#CONF	0	0	0	0	0	0	0	0	0	0	-
	Runtime	0.01	0.01	0.01	0.02	0.10	3.70	1.49	1.90	612.19	784.55	4.468
3rd	Delay	**6.5**	**7.5**	**11.5**	**18.5**	132.5	287	76	123	177	5700	1.171
	#CONF	0	0	0	0	0	0	0	0	0	0	-
	Runtime	0.01	0.01	0.01	0.03	0.87	263.14	85.56	303.77	2274.93	2115.07	34.362
[18][1]	Delay	**6.5**	**7.5**	**11.5**	**18.5**	**131**	211.5	78.5	113.5	154.5	4739.5	1.076
	#CONF	0	0	0	0	0	0	0	0	0	0	-
	Runtime	0.01	0.01	0.01	0.02	0.30	18.76	6.04	5.94	363.28	2169.83	5.761
Adapted [9]	Delay	**6.5**	12	28	32	185	FAIL[2]	270	505	FAIL	FAIL	2.353
	#CONF	0	0	0	0	0	11052	0	0	676430	4920599	-
	Runtime	0.08	0.06	0.06	0.07	0.14	2.96	1.34	1.59	22.69	106.38	2.323
Ours	Delay	**6.5**	**7.5**	**11.5**	**18.5**	136	**163**	**74**	**107.5**	**122.5**	**4207.5**	1.000
	#CONF	0	0	0	0	0	0	0	0	0	0	-
	Runtime	0.01	0.02	0.02	0.02	0.10	5.08	2.09	2.88	21.01	84.50	1.000

[1] We obtained the latest binary from the authors of [18]. The results on our machine are slightly different from the original paper, which have been verified by the authors [18].

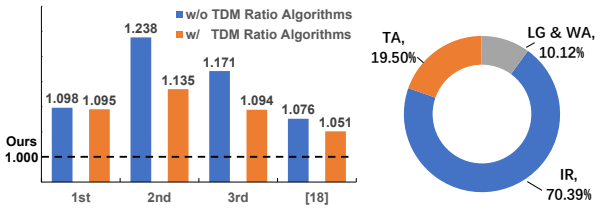
[2] 'FAIL' means the routing result is illegal as there are overlapping on the SLL edges.

We collect 10 cases from the die-level routing contest 2023 [19] provided by an industrial vendor [20], and the cases are available at [25]. The statistics of the contest benchmarks are listed in TABLE II. The number of FPGAs in the multi-FPGA system varies from 2 to 5, and the number of dies in the multi-FPGA system ranges from 8 to 20. The number of nets and connections can be up to 10^6.

We collect the binary files of the top-3 winners from the die-level routing contest [19] and the latest binary file of [18] and run their binary files on our machine by 10 threads. We list the results of those four routers and our routers in TABLE III. Compared to the contest winners, our router has a 9.8%, 23.8%, and 17.1% less critical connection delay with a 1.557×, 4.468×, and 34.362× speedup. Also, our router has a 7.6% less critical connection delay and a 5.761× speedup compared with [18].

We also collect the binary file of [9], which is the SOTA FPGA-level multi-FPGA routing to the best of our knowledge. We run their binary files by faking each die as an FPGA, with each edge as a fake FPGA-to-FPGA connection while regarding each net as a net group. We use our TDM ratio legalization and TDM wire assignment algorithm to assign the TDM ratio of their result. The adapted result is also listed in TABLE III. The adapted router fails to deal with 3 of the 10 cases and has much more critical connection delay in other cases. Note that comparing the results of adapted [9] with ours is not fair since the die-level routing problem is entirely new compared to the FPGA-level routing problem from ICCAD 2019 contest [7]. The main purpose of this experiment is to show that these two problems are quite different, and thus, simple adaption is not effective.

To validate the effectiveness of our initial routing algorithm and our TDM ratio algorithms (containing initial TDM ratio assignment, TDM ratio legalization, and TDM wire assignment), we collect the routing topology generated by the router of the contest [19] winners and [18] and use our TDM ratio algorithms to refine the results. The results of the ratio of the critical connection delay are listed in Fig. 5(a). As Fig. 5(a) shows, the results assigned by our TDM ratio algorithms can refine the critical connection delay of the routing results of the four baselines by 0.3%, 10.3%, 7.7%, and 2.5%, which shows the effectiveness of our TDM ratio algorithms. The critical connection delay of the refined



(a) (b)

Fig. 5: (a) The ratio of critical connection comparison between the results of the contest [19] winners and [18] without our TDM algorithms, the results of the contest winners and [18] refined by our TDM ratio algorithms, and the results of our router. (b) Runtime breakdown of the routing algorithms on design Case #10.

results are 9.5%, 13.5%, 9.4%, and 5.1% larger than the results of our router, demonstrating the effectiveness of our initial routing algorithm.

The runtime breakdown of our routing algorithms, including initial routing (IR), initial TDM ratio assignment (TA), TDM ratio legalization and TDM wire assignment (LG & WA), is listed in Fig. 5(b). The initial routing takes 70.39% of the total routing algorithm runtime, as we need to route each connection sequentially. Assigning the TDM ratio by Lagrangian relaxation takes 19.50% of the runtime while legalizing the TDM ratio and assigning the TDM ratio to each wire takes 10.12% of the runtime, as they are multi-threaded accelerated.

V. Conclusion

In this paper, we propose a synergistic die-level multi-FPGA system router to generate die-level routing results for multi-FPGA systems. We propose a connection-based path-searching algorithm to balance connection delay and edge demand synergistically. We propose a Lagrangian-relaxation-based initial TDM ratio assignment algorithm to effectively reduce the critical connection delay with a margin-aware TDM ratio legalization and TDM wire assignment method. Compared to the SOTA, we can achieve a 7.6% less critical connection delay with a 5.761× speed-up.

Acknowledge

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

REFERENCES

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

[2] 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)*, 2018, pp. 16–164.

[3] 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.

[4] AMD, "Vivado overview," 2024, https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vivado.html.

[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] Y.-H. Su, R. Sun, and P.-H. Ho, "2019 CAD contest: System-level FPGA routing with timing division multiplexing technique," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–2.

[8] P. Zou, Z. Lin, X. Shi, Y. Wu, J. Chen, J. Yu, and Y.-W. Chang, "Time-division multiplexing based system-level FPGA routing for logic verification," in *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference*, ser. DAC '20. IEEE Press, 2020.

[9] W.-K. Liu, M.-H. Chen, C.-M. Chang, C.-C. Chang, and Y.-W. Chang, "Time-division multiplexing based system-level FPGA routing," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–6.

[10] D. Zheng, X. Zhang, C.-W. Pui, and E. F. Young, "Multi-FPGA co-optimization: Hybrid routing and competitive-based time division multiplexing assignment," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, ser. ASPDAC '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 176–182. [Online]. Available: https://doi.org/10.1145/3394885.3431565

[11] C. Y. Lee, "An algorithm for path connections and its applications," *IRE Transactions on Electronic Computers*, vol. EC-10, no. 3, pp. 346–365, 1961.

[12] H. Li, G. Chen, B. Jiang, J. Chen, and E. F. Y. Young, "Dr. cu 2.0: A scalable detailed routing framework with correct-by-construction design rule satisfaction," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–7.

[13] K. Mehlhorn, "A faster approximation algorithm for the steiner problem in graphs," *Inf. Process. Lett.*, vol. 27, no. 3, p. 125–128, mar 1988. [Online]. Available: https://doi.org/10.1016/0020-0190(88)90066-X

[14] Z. Zhuang, X. Huang, G. Liu, W. Guo, W. Qian, and W.-H. Liu, "ALIFRouter: A practical architecture-level inter-FPGA router for logic verification," in *Proc. DATE*, 2021, pp. 1570–1573.

[15] T.-W. Lin, W.-C. Tai, Y.-C. Lin, and I. H.-R. Jiang, "Routing topology and time-division multiplexing co-optimization for multi-FPGA systems," in *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference*, ser. DAC '20. IEEE Press, 2020.

[16] C.-W. Pui and E. F. Y. Young, "Lagrangian relaxation-based time-division multiplexing optimization for multi-FPGA systems," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 25, no. 2, feb 2020. [Online]. Available: https://doi.org/10.1145/3377551

[17] C.-W. Pui, G. Wu, F. Y. C. Mang, and E. F. Y. Young, "An analytical approach for time-division multiplexing optimization in multi-FPGA based systems," in *2019 ACM/IEEE International Workshop on System Level Interconnect Prediction (SLIP)*, 2019, pp. 1–8.

[18] 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)*, 2024, pp. 324–331.

[19] S2C, "FPGA die-level system routing algorithm design," 2023, https://edaicisc.oss-cn-shanghai.aliyuncs.com/file/cacheFile/2023/10/23/0e6d23494cd24c7c94816d5bd0acf89c.pdf.

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

[21] R. W. Floyd, "Algorithm 97: Shortest path," *Commun. ACM*, vol. 5, no. 6, p. 345, jun 1962. [Online]. Available: https://doi.org/10.1145/367766.368168

[22] L. McMurchie and C. Ebeling, "Pathfinder: A negotiation-based performance-driven router for FPGAs," in *Third International ACM Symposium on Field-Programmable Gate Arrays*, 1995, pp. 111–117.

[23] J. Ciesko, S. Mateo, X. Teruel, X. Martorell, E. Ayguadé, J. Labarta, A. Duran, B. R. de Supinski, S. Olivier, K. Li, and A. E. Eichenberger, "Towards task-parallel reductions in OpenMP," in *OpenMP: Heterogenous Execution and Data Movements*. Cham: Springer International Publishing, 2015, pp. 189–201.

[24] "OpenMP," http://www.openmp.org/.

[25] "2023 Integrated Circuit EDA Elite Challange contest problems," 2023, https://eda.icisc.cn/en/download/index?periodId=c87b85c3751c43ceb3ca482ee4cc313d&type=2.