ALIFRouter: A Practical Architecture-Level Inter-FPGA Router for Logic Verification

Zhen Zhuang[†], Xing Huang[‡], Genggeng Liu[†⋆], Wenzhong Guo[†], Weikang Qian[§], and Wen-Hao Liu[*]

[†]College of Mathematics and Computer Science, Fuzhou University, Fuzhou, China
[‡]Technical University of Munich, Munich, Bavaria, Germany
[⋆]State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[§]University of Michigan-Shanghai Jiao Tong University Joint Institute, Shanghai Jiao Tong University, Shanghai, China
[*]Block Implementation, ICD, Cadence Design Systems, Austin, TX, USA
Email: liu_genggeng@126.com

***Abstract*—As the scale of VLSI circuits increases rapidly, multi-FPGA prototyping systems have been widely used for logic verification. Due to the limited number of connections between FPGAs, however, the routability of prototyping systems is a bottleneck. As a consequence, timing division multiplexing (TDM) technique has been proposed to improve the usability of prototyping systems, but it causes a dramatic increase in system delay. In this paper, we propose *ALIFRouter*, a practical architecture-level inter-FPGA router, to improve the chip performance by reducing the corresponding system delay. ALIFRouter consists of three major stages, including i) routing topology generation, ii) TDM ratio assignment, and iii) system delay optimization. Additionally, a multi-thread parallelization method is integrated into the three stages to improve the efficiency of ALIFRouter. With the proposed algorithm, major performance indicators of multi-FPGA systems such as signal multiplexing ratio can be improved significantly.**

I. Introduction

With the development of technology node, logic verification has become a very time-consuming step. In the SoC design process, it is estimated that 60-80% of an ASIC design time is spent in performing verification [1]. Since the FPGA prototyping approach can achieve a good trade-off between runtime and cost, it is widely used in industry to make logic verification cheaper and faster. For a prototyping system, it is difficult to be realized by one FPGA [2]. Therefore, many FPGAs are connected together to construct a complete system. Since the number of inter-FPGA signals usually exceeds the number of I/O pins, a timing division multiplexing (TDM) technique was proposed to transmit different signals in one wire at different time [3]. However, the inter-FPGA delay increases due to signal multiplexing. The signal multiplexing ratio known as TDM ratio can be used to measure the system delay.

TDM ratio is usually determined after inter-FPGA routing in the whole design flow [4]. There are some works about the design of FPGA prototyping systems. The work [5] proposed a method to optimize partitioning and TDM at the same time. However, its optimization target is not the TDM ratio. The works [6]–[8] proposed some ILP-based routing algorithms to optimize TDM. Nevertheless, the TDM ratio of these works is usually an arbitrary integer, which is not practical [9]. The works [10], [11] proposed more practical ILP-based algorithms for TDM ratio assignment after inter-FPGA routing. However, it is difficult for those algorithms proposed by the works [6]–[8], [10], [11] to obtain good solutions with appropriate runtime because of the much larger scale of FPGA systems.

The problem solved in this work is a more complicated and more practical problem in which the TDM ratio constraints and the net groups are given according to design purpose. The target of the problem is to optimize routability and TDM ratio at the same time for the inter-FPGA routing stage. In this paper, we propose *ALIFRouter*, a practical architecture-level inter-FPGA router, to optimize routability and TDM ratio simultaneously. Compared with MSFRoute [12], ALIFRouter can obtain better optimization results of TDM ratio and faster runtime. The major contributions are shown below:

- A Dijkstra-based routing algorithm for inter-FPGA routing is used to route all nets for the routability optimization.
- A simplified TDM ratio assignment algorithm is proposed to assign TDM ratio for each edge. Furthermore, an optimization algorithm for system delay is proposed to reduce the maximum TDM ratio of net groups. The TDM ratio can be effectively optimized by this algorithm.
- A multi-thread parallelization method is integrated into the overall flow to further improve the efficiency.
- Compared with the state-of-the-art inter-FPGA router, ALIFRouter can reduce TDM ratio by up to 11.61% with a 1.97X speedup on average.

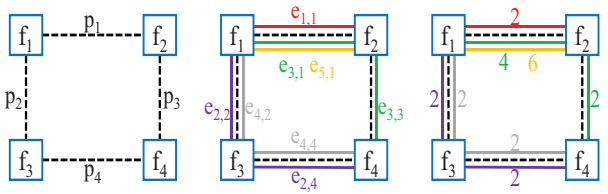
II. Problem Formulation

Timing division multiplexing (TDM) technique is used to solve the lack of routing channels such that many signals can be transmitted in one channel at different time. However, the side-effect of TDM is the increase of system delay. TDM ratio related to the system clock frequency can become an indicator to measure system delay. For the routing graph of a system, the TDM ratio of edge e is shown below:

$$Tr(e) = \frac{D(e)}{Cap(e)} \tag{1}$$

where $D(e)$, $Cap(e)$ and $Tr(e)$ represent the demand of e, the capacity of e, and the TDM ratio of e, respectively. In practice, the capacity of each edge is fixed to be 1 for multiplexing hardware implementation.

978-3-9819263-5-4/DATE21/©2021 EDAA



(a) Graph. (b) Routing. (c) Solution.

Fig. 1: Illustration of the problem.

The inter-FPGA routing problem based on TDM was defined by the 2019 ICCAD contest [13]. We will introduce this problem by an example shown in Fig. 1. This problem is modeled by a graph. The graph includes a set of FPGAs represented by F and a set of FPGA connection pairs represented by P. Furthermore, a set of nets represented by N and a set of net groups represented by NG are defined, respectively. Net groups are defined due to the design purpose. For example, the nets with similar attributes or same power consumption would be in the same net group. The objective is to route all nets and assign TDM ratio for each net to minimize the maximum TDM ratio for net groups. In Fig. 1, different nets are highlighted by different colors. f_i represents the i-th FPGA and p_k represents the k-th FPGA connection pair. $e_{j,k}$ represents the edge of net n_j using p_k. To analyze the system delay, each edge $e_{j,k}$ should be assigned a TDM ratio $etr_{j,k}$. The TDM ratio constraints of a FPGA connection pair are shown below:

$$\sum_{e_{j,k} \in epl_k} \frac{1}{etr_{j,k}} \leqslant 1 \tag{2}$$

where epl_k represents the set of edges using p_k. The value of $etr_{j,k}$ must be an even integer greater than zero due to multiplexing hardwire implementation. Fig. 1(c) illustrates the TDM ratio assignment for each edge. For p_1, the TDM ratio of $e_{1,1}$, $e_{3,1}$ and $e_{5,1}$ is 2, 4 and 6, respectively. Due to $1/2 + 1/4 + 1/6 < 1$, p_1 satisfies the TDM ratio constraints. The TDM ratio of a net is the sum of the TDM ratio of all its edges. The TDM ratio of a net group is the sum of the TDM ratio of all its nets. Since the system clock is determined by the maximum TDM ratio of net groups, the optimization objective of this problem is to minimize the TDM ratio of net groups as shown below:

$$Minimize: ngmtr = \{x | x = max(ngtr_1, \cdots, ngtr_\alpha)\} \tag{3}$$

where $ngtr_i$, α and $ngmtr$ represent the TDM ratio of net group ng_i, the number of net groups and the maximum TDM ratio of net groups, respectively.

III. The Proposed Framework of ALIFRouter

ALIFRouter can be divided into three stages: the routing topology generation stage, the TDM ratio assignment stage, and the system delay optimization stage.

1) *Routing Topology Generation*:

Algorithm 1 Routing Topology Generation Algorithm

Input: global routing graph G, net set N, FPGA set F
1: Sort N according to two metrics
2: Initialize G
3: **for** each net n_j in N **do**
4: Select an FPGA f_s from F_n
5: $V \leftarrow f_s$
6: $U \leftarrow F - f_s$
7: $E \leftarrow \varnothing$
8: $g \leftarrow G$
9: **while** $F_n \neq \varnothing$ **do**
10: Find shortest paths from V to U by g
11: Select f_i with the minimum cost
12: $V \leftarrow V + F_p$
13: $U \leftarrow U - F_p$
14: $E \leftarrow E_p$
15: Update g
16: **end while**
17: Record E
18: Update G
19: **end for**

The routing topology generation stage is the first stage to obtain the routing tree of each net. The FPGAs of each net are connected together. Since Dijkstra algorithm can be effectively used to construct a Steiner tree like the work [14], a Dijkstra-based routing algorithm for inter-FPGA routing is used to route all nets. The pseudo code is shown in Algorithm 1.

At first, nets are sorted by two metrics (line 1). The net of the net group with more nets has higher priority. For the nets belonging to the same net group, the net with more FPGAs has higher priority. Then the routing graph is initialized (line 2). Based on the graph, the cost of each FPGA connection pair is assigned to be 1. Finally, nets are routed iteratively (lines 3-19). For each net, we use an effective Dijkstra-based Steiner tree algorithm to obtain the topology. In Algorithm 1, F_n represents the set of the target FPGAs that must be connected to net n_j. g represents the routing graph used to route the current net. V represents the FPGAs which belong to the Steiner tree, and U represents the rest FPGAs in F. E represents the edges of the net. f_s is the first FPGA which is selected from F_n. During constructing a Steiner tree, all data is initialized at first (lines 4-8). Then the shortest paths from V to each target FPGA are found by Dijkstra algorithm (lines 9-16). For each shortest path from V to a target FPGA, all shortest paths are found from V to U by Dijkstra algorithm (line 10). Then, we find FPGA f_i with the shortest path length from V to f_i compared with other FPGAs of F_n (line 11). Next, V and U are updated by F_p which are the FPGAs of the path from V to f_i (lines 12-13). E is updated by E_p which are the edges of the path from V to f_i (line 14). Finally, based on the shortest path from V to f_i, the routing graph is updated (line 15). The costs of the edges of the path from V to f_i are all set to be 0. When all target FPGAs are routed, a Steiner tree is constructed, and E is recorded (line 17). Based on E, G is updated (line 18). The cost of each FPGA connection pair used by the Steiner tree adds 1 in each iteration.

2) *TDM Ratio Assignment*:

TDM ratio assignment stage assigns TDM ratio for each edge

of a net topology. In this stage, the procedure of the work [12] is simplified to make ALIFRouter more efficient.

For each edge, the TDM ratio is calculated based on the edge number of its net groups considering the TDM ratio constraints. Therefore, the assignment of TDM ratio can be efficiently dealt by processing FPGA connection pairs one by one based on the edge number of net groups. A method to compute edge ratio is proposed for the edges belonging to the same FPGA connection pair. The edge ratio of $e_{j,k}$ in p_k is shown below:

$$pct_{j,k} = \frac{ngmec_{j,k}}{\sum_{e_{o,k} \in epl_k} ngmec_{o,k}} \tag{4}$$

where $ngmec_{j,k}$ represents the maximum edge number of the net groups which include $e_{j,k}$. $pct_{j,k}$ is the edge ratio of $e_{j,k}$. Due to the definition of edge ratio, $etr_{j,k}$ is shown below:

$$etr_{j,k} = \frac{1}{pct_{j,k}} \tag{5}$$

At the beginning, for each edge $e_{j,k}$ of each FPGA connection pair p_k, the maximum edge number of the net groups which include $e_{j,k}$ is calculated. It is the basis of the calculation of the edge ratio. Then TDM ratio is assigned for each FPGA connection pair. For each FPGA connection pair, the first step is to calculate the edge ratio of each edge according to Eq. (4). Then the TDM ratio of each edge is assigned according to Eq. (5). Experimental results show that the TDM ratio assignment stage of ALIFRouter is efficient and effective.

3) *System Delay Optimization*:

In the TDM ratio assignment stage, initial TDM ratio calculated by a rough method is assigned. Therefore, the initial assignment solution should be optimized to reduce the maximum TDM ratio of net groups. An optimization algorithm for system delay is proposed in this section.

The pseudo code of the proposed optimization algorithm for system delay is shown in Algorithm 2. This algorithm includes two steps. The first step is the reduction step (lines 1-7). In this step, the TDM ratio of the edges of the net groups with larger TDM ratio should be reduced. However, the TDM ratio of the edges of the net groups with smaller TDM ratio should be added. The second step is the legalization step (lines 8-15). After the reduction step, some FPGA connection pairs may violate the TDM ratio constraints. Therefore, these FPGA connection pairs should be legalized in this step.

The first step is processed for each net because the reduction of the former nets may influence the later nets. Nets are sorted by the maximum TDM ratio of its net groups from large to small (line 1). In this order, it is flexible for the net groups with larger TDM ratio to be optimized. For each edge $e_{j,k}$, its TDM ratio $etr_{j,k}$ is updated by the following equation (line 4):

$$etr^{'}_{j,k} = \frac{staratio \times etr_{j,k}}{mng_j} \tag{6}$$

where $etr^{'}_{j,k}$ is the new TDM ratio and $staratio$ is the optimization target which can be defined by users. mng_j represents the maximum TDM ratio of the net groups including net n_j.

Algorithm 2 Optimization Algorithm for System Delay

```
Input: net set N
1: Sort N by the TDM ratio of the net groups of each net
2: for each net n_j in N do
3:     for each edge e_{j,k} of n_j do
4:         Update etr_{j,k}
5:     end for
6:     Update ngl_j
7: end for
8: for each FPGA connection pair p_k in P do
9:     if p_k satisfies the constraint then
10:        Update old epl_k
11:    else
12:        Update added e_{j,k}
13:        Legalize reduced e_{j,k}
14:    end if
15: end for
```

After the reduction of the TDM ratio of net n_j, the TDM ratio of each net group of ngl_j (the net groups including n_j) should be updated to make the later reduction better (line 6).

The second step is processed for each FPGA connection pair, because the TDM ratio constraints are defined for FPGA connection pairs. After the reduction step, if FPGA connection pair p_k satisfies the TDM ratio constraints, TDM ratio $etr_{j,k}$ of edge $e_{j,k}$ is changed by new TDM ratio $etr^{'}_{j,k}$ (line 10). However, if p_k cannot satisfy the TDM ratio constraints, edge $e_{j,k}$ whose TDM ratio is added can directly use new TDM ratio $etr^{'}_{j,k}$ (line 12). But edge $e_{j,k}$ whose TDM ratio is reduced should be legalized by the following equation (line 13):

$$etr^{''}_{j,k} = \frac{etr^{'}_{j,k} \times rec \times (rec + ad)}{1 - ad} \tag{7}$$

where rec is the sum of the reciprocals of reduced $etr^{'}_{j,k}$ and ad is the sum of the reciprocals of added $etr^{'}_{j,k}$.

4) *Multi-Thread Parallelization*:

In the routing topology generation stage, the routing of each net can be parallelized. However, the eighteenth line of Algorithm 1 should be locked to avoid the resource conflict of different nets. In the TDM ratio assignment stage, the processing of FPGA connection pairs is completely parallelized. In the system delay optimization stage, the reduction step can be parallelized for each net. However, the sixth line of Algorithm 2 should be locked to avoid the resource conflict of different nets. The legalization step can be completely parallelized.

IV. Experimental Results

The algorithm of this work is implemented in C++ language on a Linux server. The benchmarks including nine designs are released by the 2019 ICCAD contest [13].

We compare ALIFRouter with a state-of-the-art inter-FPGA router, named MSFRoute [12], as shown in Table I. "AVG" represents the arithmetic average ratios based on our results. The data of ALIFRouter and MSFRoute shown in Table I is obtained from the programs running with eight threads in the same environment. Compared with MSFRoute, ALIFRouter can reduce TDM ratio by up to 11.61% with an average reduction rate of 3.20%. Since the values of TDM ratio are very large,

TABLE I: Comparison with MSFRoute about runtime and TDM ratio ($\times 10^3$)

Design	MSFRoute [12]		Ours	
	TDM Ratio	Time (s)	TDM Ratio	Time (s)
S1	39	8.63	38	4.23
S2	32097	20.51	31713	6.33
S3	129396	193.28	127519	91.15
S4	7301	963.00	6453	353.73
S5	5370	1195.36	4749	724.46
S6	15759857	1360.90	15747863	1780.43
H1	409654	38.43	408855	14.92
H2	45947775	1058.37	45940000	501.14
H3	4871917	1056.78	4865697	2066.20
AVG	1.04	1.97	1.00	1.00

TABLE II: Comparison with the top-3 ICCAD contest winners about TDM ratio ($\times 10^3$)

Design	A	B	C	Ours
	TDM Ratio	TDM Ratio	TDM Ratio	TDM Ratio
S1	38	40	40	38
S2	31740	32094	32007	31713
S3	127589	129290	128206	127519
S4	6396	6335	7049	6453
S5	4663	4613	5190	4749
S6	15749497	15759303	15751029	15747863
H1	408871	409891	409300	408855
H2	45941183	45952595	45942030	45940000
H3	4865352	4872933	4867326	4865697
AVG	1.78	3.11	3.33	1.56

these reduction rates show that ALIFRouter can effectively reduce system delay. Furthermore, for each benchmark, the results of ALIFRouter are better than those of MSFRoute. Compared with MSFRoute, ALIFRouter can achieve up to 3.24X speedup with a 1.97X speedup on average. Therefore, ALIFRouter can efficiently reduce runtime.

The experimental results shown in Table I are obtained from the program with eight threads. Compared with the program processed by one thread, the program processed by eight threads can achieve up to 5.75X speedup with a 3.79X speedup on average. The multi-thread method can achieve 3.52X, 4.21X and 4.34X speedup by using 8, 16 and 24 CPUs, respectively.

We compare ALIFRouter with the top-3 winners at the 2019 ICCAD contest. We rank all routers to make experimental comparison more visible. Firstly, we rank the maximum TDM ratio of net groups of each benchmark between the top-3 teams. Then we calculate the average rank of all benchmarks of each team and sort them from small to large. According to this order, these three teams are A, B and C, respectively. Finally, we rank the top-3 routers and ALIFRouter together.

As shown in Table II, ALIFRouter achieves the best average rank which means it achieves the best solution quality in terms of system delay compared with the top-3 winners. "AVG" represents the arithmetic average ranks. Furthermore, ALIFRouter achieves the best optimization solutions of TDM ratio of most benchmarks. In conclusion, ALIFRouter is effective. Since the binary codes of the top-3 winners are not available, the runtime of different routers cannot be obtained. Therefore, the runtime comparison cannot be shown.

V. Conclusion

We propose ALIFRouter, a practical architecture-level inter-FPGA router including routing topology generation, TDM ratio assignment, and system delay optimization. Due to the effective optimization of TDM ratio, the system delay of prototyping systems can be effectively reduced. Compared with the top-3 ICCAD contest winners, ALIFRouter can achieve better solutions. Compared with MSFRoute, ALIFRouter can reduce TDM ratio by up to 11.61% with a 1.97X speedup on average.

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China under Grants No. 61877010 and No. 11501114, the State Key Laboratory of Computer Architecture (ICT, CAS) under Grant No. CARCHB202014, and the Fujian Natural Science Funds under Grant No. 2019J01243. Genggeng Liu is the corresponding author of the article.

References

[1] M. Turki, Z. Marrakchi, H. Mehrez, and M. Abid, "Signal multiplexing approach to improve inter-FPGA bandwidth of prototyping platform," *Design Automation for Embedded Systems*, vol. 19, no. 3, pp. 223–242, 2015.

[2] S. Hauck, "The roles of FPGAs in reprogrammable systems," *Proceedings of the IEEE*, vol. 86, no. 4, pp. 615–638, 1998.

[3] J. Babb, R. Tessier, and A. Agarwal, "Virtual wires: Overcoming pin limitations in FPGA-based logic emulators," in *Proc. IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 142–151, 1993.

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

[5] S.-C. Chen, R. Sun, and Y.-W. Chang, "Simultaneous partitioning and signals grouping for time-division multiplexing in 2.5D FPGA-based systems," in *Proc. ICCAD*, pp. 4:1–4:7, 2018.

[6] M. Inagi, Y. Takashima, and Y. Nakamura, "Globally optimal time-multiplexing of inter-FPGA connections for multi-FPGA prototyping systems," *IPSJ Trans. on System LSI Design Methodology*, vol. 3, pp. 81–90, 2010.

[7] M. Inagi, Y. Takashima, and Y. Nakamura, "Globally optimal time-multiplexing in inter-FPGA connections for accelerating multi-FPGA systems," in *Proc. International Conference on Field Programmable Logic and Applications*, pp. 212–217, 2009.

[8] M. Inagi, Y. Nakamura, Y. Takashima, and S. Wakabayashi, "Inter-FPGA routing for partially time-multiplexing inter-FPGA signals on multi-FPGA systems with various topologies," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E98-A, no. 12, pp. 2572–2583, 2015.

[9] W. N. N. Hung and R. Sun, "Challenges in large FPGA-based logic emulation systems," in *Proc. International Symposium on Physical Design*, pp. 26–33, 2018.

[10] C.-W. Pui and E. F. Y. Young, "Lagrangian relaxation-based time-division multiplexing optimization for multi-FPGA systems," *ACM Trans. on Design Automation of Electronic Systems*, vol. 25, no. 2, pp. 21:1–21:23, 2020.

[11] C.-W. Pui and E. F. Y. Young, "Lagrangian relaxation-based time-division multiplexing optimization for multi-FPGA systems," in *Proc. ICCAD*, pp. 1–8, 2019.

[12] Z. Zhuang, G. Liu, X. Huang, X. Jia, W.-H. Liu, and W. Guo, "MSFRoute: Multi-stage FPGA routing for timing division multiplexing technique," in *Proc. Great Lakes Symposium on VLSI*, pp. 107–112, 2020.

[13] Y.-H. Su, R. Sun, and P.-H. Ho, "2019 CAD Contest: System-level FPGA routing with timing division multiplexing technique," in *Proc. ICCAD*, pp. 1–2, 2019.

[14] C. J. Alpert, W.-K. Chow, K. Han, A. B. Kahng, Z. Li, D. Liu, and S. Venkatesh, "Prim-Dijkstra revisited: Achieving superior timing-driven routing trees," in *Proc. International Symposium on Physical Design*, pp. 10–17, 2018.