Timing-Aware Optimization of Die-Level Routing and TDM Assignment for Multi-FPGA Systems

Yijun Chen[1], Haoyuan Li[2], Chunyan Pei[2], Jianwang Zhai[1], Kang Zhao[1], and Wenjian Yu[2]
[1]*Beijing University of Posts and Telecommunications, Beijing 100876, China*
[2]*Dept. Computer Science & Tech., BNRist, Tsinghua Univ., Beijing 100084, China*

***Abstract*—The escalating scale and complexity of modern circuits demand multi-FPGA emulation platforms that incorporate multi-die architectures. However, most existing routers remain FPGA-level, optimizing wire-length or total Time-Division Multiplexing (TDM) ratios while disregarding die-level load imbalance and path-level slack. They result in suboptimal performance and timing violations. In this paper, we propose a timing-aware co-optimization framework for die-level routing and TDM assignment, explicitly linking physical constraints to critical path timing slack. The proposed flow features a timing-aware load-balanced die-level router with timing path compression and a timing graph-based TDM assignment. Experiments on industrial designs show that the proposed method improves the worst-path slack by 98% over the existing methods.**

***Index Terms*—Timing path, Timing-aware, Die-level, System routing, TDM assignment.**

I. Introduction

Driven by the rapid escalation of circuit complexity, logic verification has become a dominant factor in VLSI development schedules and budgets [1]. Although FPGAs serve as the basic platform for hardware emulation, the most advanced 2.5-D device, exemplified by the Versal Premium VP1902 FPGA [2], remains insufficient to accommodate billion-gate designs. Vendors therefore construct multi-FPGA systems (MFS) by interconnecting multiple ultra-scale FPGAs, each internally partitioned into super logic regions (SLRs, or dies) linked via dedicated super long lines (SLLs) [3]–[6]. Despite their scalability and parallelism, such systems pose substantial challenges in interconnect routing and timing closure.

A key challenge lies in routing signals through a limited directional interconnect fabric without full pairwise connectivity. As illustrated in Fig. 1, the scarcity of inter-FPGA channels and fixed directionality necessitate relay path selection that balances timing performance and channel utilization, motivating a joint optimization of routing and I/O resource allocation.

To support signal transmission across multiple dies, MFS deploys physical cables or programmable networks to accommodate designs that exceed the capacity of individual FPGAs [7]. As a remedy to I/O bottlenecks, modern MFS prototypes overlay multiple logical signals onto each physical cable through Time-Division Multiplexing (TDM) [8], as illustrated in Fig. 1. While this technique improves resource utilization, the corresponding TDM-induced delay increases with the multiplexing ratio and significantly impacts system timing. At the die level, these delays compound with routing complexity, further reinforcing the need for joint optimization of routing and TDM assignment under strict timing constraints.

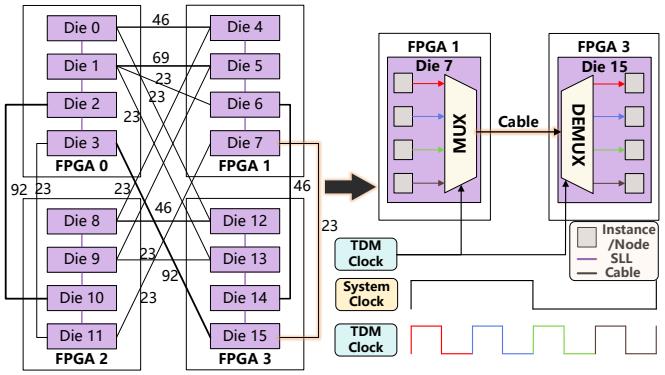

Fig. 1: Overview of topological and TDM architecture.

Existing studies on MFS primarily focus on FPGA-level [9]–[11] with limited exploration of die-level constraints. System-level routing and TDM assignment originate from the 2019 ICCAD Contest [11], followed by works employing Steiner tree approximations with timing-driven TDM assignment [12], profiling-guided assignment [13], Lagrangian relaxation [14], and hybrid maze routing with MTST [15]. However, these methods aim to minimize the total TDM ratio across nets, without optimizing the worst timing path slack or incorporating accurate timing models. Techniques that optimize ratios based on ILP [16], [17] or Lagrangian methods [18] similarly neglect connection directionality, die hierarchy, and realistic timing constraints, notably inter-die delays and clock domain effects. More recent die-level efforts [19], benchmarked on the 2023 EDA Elite Challenge [20], still adopt net-centric costs and simplified timing models.

In contrast, industrial verification prioritizes comprehensive timing closure, measured by the slack reported from static timing analyzers (e.g., *OpenSTA* [21]) rather than proxy metrics, namely TDM ratio or individual net delay [22]. Addressing this gap, our work targets post-partitioning industrial scenarios where sub-circuits are pre-assigned to specific dies. We optimize inter-die routing to guide critical paths through low-delay channels while mitigating congestion and balancing channel utilization for non-critical traffic. By directly embedding STA-reported slack into both routing and multiplexing, our approach enables first-pass timing closure without compromising resource efficiency. Our main contributions are as follows.

- We propose a timing-aware co-optimization framework that considers STA-reported timing paths, TDM-induced

Y. Chen and H. Li contributed equally. This work was done when Y. Chen visited Tsinghua Univ. This work was supported in part by National Key R&D Program of China (No. 2022YFB2901100) and Beijing Natural Science Foundation (No. Z230002). W. Yu is the corresponding author.

delay, inter-die latency, and clock-domain effects, aiming to maximize worst-path slack under topology, direction, capacity, and timing constraints.
- We propose a timing-aware, load-balanced die-level routing algorithm, which compresses timing paths and routes nets to balance channel load without degrading slack.
- We develop a practical TDM assignment method, based on continuous Lagrangian optimization over the timing graph, followed by timing-aware legalization and post-refinement to improve slack under industrial constraints.
- We demonstrate consistent slack improvement across ten contest benchmarks and eight industrial designs, with notable gains on large-scale and multi-clock systems.

II. Preliminaries

A. Timing Information

Timing constraints are fundamental to synchronous circuit correctness. A key metric is the **timing slack**, defined as the margin between the clock period and the accumulated delay of the path. Positive slack indicates timing margin, whereas negative slack implies a violation of setup constraints.

*1) **Timing Path**:* A timing path represents the basic unit of the timing analysis. Register-to-register paths, the primary focus of this work, originate from source flip-flops, traverse combinational logic and interconnect, and terminate at destination flip-flops [23]. The minimum clock period T_{clk} is bounded by the delay of the worst timing path:

$$T_{\text{clk}} \geq \max_{p \in \mathcal{P}} \sum_{e \in p} \mathit{delay}(e), \tag{1}$$

where $\mathcal{P}$ denotes the set of all timing paths. The path with maximum delay and minimal slack determines the **critical path** [24]. In our method, STA-reported timing paths are directly incorporated into the optimization model to preserve delay accuracy and clock domain association.

*2) **Clock Domain**:* A clock domain consists of elements synchronized by a common clock [25]. In multi-clock designs, direct slack comparisons across domains are invalid due to differing clock periods. To enable the analysis of multi-clock domains, we define the normalized slack $slack_n$ as:

$$slack_n = \begin{cases} \dfrac{slack \times T_{\text{clk}}}{slack_{\max} \times T_{\text{clk}}^{\max}}, & slack \geq 0, \\ \dfrac{slack}{|slack_{\min}| \times T_{\text{clk}}}, & slack < 0, \end{cases} \tag{2}$$

where $slack_{\max}$ and $slack_{\min}$ denote the maximum slack and minimum slack across all paths, and $T_{\text{clk}}^{\max}$ is the longest clock period. This normalization is applied to all slack calculations. Since no paths cross the clock domains in our target systems, all STA-reported timing paths inherit T_{clk} of their domain.

B. Architecture and Problem Formulation

Modern multi-FPGA systems must satisfy stringent timing constraints, limited interconnect capacity, and fixed signal directions. As shown in Fig. 1, interconnects are classified into inter-die and inter-FPGA links. Let E denote the complete interconnect graph, with E_{SLL} representing inter-die edges and E_{Cable} denoting inter-FPGA edges.

Communication between adjacent dies on the same FPGA utilizes dedicated SLLs with fixed delay d_{die} and sufficient capacity. These links do not support TDM, and signal assignment must remain within their physical limits. In contrast, inter-FPGA links enable TDM, allowing multiple logical signals to share limited physical channels. For any inter-FPGA edge e, the delay depends on its assigned ratio r_e and is modeled as

$$\text{delay}(r_e) \;=\; \begin{cases} \alpha + \beta, & r_e \leq 1, \\ \alpha + \beta\, r_e, & r_e > 1, \end{cases} \tag{3}$$

where α and β are constants obtained from vendor-provided timing data. All signals sharing the same physical channel must adopt a uniform TDM ratio and direction.

Formally, for each net n, a routing tree $T_n \in E$ connects its driving die $S(n)$ to all load dies $D(n)$. For each timing path $p \in \mathcal{P}$, denote $t_{\text{fixed}}(p)$ for its inherent delay. The objective is to **minimize the worst-path delay** under joint constraints.

$$\min_{\substack{T_n \subseteq E \\ r_e \in \mathcal{R}}} \max_{p \in \mathcal{P}} \left[t_{\text{fixed}}(p) + \sum_{e \in p} \text{delay}(r_e) \right] \tag{4a}$$

$$\text{s.t.} \quad \sum_{e \in E} \frac{1}{r_e} \leq IO_c, \quad \forall c, \tag{4b}$$

$$T_n \text{ spans } S(n) \cup D(n), \quad \forall n, \tag{4c}$$

$$r_e \geq \max_{n : e \in T_n} \text{demand}(n, e), \quad \forall e, \tag{4d}$$

where IO_c is the physical capacity of channel c, $\mathcal{R}$ is the set of valid TDM ratios defined by the vendor, and $\text{demand}(n, e)$ denotes the number of signals from net n assigned to edge e.

III. Methodology

A. Overview

Our framework takes as input a post-synthesis netlist with die-level partitioning, a multi-die connectivity graph, and STA-reported timing paths. Each net is defined by its driving die and a set of load dies. The output is a legal routing solution and a TDM assignment that minimize the worst-path delay under topology, direction, capacity, and timing constraints.

The full flow begins with compressing the timing paths. A die-level router then performs timing-aware load-balanced routing with direction locking and adaptive edge weighting to preserve slack. The resulting routing trees are converted into a timing graph that guides TDM assignment via Lagrangian relaxation, followed by timing-aware legalization and refinement. This unified flow balances the channel load and assigns optimized TDM ratios to improve the worst-path slack.

B. Timing-Aware Die-Level System Routing

The main workflow of our routing algorithm is illustrated in Fig. 2. Industrial-scale STA reports contain tens of millions of timing paths, necessitating compression for tractable optimization. We therefore collapse redundant paths by defining each path p to have a *cut signature* $\sigma(p) = (e_1, e_2, \ldots, e_k)$, which

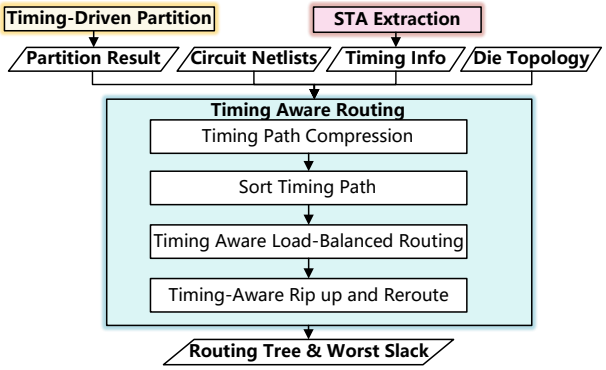

Fig. 2: Flow of timing-aware routing framework

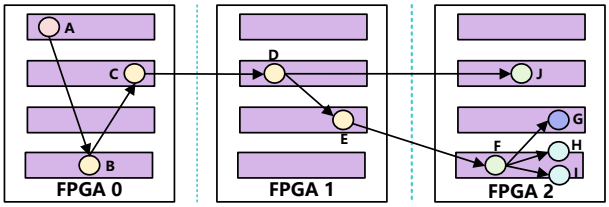

Fig. 3: Timing path compression.

is the ordered sequence of inter-FPGA edges $e_i \in E_{\text{Cable}}$ traversed from driver to sink. Paths sharing identical $\sigma(p)$ incur equivalent inter-FPGA delays under fixed TDM ratios. We therefore retain a representative per signature:

$$P' = \left\{ p^* \mid p^* = \arg \max_{p:\sigma(p)=\sigma} t_{\text{fixed}}(p) \right\}, \tag{5}$$

where P' denotes the compressed set of timing paths. This compression reduces the timing path set by over 90% while preserving the worst-path slack ($\min_{p\in P} \text{slack}(p) = \min_{p\in P'} \text{slack}(p)$), achieving **lossless critical path preservation**. Fig. 3 illustrates how three original timing paths from A to G, H, and J collapse into one representative (shared cut edges CD and EF), whereas the distinct traversal to I remains separate.

To embed timing objectives directly into die-level routing, we adopt a **Timing-Aware Load-Balanced Routing** (*TLR*) strategy, summarized in Algorithm 1. A single Floyd–Warshall pass initially computes all-pairs delay matrix $\mathbf{D}^{(0)}$ (lines 1-5), serving dual purposes: $\mathcal{O}(1)$ relay lookup for saturated channels and SLL direction locking via majority-flow analysis.

After this initialization, compressed timing paths are ordered by normalized slack; ties are resolved with the largest predicted delay derived from $\mathbf{D}^{(0)}$ (lines 6-8), ensuring nets with larger estimated delays are routed earlier to reserve low-latency channels. Each net is routed in that order with Dijkstra search, whose edge weights are dynamically updated to reflect timing criticality and channel utilization (lines 9-13).

$$w_{v_i,v_j} = \begin{cases} \infty, & (v_i,v_j) \notin E, \\ d_{\text{die}}, & (v_i,v_j) \in E_{\text{SLL}}, \\ \text{delay}(r_{v_i,v_j}), & (v_i,v_j) \in E_{\text{Cable}}, \end{cases} \tag{6}$$

where v_i denotes the i-th node in die topology graph $G(V,E)$, representing the i-th die. The variable r_{v_i,v_j} is the estimated

Algorithm 1 Timing-Aware Load-Balanced Routing (TLR)

Input: Topology weights W (Eq. (6)), nets $\mathcal{N}$, driving die $S(n)$, load dies set $D(n)$ for each net n, die topology graph $G(V,E)$
Output: Routing tree $\mathcal{T}$
1: $dist \leftarrow \infty$, $next \leftarrow -1$ // initialize
2: **for all** pair (i,j) with $W_{ij} < \infty$ **do**
3: $\quad dist[i][j] \leftarrow W_{ij}$; $next[i][j] \leftarrow j$
4: Compute $dist[i][j]$ and $next[i][j]$ for every die pair via a single **Floyd–Warshall** pass on $G(V,E)$.
5: Lock SLL direction by majority flow using $dist$
6: **for all** cut timing path p reported by STA **do**
7: $\quad slack_n(p) \leftarrow$ normalize Slack(p) // Using Eq. (2)
8: Sort $\mathcal{N}$ by *ascending* $slack_n(p)$ and *predicted_delay* from $\mathbf{D}^{(0)}$
9: **for** $n \in \mathcal{N}$ **do** // nets sorted by timing path slack
10: $\quad$ *// Dijkstra on current W*
11: $\quad$ Initialize a min-priority queue Q and push the source $S(n)$ with distance 0
12: $\quad$ Apply Dijkstra's algorithm over current edge weights W to compute shortest paths from $S(n)$
13: $\quad$ Record the predecessor of each node in an array $prev$
14: $\quad$ *// back-trace every sink*
15: $\quad$ **for** $d \in D(n)$ **do**
16: $\quad\quad$ **while** $d \neq -1$ and $\neg visited[d]$ **do**
17: $\quad\quad\quad$ Add $(prev[d], d)$ to $\mathcal{T}$; $U_{prev[d],d} \leftarrow U_{prev[d],d} + 1$
18: $\quad\quad\quad$ Update $W_{prev[d],d}$ with Eq. (6)
19: $\quad\quad\quad$ $visited[d] \leftarrow \mathbf{T}$; $d \leftarrow prev[d]$

Algorithm 2 Timing-Aware Rip-up and Reroute (TRR)

Input: Timing paths $\mathcal{P}$, cut nets $\mathcal{N}$, cost matrix C, capacity matrix U, penalty factors $\lambda_{\text{tdm}}, \lambda_{\text{die}}$, reroute rounds R
Output: Updated routing trees $\mathcal{T}$, path slacks $\{\text{slack}_p\}$
1: Initialize best slack S^*, best cut cost $(C^*_{\text{tdm}}, C^*_{\text{die}})$
2: **for** $r = 1$ to R **do**
3: $\quad p^* \leftarrow \arg\min_{p\in\mathcal{P}} \text{slack}_p$ // critical path
4: $\quad N^* \leftarrow$ extract $\mathcal{N}$ on p^* // optional expansion
5: $\quad$ Backup routing state and rip up N^* from $\mathcal{T}$, C, U
6: $\quad$ **for** $n \in N^*$ **do**
7: $\quad\quad$ route n by TLR with current C; update $\mathcal{T}, C, U$
8: $\quad$ Recompute $\{\text{slack}_p\}$; $S_{\text{new}} \leftarrow \min_p \text{slack}_p$
9: $\quad$ **if** $S_{\text{new}} < S^*$ **and** cut costs not worse **then**
10: $\quad\quad$ Accept update, set $S^* \leftarrow S_{\text{new}}$
11: $\quad$ **else**
12: $\quad\quad$ Rollback all changes
13: $\quad\quad$ **if** cut cost worsened **then**
14: $\quad\quad\quad$ $\lambda_{\text{tdm}}, \lambda_{\text{die}} \leftarrow \alpha(\lambda_{\text{tdm}}, \lambda_{\text{die}})$
15: $\quad\quad$ **else**
16: $\quad\quad\quad$ **break**

TDM ratio on edge (v_i, v_j), computed as

$$r_{v_i,v_j} = \max\left(\frac{U_{v_i,v_j}}{IO_{v_i,v_j}}, 1\right), \tag{7}$$

where U_{v_i,v_j} is current load and IO_{v_i,v_j} denotes the physical channel capacity. The ∞ weight excludes unreachable channels. Each completed route is back-traced to update the splay tree, channel usage, and edge weights (lines 15-19). These immediate updates steer later nets away from emerging congestion, achieving load balancing and maximizing slack.

To further improve timing, we propose a **Timing-Aware Rip up and Reroute** (*TRR*) algorithm that selectively re-optimizes nets on the worst-slack path (Algorithm 2). Each iteration identifies the current critical timing path (lines 2-7), optionally expanding to neighboring nets, and reroutes them via timing-aware load-balanced routing under dynamic TDM and die-cut penalties $\lambda_{\text{tdm}}, \lambda_{\text{die}}$. After rerouting, the slacks are recomputed (line 8). If the worst-path slack improves without

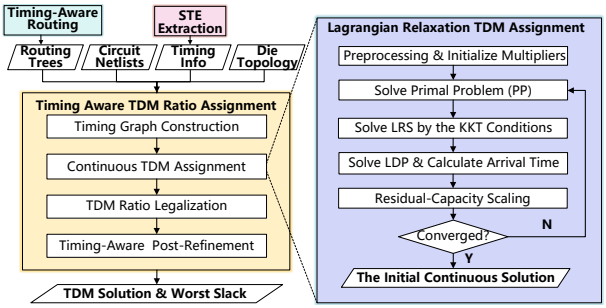

Fig. 4: Flow of timing-aware TDM assignment framework.

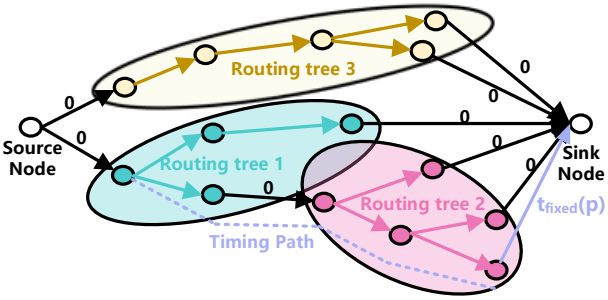

Fig. 5: Example of timing graph construction.

violating capacity, the update is accepted; otherwise, routing is reverted, and any increased cut cost triggers a penalty update by factor α (lines 12-14). The process terminates upon convergence or iteration limits, consistently reducing worst-path delay with minimal disturbance to the baseline routing.

C. Timing-Aware TDM Assignment

We solve the TDM assignment problem via a timing-graph-based optimizer, as shown in Fig. 4.

To efficiently model full path-level delays, we construct a simplified **timing directed acyclic graph** (DAG) $T(\hat{V}, \hat{E})$ from P', as shown in Fig. 5. Each colored ellipse represents the routing tree of a distinct net. The vertex set $\hat{V}$ includes one vertex per die partition, augmented by two auxiliary nodes: a global source and a global sink. Zero-delay edges connect the source to every driving partition and symmetrically connect each load partition to the sink. Suppose $\hat{e}$ is the edge connecting node i and j, and its weight is denoted by $\text{delay}_{\hat{e}}$. The edge set $\hat{E}$ consists of inter-die edges $\hat{E}_{\text{SLL}}$ with fixed delay d_{die} and inter-FPGA edges $\hat{E}_{\text{Cable}}$ with $\text{delay}(r_{\hat{e}})$, where $r_{\hat{e}}$ is the TDM ratio assigned.

Each compressed timing path p with cut signature translates into a unique path in T starting from the source node and sequentially traversing the vertices corresponding to the driving and load partitions of each edge, culminating in a sink-bound edge weighted by $t_{\text{fixed}}(p)$. Nodes and edges absent from all timing paths are similarly linked to the source and sink via zero-delay auxiliary edges, provided that they do not influence critical-path computations. The arrival times at all nodes can be obtained via the following iterative propagating procedure:

$$a_j = \max_{\hat{e}\in\hat{E}} \left(a_i + \text{delay}_{\hat{e}}\right), \quad \text{while } a_{\text{source}} = 0. \tag{8}$$

This ensures that the arrival time of the sink node, a_{sink}, matches precisely the worst-path delay of the original design.

Due to the combinatorial complexity of discrete TDM assignment, we relax each $r_{\hat{e}}$ to a continuous variable in the range $[1, r_{\max}]$ and approximate the TDM-induced delay using a linear model. This leads to the convex primal problem (*PP*):

$$\min_{\{r_e\},A} \quad A \tag{9a}$$

$$\text{s.t.} \quad a_{\text{sink}(p)} \le A, \quad \forall p \in P', \tag{9b}$$

$$a_{\text{sink}(p)} = t_{\text{fixed}}(p) + \sum_{\hat{e}\in\hat{E}_{\text{Cable}}} (\alpha + \beta r_{\hat{e}}), \tag{9c}$$

$$\sum_{\hat{e}\in\hat{E}_{\text{Cable}}} \frac{1}{r_{\hat{e}}} \le IO_{\hat{e}}, \quad \forall \hat{e} \in E_{\text{Cable}}, \tag{9d}$$

$$1 \le r_{\hat{e}} \le r_{\max}, \quad \forall \hat{e} \in E_{\text{Cable}}, \tag{9e}$$

where A is the **optimization target** representing the upper-bound of worst-path delay and $a_{\text{sink}(p)}$ is the actual arrival time at sink for timing path p.

Since PP is computationally intractable, we apply **Lagrangian relaxation** by introducing multipliers λ and μ for timing constraints (9b) and capacity constraints (9d), respectively, yielding the Lagrangian function $L_{\mu,\lambda}(r, a, A)$:

$$L_{\mu,\lambda}(r,a,A) = A + \sum_{(i,j)\in\hat{E}_{\text{Cable}}} \mu_{(i,j)}(a_i + \alpha + \beta r_{(i,j)} - a_j) + \sum_{p\in P'} \mu_p(a_{\text{sink}(p)} - A) + \sum_{e\in\hat{e}} \lambda \left(\sum_{\hat{e}\in\hat{E}_{\text{Cable}}} \frac{1}{r_{\hat{e}}} - IO_{\hat{e}} \right). \tag{10}$$

For any $\lambda, \mu \ge 0$, the minimized $L_{\lambda,\mu}(r, a, A)$ provides a *lower bound* on the primal optimal value A^*, which is the Lagrangian-relaxed sub-problem (*LRS*):

$$LRS^\star(\lambda,\mu) = \min_{\mathbf{r},\mathbf{a},\mathbf{A}} L_{\lambda,\mu}(\mathbf{r},\mathbf{a},\mathbf{A}). \tag{11}$$

To find the *tightest possible lower bound*, we maximize $LRS^\star(\lambda,\mu)$, yielding the Lagrangian dual problem (*LDP*):

$$\max_{\lambda,\mu\ge 0} LRS^\star(\lambda,\mu) = \max_{\lambda,\mu\ge 0} \min_{\mathbf{r},\mathbf{a},\mathbf{A}} L_{\lambda,\mu}(\mathbf{r},\mathbf{a},\mathbf{A}). \tag{12}$$

To solve the LDP, we minimize (10) over the variables $r_{\hat{e}}, a_i, A$ by applying the Karush–Kuhn–Tucker (KKT) optimality conditions. Setting the partial derivatives of the Lagrangian to zero yields the following conditions:

$$\frac{\partial L}{\partial r_{\hat{e}}} = 0 \implies r^\star_{\hat{e}} = \text{clip}\left(\sqrt{\frac{\lambda_{(\hat{e})}}{\beta\mu_{\hat{e}}}}, 1, r_{\max}\right), \tag{13}$$

$$\frac{\partial L}{\partial a_i} = 0 \implies \sum_{\text{in-edges}} \mu_{(k,i)} = \sum_{\text{out-edges}} \mu_{(i,j)}, \tag{14}$$

$$\frac{\partial L}{\partial A} = 0 \implies \sum_{p\in P'} \mu_p = 1, \tag{15}$$

where $\mu_{\hat{e}}$ denotes the aggregated multiplier reflecting the timing criticality of edge $\hat{e}$. These conditions show that $r^\star_{\hat{e}}$, the optimal TDM ratio, balances timing constraint ($\mu_{\hat{e}}$) and congestion cost(λ); μ forms a conserved flow; and A is a convex combination of path delays. We then initialize and

Algorithm 3 Continuous TDM Ratio Optimization Process

Input: Timing graph $T(\hat{V}, \hat{E})$
Output: Optimal continuous TDM ratio assignment $r^*_{\hat{e}}$
1: Initialize dual variable μ by Eq. (17)
2: **while** not converged and iteration count $<$ max iterations **do**
3: Update λ based on μ using Eq. (19)
4: Update TDM ratio r using Eq. (13)
5: Calculate arrival time a by Eq. (8), store if improved
6: Update μ based on timing slack using Eq. (17)
7: Scale residual capacity by γ using Eq. (20)

update the multipliers μ and λ. For μ, timing criticality is propagated backward from the sinks. For each edge (i, j),

$$DC_{(i,j)} = \frac{DC_i}{\deg^+(i)} + \text{delay}_{(i,j)}, DC_j = \sum_{(i,j)\in\hat{E}} DC_{(i,j)}, \tag{16}$$

where $DC_{(i,j)}$ represents the delay cost along the edge (i, j), combining the upstream node cost DC_i with the edge delay, and DC_j aggregates the total delay cost from all incoming edges at node j. Then in reverse topological order:

$$\mu_{(i,j)} = \frac{DC_{(i,j)}}{DC_j} \cdot \sum_{(j,k)\in\hat{E}} \mu_{(j,k)} \tag{17}$$

distributes timing criticality proportionally to the delay cost.

For λ, we enforce full utilization via (13):

$$\sum_{(i,j)\in\hat{E}(\hat{e})} \sqrt{\frac{\beta\mu_{(i,j)}}{\lambda_{\hat{e}}}} \le IO_{(\hat{e})}. \tag{18}$$

Achieving equality yields the initialization and updates:

$$\lambda_{\hat{e}} = \left(\frac{\sum_{(i,j)\in E(\hat{e})} \sqrt{\beta\mu_{(i,j)}}}{IO_{(\hat{e})}} \right)^2. \tag{19}$$

Since the minimum legal TDM ratio is 1, some channels may remain underutilized if $\sum_{\hat{e}\in c} \frac{1}{r^\star_{\hat{e}}} < IO_c$. In such cases, we redistribute the residual capacity by scaling all edges $\hat{e}$ with $r^\star_{\hat{e}} > 1$ on channel c.

$$\gamma = \frac{\sum_{r^\star_{\hat{e}}>1} \frac{1}{r^\star_{\hat{e}}} - n_1}{IO_c - n_1}, \quad r^\star_{\hat{e}} \leftarrow \gamma r^\star_{\hat{e}}, \tag{20}$$

where $n_1 = |\{\hat{e} : r^\star_{\hat{e}} = 1\}|$, achieving near 100% utilization.

Both μ and λ are updated according to (17) and (19), which follow from the KKT conditions of the underlying constrained optimization. Algorithm 3 illustrates the specific process of solving for **continuous TDM assignment**.

To translate the continuous solution $r^\star_{\hat{e}}$ into hardware-feasible ratios $\hat{r}_e \in \mathcal{R}$, we propose a **Timing-Aware TDM Legalization and Post-Refinement** algorithm (Algorithm 4). This stage ensures that the legal solution satisfies capacity constraints while preserving timing quality. Firstly, signals are grouped by physical direction d, with each group T_d assigned a discrete number of physical channels p_d based on its total inverse-ratio demand (lines 1-3). This step ensures conformity with both direction and capacity constraints. Next, a minimum-deviation packing procedure selects feasible ratio sequences for each group T_d (lines 4-11), with deviations from

Algorithm 4 TDM Legalization and Post-Refinement

Input: Continuous ratios r_c, grouped TDM signals $T = \{T_d\}$ by direction d, slack bound b
Output: Legal discrete TDM ratios r^d
1: **for** each direction d with signals T_d **do**
2: Compute total usage: $u_d \leftarrow \sum_{e\in T_d} \frac{1}{r^c_e}$
3: Allocate physical channels: $p_d \leftarrow \max(1, \lfloor u_d + 0.5 \rfloor)$
4: **for** each direction d **do**
5: Sort r_c in T_d as $r^*_1, \ldots, r^*_{|T_d|}$ ascending
6: $n \leftarrow 0$, $i \leftarrow 1$, $m \leftarrow |T_d|$
7: **while** $i \le m$ **do**
8: *choice* $\leftarrow \max\{r \in \mathcal{R} \mid r^*_i \le r \le r^*_i + b\}$
9: **for** $j \leftarrow i$ **to** $i +$ *choice* $- 1$ **do**
10: $r^d_j \leftarrow$ *choice*
11: $i \leftarrow i +$ *choice*; $n \leftarrow n + 1$
12: **for** each direction group T_d **do**
13: Initialize lower/upper bounds $l = -b$, $u = b$
14: *Binary-search* $b^* \in [l, u]$: repeatedly bisect the interval, repack T_d with margin m, and shrink the interval until the resulting channel count equals the budget p_d (tolerance ϵ)
15: Assign r^d for T_d with $b^* \leftarrow r$ // $r^d \in \mathcal{R}$
16: **repeat**
17: Compute slacks: $s_{(p,q)} = s_q + a_q - (a_p + \text{delay}_{(p,q)})$
18: Trace critical path (where $s_e = 0$)
19: **for** each critical inter-FPGA edge e_i **do**
20: **for** each e_j in same direction with $r^d_j < r^d_i$ **do**
21: $\Delta\text{delay} \leftarrow \beta(r^d_i - r^d_j)$
22: **if** $\Delta\text{delay} \le s_{e_j}$ **then**
23: Swap r^d_i, r^d_j, update a_v and slacks
24: **break**
25: **until** no critical path improves

the original continuous values bounded by a predefined slack bound b. The binary search identifies the smallest bound b^* that still satisfies the capacity constraint, minimizing the maximum rounding error across the group (lines 12-15). Finally, a post-refinement stage iteratively improves the worst-case timing by reassigning TDM ratios across edges. Each iteration identifies critical paths based on slack calculations and swaps ratio between critical and non-critical edges in the same direction, provided the swap remains within timing slack (lines 16-25). This preserves total inverse-ratio usage, maintaining legality while enhancing overall timing.

IV. Experimental Results

In this section, we evaluate the proposed framework on two categories of benchmarks: ten contest benchmarks (Case01 to Case10) [26], and eight industrial designs (Industry1 to Industry8). In contrast, the industrial designs, provided by our industrial partner, represent realistic system-level prototyping

TABLE I: Statistics of Industrial and Contest Benchmarks

Type	Design	Node	Net	FPGA	Die	Clock Period (ns)
Industry	Industry1	6503	6500	4	16	2.00
	Industry2	23120	23118	4	16	50.00
	Industry3	1221292	1280002	4	16	50.00
	Industry4	8653	9396	4	16	41.67, 50.00
	Industry5	25678	26574	4	16	37.04, 62.50, 1000.00, 30518.52
	Industry6	1921478	2034595	4	16	10.93, 20.00, 39.92, 100.00
	Industry7	3518579	3765663	4	16	50.00, 100.00
	Industry8	6267124	6480217	4	16	10.00, 13.33, 20.00, 83.33, 500.00
Contest	Case01	6	5	2	8	-
	Case02	71	86	2	8	-
	Case03	69	84	2	8	-
	Case04	452	449	2	8	-
	Case05	5084	5083	3	12	-
	Case06	127100	145660	3	12	-
	Case07	549700	76258	4	16	-
	Case08	62846	86139	4	16	-
	Case09	784814	871588	4	16	-
	Case10	3066539	3324963	5	20	-

TABLE II: The Worst Slack of the Partitioned Circuits under Different Strategies of Routing and TDM Assignment (unit: ns)

Strategy	Industry1		Industry2		Industry3		Industry4		Industry5		Industry6		Industry7		Industry8	
	P1	P2	P1	P2	P1	P2	P1	P2	P1	P2	P1	P2	P1	P2	P1	P2
TL+TA (proposed)	**-12.68**	**-12.33**	**33.88**	**38.45**	**-82.47**	**-302.52**	**-78.44**	**-78.44**	**55.86**	**53.86**	**-188.72**	**-188.72**	**-184.12**	**-289.09**	**-131.24**	**-160.75**
TL+AA	**-12.68**	**-12.33**	**33.88**	**38.45**	-219.07	-878.82	-202.79	-154.58	2.43	0.43	-209.06	-246.35	-185.61	-359.09	-203.36	-217.00
SP+TA	**-12.68**	**-12.33**	**33.88**	36.45	-96.47	-597.38	**-78.44**	**-78.44**	**55.86**	**53.86**	-250.35	-250.35	-185.41	-317.20	-313.36	-185.39
SP+AA	**-12.68**	**-12.33**	**33.88**	**38.45**	-119.07	-828.81	-202.79	-208.01	2.43	0.43	-209.06	-290.35	-235.33	-463.20	-237.36	-261.92

TL+TA: timing-aware load-balanced router + timing-aware TDM; TL+AA: timing-aware load-balanced router + average assignment TDM; SP+TA: shortest-path router + timing-aware TDM; SP+AA: shortest-path router + average assignment TDM. P1 and P2 denote different circuit partition results.

TABLE III: The Worst Slack of Partitioned Circuits

Clock Domain	Design	Huang [19] (ns)	Ours (ns)	Improve (%)
Single	Industry1	−114.11	−12.68	88.89
	Industry2	33.88	33.88	0.00
	Industry3	−263.07	−82.47	68.64
Multi	Industry4	−262.81	−78.44	70.15
	Industry5	−11.57	55.86	582.65
	Industry6	−379.06	−189.06	50.14
	Industry7	−377.67	−184.12	51.23
	Industry8	−545.36	−131.24	75.94
			Avg.	**98.45**

TABLE IV: The Worst Delay Comparison on Contest Cases

Design	1st (ns)	2nd (ns)	3rd (ns)	Huang [19] (ns)	Ours (ns)	Improve (ns)
Case01	6.5	6.5	6.5	**6.5**	**6.5**	0.0
Case02	7.5	7.5	7.5	**7.5**	**7.5**	0.0
Case03	11.5	14.5	11.5	**11.5**	**11.5**	0.0
Case04	19.5	19.5	18.5	**18.5**	**18.5**	0.0
Case05	135.5	136.0	132.5	**130.0**	132.0	-2.0
Case06	213.0	260.0	287.0	202.0	**171.0**	31.0
Case07	84.5	102.5	**76.0**	77.5	78.0	-0.5
Case08	124.0	145.5	123.0	**111.5**	114.5	-3.0
Case09	150.0	196.5	177.0	147.5	**144.0**	3.5
Case10	4657.5	4745.0	5700.0	4740.5	**4600.5**	**140.0**
Avg.	–	–	–	–	–	**17.5**

scenarios, with realistic netlists, physical constraints, and STA reports. Among them, Industry1 through Industry3 adopt a single clock domain, while Industry4 through Industry8 involve multiple clock domains. Table I summarizes the key statistics for each benchmark. The proposed framework and the counterparts are implemented in C++. All experiments are conducted on a Linux server with Intel Xeon 32-core E5-2630 CPU and 256 GB memory. We present results from three perspectives to evaluate the effectiveness of our methods.

A. Evaluation on Industry Designs

To enable fair comparison with previous work, we re-implement the entire flow proposed by [19], as their source code is unavailable. Since their method demonstrates strong performance and is well documented at the die level, we adopt it as our baseline. Table III reports the worst-path slack under both approaches, categorized by clock domain.

Our timing-aware routing and TDM assignment reduce the worst slack by an average of 98.45%. In the most challenging single-clock design, our method improves slack by 88.89%, while in the largest multi-clock design, the improvement reaches 75.94%. All results are obtained under the same constraints and environment. These results confirm that explicit path-level timing guidance and the load-balanced co-optimization of routing and TDM assignment are essential for closing timing in large-scale, multi-die systems for industrial designs, particularly when multi-clock domains are involved.

B. Evaluation on Public Contest Benchmarks

To test generalization, we apply our method to the official benchmarks of the 2023 EDA Elite Challenge [20], which optimizes the worst delay for a single net under a simplified timing model. To ensure comparability, we adopt the same objective function and legal TDM ratios as in the original contest. We compare against the best published results from the top three official solutions, as well as the results reported in [19], which we independently re-implement.

Table IV reports the worst delays achieved by various methods across ten contest benchmark designs. For the first four small cases (Case01–04), all methods achieve identical results, indicating consistent performance under constrained scenarios. For the remaining six cases, our method achieves notable improvements in three cases, with the largest improvement of 140 ns in Case10. In the other three cases, the differences are minor. On average, our approach improves the worst delay by 17.5 ns. These results demonstrate that our full approach remains effective under relaxed constraints and consistently delivers robust performance, especially on large-scale circuits.

C. Ablation Study on Strategy Components

To isolate the impact of each component, we evaluate four variants of our flow on the eight industrial designs, using two distinct partition results for each. Table II shows that our full pipeline (TL+TA) consistently produces the best slack in all designs and partition results. The row corresponding to our complete method is highlighted for clarity. Substituting either component leads to noticeable slack degradation, with differences reaching up to hundreds of nanoseconds on some designs. These results confirm that both the timing-aware load-balanced routing and the timing-aware TDM assignment are indispensable for achieving timing closure, especially under practical industrial constraints.

V. Conclusion

This paper presents a timing-aware optimization flow that jointly solves die-level load-balanced routing and TDM assignment while satisfying constraints on topology, direction, channel capacity, and timing. Compressed STA-reported timing paths are fed to the timing-aware router, which then constructs a timing graph to guide a multistage, timing-aware TDM assignment based on Lagrangian relaxation and post-refinement. Compared to the existing methods, the proposed flow achieves 98% improvement, on average, in the worst-path slack on industrial designs, with similar superiority in the EDA Elite Challenge Contest benchmarks.

REFERENCES

[1] S.-C. Chen *et al.*, "Simultaneous Partitioning and Signals Grouping for Time-Division Multiplexing in 2.5D FPGA-Based Systems," in *Proc. ICCAD*, 2018, pp. 1–8.

[2] S2C. (2025) Prodigy s8-100 logic system-vp1902 prototyping. [Online]. Available: https://www.s2cinc.com/products/prototyping-s8-100-with-VP1902/

[3] AMD. (2022) Super long line (sll) routes – vivado design methodology guide (ug949). [Online]. Available: https://docs.amd.com/r/en-US/ug949-vivado-design-methodology/Super-Long-Line-SLL-Routes

[4] ——. (2022) Vivado design suite user guide: Vivado properties (ug912). [Online]. Available: https://docs.amd.com/r/2022.2-English/ug912-vivado-properties/SLR

[5] Z. Di *et al.*, "LEAPS: Topological-Layout-Adaptable Multi-Die FPGA Placement for Super Long Line Minimization," *IEEE TCAS-I*, vol. 71, no. 3, pp. 1259–1272, 2024.

[6] RapidWright. (2022) Xilinx architecture overview – rapidwright documentation. [Online]. Available: https://www.rapidwright.io/docs/Xilinx_Architecture.html#slr-super-logic-region

[7] M. Turki, Z. Marrakchi, and H. Mehrez, "Signal multiplexing approach to improve inter-FPGA bandwidth of prototyping platform," *Design Automation for Embedded Systems*, vol. 19, pp. 223–242, Sep. 2015.

[8] J. Babb, R. Tessier, M. Dahl, S. Z. Hanono, D. M. 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, Jun. 1997.

[9] M. Khalid, "Routing architecture and layout synthesis for multi-FPGA systems," Ph.D. dissertation, University of Toronto, 2000.

[10] M. Farooq, M. Awais, U. Younas, and M. Alam, "Inter-FPGA interconnect topologies exploration for multi-FPGA systems," in *International Conference on Engineering Emerging Technologies (ICEET)*, 2018, pp. 1–6.

[11] 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.

[12] 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 *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[13] 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.

[14] 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 *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[15] 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 *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2021, pp. 176–182.

[16] I. M, T. Y, N. Y, and et al., "Optimal time-multiplexing in inter-FPGA connections for accelerating multi-FPGA prototyping systems," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 91, no. 12, pp. 3539–3547, 2008.

[17] I. M, T. Y, and N. Y, "Globally optimal time-multiplexing in inter-FPGA connections for accelerating multi-FPGA systems," in *2009 International Conference on Field Programmable Logic and Applications*. IEEE, 2009, pp. 212–217.

[18] C.-W. Pui and E. F. Y. Young, "Lagrangian relaxation-based time-division multiplexing optimization for multi-FPGA systems," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 25, no. 2, pp. 1–23, 2020.

[19] 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)*, May 2024, pp. 324–331.

[20] S2C. (2023, Nov) Design of FPGA die-level system-routing algorithms. [EB/OL]. [Online]. Available: https://edaoss.icisc.cn/file/cacheFile/2023/11/8/fdd08c36acbb432e9d56a85400e39125.pdf

[21] The OpenROAD Project. (2024) Opensta: Open source static timing analyzer. [Online]. Available: https://github.com/The-OpenROAD-Project/OpenSTA

[22] Synopsys. (2024) What is static timing analysis (sta)? [Online]. Available: https://www.synopsys.com/glossary/what-is-static-timing-analysis.html

[23] VLSI Expert. (2011) Static timing analysis (sta) - basic timing concepts. [Online]. Available: https://www.vlsi-expert.com/2011/03/static-timing-analysis-sta-basic-timing.html

[24] I. Bustany, G. Gasparyan, A. B. Kahng, I. Koutis, B. Pramanik, and Z. Wang, "An open-source constraints-driven general partitioning multi-tool for vlsi physical design," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2023, pp. 1–9.

[25] Clock domain crossing. [Online]. Available: https://en.wikipedia.org/wiki/Clock_domain_crossing

[26] EDA Challenge Committee. (2023) 2023 China Postgraduate IC Innovation Competition · EDA Elite Challenge Contest. [Online]. Available: http://edachallenge.cn