An Analytical Approach for Time-Division Multiplexing Optimization in Multi-FPGA based Systems

Chak-Wa Pui
CUHK
cwpui@cse.cuhk.edu.hk

Gang Wu
Synopsys Inc.
gangw@synopsys.com

Freddy Y. C. Mang
Synopsys Inc.
fmang@synopsys.com

Evangeline F. Y. Young
CUHK
fyyoung@cse.cuhk.edu.hk

ABSTRACT

To increase the utilization of FPGAs in multi-FPGA based systems, time-division multiplexing (TDM) is a widely used technique to accommodate a large number of inter-FPGA signals. However, with this technique, the delay imposed by the inter-FPGA connections becomes significant. Previous research shows that TDM ratio of signals can greatly affect the performance of a system. In this paper, we extend previous problem formulation to meet more general constraints in multi-FPGA based systems and propose a novel approach to solve it. In particular, to optimize system clock period effectively and efficiently, we propose a two-step analytical framework, which first gives a continuous result using a non-linear conjugate gradient-based method and then finalizes the result optimally by a dynamic programming-based discretization algorithm. For comparison, we also solve the problem using an integer linear programming (ILP)-based method. Experimental results show that our approach can improve the system clock period by about 7% on top of a well optimized inter-FPGA routing result. Moreover, our approach scales for designs over 400K nodes while ILP-based method is not able to finish for designs with 2K nodes.

1 INTRODUCTION

In recent years, field-programmable gate array (FPGA) becomes very popular in various fields such as deep learning [10] and data center [3] due to its increasing logic density. Although the scale of FPGAs has greatly increased, it is still unlikely to fit the entire design into one FPGA in applications like logic emulation and rapid prototyping of large designs [4]. Hence, multi-FPGA systems are usually used. A multi-FPGA system consists of multiple FPGAs which are connected using direct hardwired connections or a programmable interconnection network that may consist one or more field-programmable interconnect chips [9].

In multi-FPGA systems, the available pin count of the FPGAs is relatively small compared to the number of inter-FPGA signals, which greatly limits the utilization of the FPGA logic resources in a multi-FPGA system. Time-division multiplexing (TDM) is a method that multiplexes the use of FPGA pins and inter-FPGA physical wires among multiple inter-FPGA signals. Since TDM can reduce the number of physical wires needed, it is commonly adopted in multi-FPGA systems [1]. In this way, the number of logical pins available in each FPGA can be effectively increased, which leads to higher logic utilization per FPGA. But this technique makes the system clock period longer since the inter-FPGA signal delay is lengthened due to time-multiplexing. Although minimizing the number of inter-FPGA signals during compilation can reduce the negative effect of time-multiplexing on delay, TDM optimization is still an important step in the compilation flow since different TDM ratios can result in very different system clock period [5].

In the compilation flow of multi-FPGA systems, TDM ratios are usually determined after inter-FPGA routing [1]. Several methods are proposed to optimize the TDM ratios in recent works [6–8]. In [8], an integer linear programming (ILP)-based method for 2-FPGA systems is introduced. TDM ratio and TDM path length are minimized in the formulation, where both resources and signal directions are considered. It tries to put non-critical inter-FPGA nets in TDM wires to improve the utilization and timing of the systems, which results in a 0-1 decision problem for each net. The ILP problem is divided into sub-problems and solved by integer relaxation. We can see that there are only two choices for their nets, which are time-multiplexed or not. Hence, the resulting TDM ratios between two FPGAs will only have two values, which is not practical in multi-FPGA systems [5]. The work [7] extends [8] to support multi-FPGA systems. It is further extended to consider TDM optimization in partially connected multi-FPGA systems [6]. Due to the increasing number of inter-FPGA signals in multi-FPGA systems, ILP-based methods are no longer able to produce high-quality results with scalable running time. A recent work [2] proposes a framework that performs TDM assignment and partitioning simultaneously. During partitioning, the weight of each edge is modified to model the delay induced by TDM. For signal grouping, a binary search based method is used to exhaust all the possible groupings. However, the system clock period is only used as the final performance metric, which is optimized by minimizing an estimated timing criticality cost. Moreover, their TDM ratios can be arbitrary integers, which is not practical [5]. Therefore, a high-quality, scalable TDM optimization algorithm for multi-FPGA systems is needed.

In this paper, we propose an analytical framework for the TDM optimization problem. The major contributions are summarized as follows:

SLIP '19, June 2, 2019, Las Vegas, NV, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

- An analytical framework is proposed to optimize the system clock period globally in the multi-FPGA TDM optimization problem. It first generates a continuous result of the TDM ratios with minimized system clock period under relaxed constraints. A discretization algorithm is then applied to remove all the constraint violations honoring the continuous result. Compared to previous works, our approach supports wires with a user given set of TDM ratios, which is more practical in multi-FPGA systems. To the best of our knowledge, this is the first work that directly optimizes the system clock period in a practical and effective way.
- A non-linear conjugate gradient (CG)-based continuous solver is proposed to solve the continuous TDM optimization problem which is effective and scalable. Several techniques are proposed to optimize the system clock period and the number of TDM ratio violations.
- A top-down dynamic programming (DP)-based algorithm is proposed to discretize the continuous TDM result, which can optimize the timing and total displacement exactly. Various pruning techniques are proposed to accelerate the process.

The remainder of this paper is organized as follows. Section 2 gives the preliminaries of the problem. Section 3 first gives an overview of our approach and then introduces its details. Section 4 shows the experimental results, and we finally conclude this paper in Section 5.

2 PRELIMINARIES

In this section, we first explain our targeting architecture and its compilation flow. The problem definition will then be introduced. Note that, throughout this paper, *tdmNet* refers to inter-FPGA net for simplicity and the notations we used are shown in Table 1.

2.1 Targeting Architecture

We consider a multi-FPGA system with multiplexed hardwired inter-FPGA connections where two FPGAs are adjacent if they are directly connected in the system. In our targeting system, two adjacent FPGAs are called an FPGA-pair.

Since the number of tdmNets is much larger than the number of physical wires between FPGAs, time-multiplexed wires are usually used to connect different FPGAs. Figure 1 is an illustration of the inter-FPGA time multiplexed I/Os in our targeting architecture. In such systems, only the tdmNets in the same direction and with the same TDM ratio can be assigned to the same wire. Moreover, the number of tdmNets in the wire with TDM ratio n should be no greater than n. In our target architecture, I/O TDM implementation uses an order-agnostic approach, whose scheduling assumes the same worst-case delay for all signals. Thus, the transmission delay is a function that only depends on the TDM ratio. Given the tdmNet net_i going through a wire with TDM ratio x_i, the delay of $e_{p,q}$ can be calculated as $(b_{p,q} \cdot x_i + c_{p,q})$ for all $e_{p,q} \in E_i$, where $b_{p,q}, c_{p,q}$ are parameters related to the time multiplexing architecture. Due to the architecture limitations, the TDM ratio can only be 1 or multiples of 8 instead of an arbitrary integer.

Table 1: Notations

T	The set of all FPGA-pairs.
t_i	The set of tdmNets of the FPGA-pair i in T.
p_i	Wire limit of the FPGA-pair i in T.
g_{sink}	Sink of the timing graph.
g_i	Gate i.
$e_{p,q}$	Edge from g_p to g_q in timing graph.
at_p, rt_p	Arrival time and required arrival time of g_p.
X	An assignment of the TDM ratios of the system.
E_i	The set of edges correspond to tdmNet net_i.
x_i	TDM ratio of tdmNet net_i.
$x_{p,q}$	TDM ratio of $e_{p,q}$ and $x_{p,q} = x_i$ for all $e_{p,q} \in E_i$.
$b_{p,q}, c_{p,q}$	Architecture related parameters of $e_{p,q}$.
tc	The set of all discrete TDM choices, which are $\{1, 8, 16, 24, \ldots, 1600\}$.
$delay_{p,q}$	Delay of $e_{p,q}$.
$e^c_{p,q}$ ($e^x_{p,q}$)	An edge from g_p to g_q representing an intra-FPGA (inter-FPGA) net.
$x_i^{tc_j}$	Binary variable indicates whether tdmNet net_i is assigned to discrete TDM choice tc_j.
n^f_{k,tc_j} (n^b_{k,tc_j})	Integer variable indicates the number of forward (backward) wires used in the FPGA-pair k, whose TDM ratios are tc_j.
t^f_k (t^b_k)	The set of forward (backward) tdmNets of the FPGA-pair k in T.

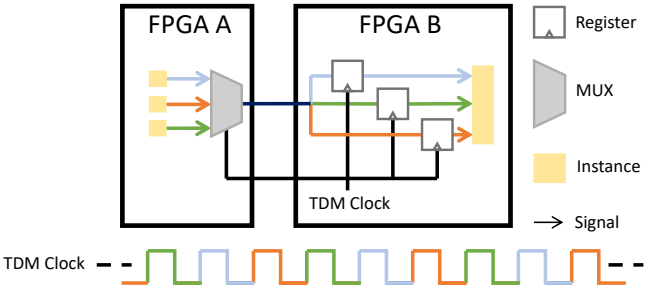


Figure 1: An illustration of inter-FPGA time-multiplexed I/Os where three different signals share one wire.

2.2 Compilation Flow

A typical compilation flow for multi-FPGA systems consists of various steps, such as partitioning, placement, routing, etc. The first step of the compilation flow is logic synthesis and technology mapping where the given circuit is mapped into a netlist of primitive elements such as lookup-tables (LUTs), flip-flops (FFs), RAMs, DSPs, etc. The netlist is then divided into partitions such that each partition can fit into a single FPGA and the number of inter-FPGA connections is minimized. Inter-FPGA placement puts each partition into a distinct FPGA on the board. Inter-FPGA routing is then performed which considers both system performance and routing resources. It determines the routing topology and the TDM ratio for each tdmNet. It must ensure that the number of tdmNets between any two adjacent FPGAs will not require more physical wires than available under their TDM specification. Given the routing result,

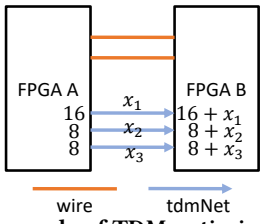


Figure 2: An example of TDM optimization problem.

TDM optimization is applied to optimize the TDM ratios regarding the system clock period. After that, pin assignment will choose the physical wire and pins for each tdmNet subject to the TDM constraints from the previous steps. Finally, the placement and routing of individual FPGAs are performed.

2.3 Problem Definition

TDM ratio represents the maximum number of signals that a physical wire can accommodate. In the TDM optimization problem, given the system architecture and the timing graph constructed from the inter-FPGA routing result, we need to determine the TDM ratio for each tdmNet. The objective is to minimize the system clock period, which equals to the arrival time of the sink in the timing graph. The TDM ratios of the tdmNets in the FPGA-pair i should satisfy the following constraints.

- Each TDM ratio should be either 1 or multiples of 8.
- Only tdmNets with the same TDM ratio and in the same direction can be assigned to the same wire.
- The number of tdmNets in a wire with TDM ratio n is no greater than n.
- The total number of used wires cannot exceed p_i.

An example of the TDM optimization problem is shown in Figure 2 where all $c_{p,q}$ and $b_{p,q}$ are assumed to be 0 and 1 respectively. There are two wires between the FPGA-pair and three subnets from FPGA A to B. Let the TDM ratios for these three subnets be x_1, x_2, x_3 and the arrival times of their driving pins are 16, 8, 8 respectively. The system clock period is the maximum arrival time at the primary outputs of FPGA B which is $at_{\text{sink}} = \max(16 + x_1, 8 + x_2, 8 + x_3)$. The optimal assignment of TDM ratios of this system will be $(x_1, x_2, x_3) = (1, 8, 8)$ so that tdmNets net_2 and net_3 can share one physical wire and $\lceil \frac{1}{x_1} \rceil + \lceil \frac{1}{x_2} + \frac{1}{x_3} \rceil \leq 2$.

3 TDM OPTIMIZATION FRAMEWORK

As shown in Figure 3, our TDM optimization algorithm is a two-step approach. Given an inter-FPGA routing result, the corresponding timing graph is first constructed. Our continuous solver is then built based on the timing graph, which minimizes the system clock period with most of the TDM constraints ignored. To be specific, each TDM ratio should not be less than 1 and $\sum_{j \in t_i} \frac{1}{x_j} \leq p_i$ holds for every FPGA-pair.

After that, we can get a continuous result of the TDM ratios with optimized system clock period. Our discretization algorithm is then performed such that all the TDM ratio violations are removed while the continuous result is honored.

Details of our continuous solver and discretization algorithm will be discussed in Sections 3.1 and 3.2 respectively.

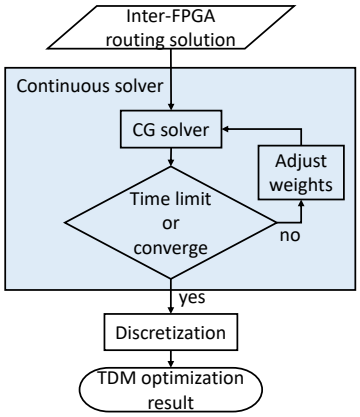


Figure 3: The overall flow of our approach.

3.1 Continuous Solver

Our continuous solver is an iterative solver based on an unconstrained non-linear CG method. At each iteration, the CG solver will optimize the objective and the parameters are adjusted accordingly. The solver will terminate after the result converges adequately or the time limit is reached.

Instead of formulating it as a constrained optimization problem, our formulation is shown as follows,

$$\min \quad \alpha \cdot f(X) + \sum_{i \in T} \beta_i \cdot g_i(X) + \sum_{i \in T} \sum_{j \in t_i} \gamma_j \cdot h(x_j) \tag{1a}$$

$$s.t. \quad g_i(X) = \max((\sum_{j \in t_i} \frac{1}{x_j}) - p_i, 0) \tag{1b}$$

$$h(x) = \max(1 - x, 0), \tag{1c}$$

where $\alpha, \beta_i, \gamma_j$ are weights, f is the system clock period, g_i and h represent the violations of wire limit constraint and minimum TDM ratio value constraint respectively. Note that, Equation (1) is a convex non-differentiable function since f, h, g are all convex and α, β, γ are non-negative values.

3.1.1 Objective Approximation. This section shows how the objective function (Equation (1)) is smoothened in the CG solver.

$f(X)$ is the system clock period which is calculated as,

$$f(X) = \max_{e_{p,sink}} (at_p + delay_{p,\text{sink}}).$$

Log-Sum-Exp (LSE) function is used to smoothen the max function in forward timing propagation. To avoid overflow, LSE is transformed as shown in Equation (2),

$$\max_{i=0}^{k} y_i \approx \log \sum_{i=0}^{k} e^{y_i} = \max_{i=0}^{k} y_i + \log \sum_{i=0}^{k} e^{y_i - \max_{i=0}^{k} y_i}. \tag{2}$$

Since the exponential function is frequently used in timing propagation and gradient calculation, a fast look-up table (LUT) and approximation are adopted as shown in Equation (3).

$$e^x \approx \begin{cases} (1 + \frac{x}{256})^{256}, & \text{if } 0 \leq x < 1, \\ \text{return } e^{\lfloor x \rfloor} \text{ from LUT}, & \text{if } 1 \leq x \leq 200, \\ +\infty, & \text{if } x > 200, \end{cases} \tag{3}$$

where a look-up table from e^1 to e^{200} is pre-built. Note that, with this approximation, the runtime for computing e^x can be greatly reduced while the percent error for $0 \leq x < 1$ is below 0.2% and the error for $x \geq 1$ is either negligible or irrelevant for the problem.

For g and h, two piecewise functions are used to replace them as shown in Equations (4) and (5), which are still smooth and convex.

$$g_i^p(X) = \begin{cases} ((\sum_{j \in t_i} \frac{1}{x_j}) - p_i)^2, & \text{if } \sum_{j \in t_i} \frac{1}{x_j} > p_i, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

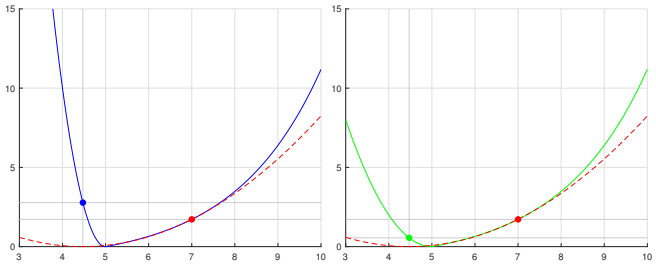
$$h^p(x) = \begin{cases} (1-x)^2, & \text{if } x < 1, \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

3.1.2 Gradient Calculation. Our objective consists of three parts f^p, g^p and h^p, which are the approximated functions of f, g and h respectively. The gradients of g^p and h^p can be easily calculated. On the other hand, f^p is a function based on the timing graph whose gradient cannot be directly calculated. In our approach, we use backpropagation to obtain the gradient by transforming the timing graph into a computational graph. Given the assignment of the TDM ratios, forward timing propagation is first performed to get the arrival time of each gate. We then calculate the local gradients for each gate with respect to its incoming nodes in the computational graph. Finally, we do backpropagation which applies chain rule to compute the derivatives. It is easy to see that the local gradient calculation of each gate node can be parallelized and the backward and forward propagation can be parallelized among the nodes at the same level in the topological order.

3.1.3 Weight Adjustment. In each iteration, we can get an optimized result of X regarding the objective after solving the non-linear CG problem. From this result, the coefficients α, β, γ are adjusted accordingly to balance the cost of f^p, g^p, h^p in the objective. To be specific, we will calculate the violation using Equations (1b) and (1c). If $g_i(X)$ is larger than 0 for the FPGA-pair i, we will double β_i such that the corresponding violation is given more penalty. Similarly, if the value of x_i is less than one, γ_i will be doubled.

Since f^p and g^p are meaningless if any x_i in the equation is less than one, the minimum TDM ratio value constraint is treated as a barrier constraint in our solver, which means that γ is set to a large value from the beginning of the iterations. Even with the barrier constraint, it is still possible that the CG solver explores places where $x_i < 1$. Hence, x_i in f^p and g^p is substituted by $max(1, x_i) \approx \frac{\log(e^{10} + e^{10x_i})}{10}$.

Unlike quadratic CG where the step size of each iteration can be calculated precisely, the step size in non-linear CG is obtained by a line search algorithm[13]. At each CG iteration, given the direction of the current point X, the line search algorithm will try to approximate the objective as a quadratic function and pick the step size which minimizes the approximated function. For example, in Figure 4(a), line search is performed at $x = 7$, it approximates the function (blue solid line) as a quadratic function (red dashed line). The minimum point of the approximate function is $x = 4.5$, hence the step size calculated by line search is 2.5 in this iteration. Figure 4 shows two different settings of α, β where both of them start from the point $x = 7$, the one with larger β results in a steeper curve (blue solid line) and is trapped in the local optimal ($x = 7$) while the one



(a) large β (b) small β

Figure 4: A one-dimensional example of how different α, β settings behave in line search. Green and blue solid lines are the curves of the objective function. Red dashed line is the curve of the approximated quadratic function at $x = 7$.

with smaller β (green solid line) can descend further to a smaller value. Hence, if α is set to a value such that $\sum_{i \in T} \beta_i \cdot g_i^p(X)$ is large comparing with $\alpha \cdot f(X)$, it is easy to get trapped in a local optimal point since the minimum value found by line search may easily be greater than the current value. Based on this knowledge and the experimental results, we set α to a value such that $\sum_{i \in T} \beta_i \cdot g_i^p(X)$ is small compared to $\alpha \cdot f(X)$, which will lead to faster convergence speed.

3.2 Discretization

Our discretization algorithms are based on a top-down DP framework. We consider two different objectives, the total displacement from the continuous solution and timing. In our DP formulation, each FPGA-pair is independent and hence they can be discretized in parallel. Details of the algorithm for each FPGA-pair are shown in Sections 3.2.1 and 3.2.2. Note that, in our discretization algorithms, the discrete TDM choices are traversed in ascending order. For simplicity, we only discuss circumstances where the tdmNets of the same FPGA-pair are all in the same direction. The algorithms can be easily extended to support two directions.

3.2.1 Displacement-Driven Discretization. Since the continuous solver optimizes the TDM ratios in a global view, its results should be honored during discretization. Hence, minimizing the total displacement is one reasonable objective for discretization. The problem formulation of the FPGA-pair A is shown in Equation (6).

$$\min \quad \sum_{k \in t_A} |x_k^{cont} - x_k^{discr}| \tag{6a}$$

$$s.t. \quad \text{All TDM ratio constraints are satisfied.} \tag{6b}$$

As shown in Algorithm 1, a top-down DP framework is applied. The sequence is first sorted in a non-decreasing order of their continuous TDM ratios. $\texttt{getMinPA}_{\text{DISP}}(i, pp)$ will then generate the optimal TDM ratios for the i^{th} to n^{th} tdmNets in the sequence using pp wires. Finally, the TDM ratio of each tdmNet is restored. In the following, we show an example of how to compute $\texttt{getMinPA}_{\text{DISP}}$. Given three discrete TDM choices (1, 8 and 16), $\texttt{getMinPA}_{\text{DISP}}(i, pp)$ is equal to the minimum of $cost_1, cost_8, cost_{16}$,

Algorithm 1 Displacement-Driven Discretization.

Input: The continuous TDM ratios of the FPGA-pair A.
Output: A legal assignment of TDM ratios of the FPGA-pair A.

1: sort the tdmNets in the t_A in non-decreasing order of their continuous solutions, let $x[k]$ be the TDM ratio of the k^{th} tdmNet in the sorted sequence;
2: $\texttt{getMinPA}_{\texttt{DISP}}(0, p_A)$;
3: restore the best solutions from the DP;

4: **function** $\texttt{getMinPA}_{\texttt{DISP}}(i, pp)$
5: return the best cost at (i, pp) if computed already;
6: return 0 if $i \geq |t_A|$;
7: return $+\infty$ if $\lceil \frac{|t_A|-i}{1600} \rceil > pp$;
8: $prevIdx \leftarrow 0$;
9: **for each** $tc_q \in tc$ no less than $\texttt{nearestTDM}(x[i])$ **do**
10: $minIdx \leftarrow prevIdx + 1$;
11: increase $minIdx$ as long as $minIdx + 1 < tc_q$ and tc_q equals $\texttt{nearestTDM}(x[i + minIdx + 1])$;
12: increase $prevIdx$ as long as $prevIdx + 1 < tc_q$ and $x[prevIdx + i + 1] \leq tc_q$;
13: $curBest \leftarrow +\infty$;
14: **for** $j \in range(i + minIdx, i + tc_q - 1)$ **do**
15: $curDisp \leftarrow \sum_{k=i}^{j} |x[k] - tc_q|$;
16: jump to line 9 if $curDisp \geq curBest$;
17: $cost \leftarrow curDisp + \texttt{getMinPA}_{\texttt{DISP}}(j + 1, pp - 1)$;
18: $curBest \leftarrow \min(cost, curBest)$;
19: update the best solution at (i, pp) if $cost$ is less than the best cost at (i, pp);
20: **end for**
21: **end for**
22: return the best cost at (i, pp);
23: **end function**

which are computed as below,

$$cost_1 = \texttt{getMinPA}_{\texttt{DISP}}(i + 1, pp - 1) + |x[i] - 1|,$$

$$cost_8 = \min_{0 \leq j < 8} (\texttt{getMinPA}_{\texttt{DISP}}(i + j + 1, pp - 1) + \sum_{k=i}^{i+j} |x[k] - 8|),$$

$$cost_{16} = \min_{0 \leq j < 16} (\texttt{getMinPA}_{\texttt{DISP}}(i + j + 1, pp - 1) + \sum_{k=i}^{i+j} |x[k] - 16|).$$

Note that for $cost_8$ and $cost_{16}$, the minimum among $0 \leq j < 8$ and $0 \leq j < 16$ are considered because we can assign 1 to 8 and 1 to 16 tdmNets respectively to the pp^{th} wire.

Some pruning techniques are applied to accelerate the process without changing the optimality. As shown on line 9, we skip the discrete TDM choices smaller than the nearest one of $x[i]$ (obtained by `nearestTDM`) because smaller choices will induce more displacement and use more wire resources. $minIdx$ and $prevIdx$ denote the minimum numbers of tdmNets in the pp^{th} wire under the current and next discrete TDM choice respectively. On line 11, we increase $minIdx$ as long as tc_q is the nearest discrete TDM choice of $x[i + minIdx + 1]$ because smaller $minIdx$ will increase the number of wires used without reducing the total displacement. On line 12, we increase $prevIdx$ as long as $x[i + prevIdx + 1] \leq tc_q$. It will be

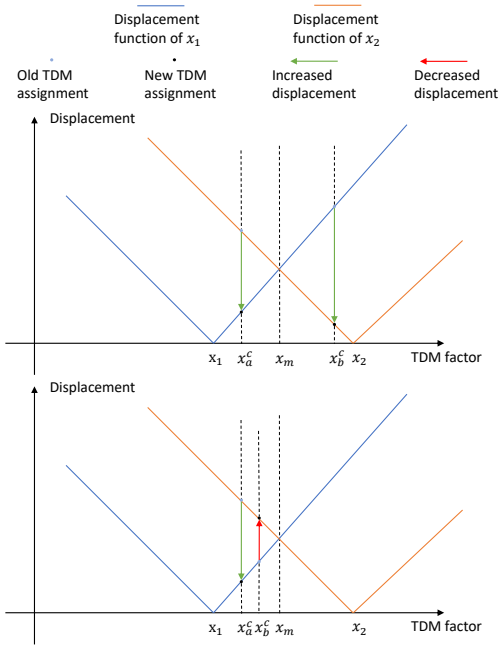


Figure 5: An illustration of swapping the TDM ratios of two tdmNets.

the starting value of $minIdx$ for the next discrete TDM choice tc_{q+1} since $\sum_{k=i}^{j} |x[k] - tc_q| < \sum_{k=i}^{j} |x[k] - tc_{q+1}|$ holds for any j no greater than $i + prevIdx$. On line 16, the loop ends since a larger j will only increase the total displacement ($curDisp$) of the pp^{th} wire which is already larger than the best cost ($curBest$) of the current discrete TDM choice tc_q.

Given a tdmNet net_a, let its TDM ratios before and after discretization be x_a^c and x_a^d respectively. Theorem 1 proves that Algorithm 1 can obtain the optimal solution for the problem shown in Equation (6).

Lemma 1. *Given a solution of Equation* (6) *where* $x_a^c < x_b^c$ *and* $x_a^d > x_b^d$, *we can swap their TDM ratios without increasing the cost of the solution.*

Proof. There are three cases: (1) $x_b^d < x_a^d < x_a^c$, (2) $x_b^c < x_b^d < x_a^d$ and (3) $x_b^d < x_a^c < x_a^d$ or $x_b^d < x_b^c < x_a^d$. The difference before and after swapping is $d = |x_a^c - x_a^d| + |x_b^c - x_b^d| - |x_a^c - x_b^d| - |x_b^c - x_a^d|$. For case 1, $d = (x_a^c - x_a^d) + (x_b^c - x_b^d) - (x_a^c - x_b^d) - (x_b^c - x_a^d) = 0$. For case 2, $d = (x_a^d - x_a^c) + (x_b^d - x_b^c) - (x_a^d - x_b^c) - (x_b^d - x_a^c) = 0$. Hence, swapping will not change the cost in the first two cases. An illustration of case 3 is given in Figure 5. We can see that if x_a^c and x_b^c are on different sides of x_m where $|x_m - x_b^d| = |x_m - x_a^d|$, both of their displacement will be reduced after swapping. If they are on the same side of x_m, the total displacement is also reduced. Hence, if $x_a^c < x_b^c$ and $x_a^d > x_b^d$, swapping their TDM ratios will not increase the cost of the solution. □

Corollary 1. *Given an optimal solution of Equation* (6) *where no swapping as described in Lemma 1 can be performed, it is true that* $x_a^d \leq x_b^d$ *if* $x_a^c \leq x_b^c$.

Theorem 1. *Algorithm 1 can get the optimal solution of Equation* (6).

Proof. We can see that Algorithm 1 exhausts all possible ways of partitioning the given non-decreasing sequence of continuous TDM ratios and returns the best partition in minimizing the total displacement. However, the optimality of Algorithm 1 (on the problem shown in Equation (6)) depends on whether there exists an optimal solution of the problem that can be transformed into a partitioning of a non-decreasing sequence of the tdmNets' continuous TDM values.

Given an optimal solution of Equation (6) where no swapping as described in Lemma 1 can be performed, we can first sort the sequence in a non-decreasing order of their discrete TDM ratios. According to Corollary 1, it is true that for all pairs a and b, if $x_a^d \leq x_b^d$, then $x_a^c \leq x_b^c$. For tdmNets that have the same discrete TDM ratios, they can also be sorted in a non-decreasing order of their continuous TDM ratios without changing the objective value. In this way, we can get a partitioning of a non-decreasing sequence of the tdmNets' continuous TDM ratios while the objective value is still optimal. □

3.2.2 Timing-Driven Discretization. Slack ratio (Equation (7)) is a metric to measure the timing criticality of a tdmNet in the netlist.

$$\text{slack\_ratio}_{p,q} = 1 + \frac{at_p + b_{p,q} \cdot x_{p,q} + c_{p,q} - rt_q}{at_{sink}}, \forall e_{p,q}. \quad (7)$$

In our timing-driven discretization, the TDM ratio assignment is legalized according to the slack ratio of the continuous result. The problem formulation of the FPGA-pair A is shown in Equation (8). The arrival time is calculated given the continuous result and is fixed during the optimization. In our targeting designs, $b_{p,q}$ of each edge $e_{p,q}$ is the same among all the tdmNets.

$$\min \quad \max_{i \in t_A}(K_i + B_i \cdot x_i) \quad (8a)$$

$$s.t. \quad K_i = \max_{e_{p,q} \in E_i} \frac{at_p + at_{\text{sink}} - rt_q + c_{p,q}}{at_{\text{sink}}}, \forall i \in t_A \quad (8b)$$

$$B_i = \max_{e_{p,q} \in E_i} \frac{b_{p,q}}{at_{\text{sink}}}, \forall i \in t_A \quad (8c)$$

$$\text{All TDM ratio constraints are satisfied.} \quad (8d)$$

Details of our algorithm are shown in Algorithm 2, where a top-down DP framework is applied. B_i and K_i for each tdmNet are first calculated according to the continuous result as shown in Equations (8c) and (8b). The input sequence is then sorted in a non-increasing order of their Ks. Note that, a larger K indicates that a tdmNet is more timing critical since it is the constant part of the slack ratio as shown in Equation (8). $\texttt{getMinPA}_{\text{SR}}(i, pp)$ will generate the optimal TDM ratios for the i^{th} to n^{th} tdmNets in the sequence using pp wires. Note that, in our algorithm, we will use the resources of each wire as much as possible. Finally, the TDM ratio of each tdmNet is restored. In the following, we show an example of how to compute $\texttt{getMinPA}_{\text{SR}}$. Given three discrete TDM choices (1, 8 and 16), $\texttt{getMinPA}_{\text{SR}}(i, pp)$ is equal to the minimum of $cost_1, cost_8, cost_{16}$, which are computed as below,

$cost_1 = \max(K[i] + 1 \cdot B[i], \texttt{getMinPA}_{\text{SR}}(i+1, pp-1))$,

$cost_8 = \max(K[i] + 8 \cdot B[i], \texttt{getMinPA}_{\text{SR}}(i+8, pp-1))$,

$cost_{16} = \max(K[i] + 16 \cdot B[i], \texttt{getMinPA}_{\text{SR}}(i+16, pp-1))$.

Algorithm 2 Timing-Driven Discretization.

Input: The continuous TDM ratios of the FPGA-pair A.
Output: A legal assignment of TDM ratios of the FPGA-pair A.

```
calculate K_i, B_i for all i ∈ t_A as shown in Equation (8).
sort the tdmNets in the t_A in non-increasing order of their Ks, let K[i], B[i] be the K, B of the i^th tdmNet in the sorted sequence;
getMinPA_SR(0, p_A);
restore the best solutions from the DP;

function getMinPA_SR(i, pp)
    return the best cost at (i, pp) if computed already;
    return 0 if i ≥ |t_A|;
    return +∞ if ⌈(|t_A| − i)/1600⌉ > pp;
    for each tc_q ∈ tc do
        msr ← K[i] + B[i] · tc_q;
        exit loop if msr ≥ bestCost[i][pp];
        cost ← max(msr, getMinPA_SR(i + tc_q, pp − 1));
        update the best solution at (i, pp) if cost is less than the best cost at (i, pp);
    end for
    return the best cost at (i, pp);
end function
```

As proved in Theorem 2, Algorithm 2 can get the optimal solution of the problem shown in Equation (8).

Lemma 2. *Algorithm 2 can get the optimal partitioning of an input sequence sorted in a non-increasing order of Ks.*

Proof. Similar to Algorithm 1, Algorithm 2 can also be viewed as a partitioning algorithm of the input sequence. The main difference is that it assumes the length of each partition is equal to its TDM ratio except that of the last partition. If this assumption is true, it is obvious that this algorithm exhausts all possible ways of partitioning the input sequence and returns the optimal solution. Next, we will prove that this assumption is true if the given sequence is in a non-increasing order of their Ks, that is:

$$F(i+x, p-1) \leq F(i+x-l, p-1),$$

where F denotes $\texttt{getMinPA}_{\text{SR}}$ and $0 < l < x$. Every solution for $F(i+x-l, p-1)$ can be transformed to a solution for $F(i+x, p-1)$ with the same or smaller cost. To be specific, given a TDM assignment of the $(i+x-l)^{th}$ to n^{th} tdmNets using $p-1$ wires, we can fit the $(i+x)^{th}$ to n^{th} tdmNets into these wires using the same TDM configuration which will result in the same cost. Therefore, $F(i+x, p-1) \leq F(i+x-l, p-1)$. It means that we should use the resources of each wire as much as possible. □

Theorem 2. *Algorithm 2 can get the optimal solution of Equation* (8).

Proof. As shown in Lemma 2, our algorithm is optimal given a sequence in a non-increasing order of their Ks. However, the optimality of Algorithm 2 (on the problem shown in Equation (8)) depends on whether there exists an optimal solution of the problem that can be transformed into a partitioning of a non-increasing sequence of the tdmNets' Ks.

Here, we show how an optimal solution is transformed into such a partitioning. Given an optimal solution of Equation (8), for two tdmNets net_i and net_j, if net_j is more critical ($K_i < K_j$) but the discrete TDM ratio of net_j is larger than that of net_i ($x_i^d < x_j^d$), their TDM ratios can be swapped without changing the the optimality of the solution since $\max(K_i + B_i \cdot x_i^d, K_j + B_j \cdot x_j^d) \geq \max(K_i + B_i \cdot x_j^d, K_j + B_j \cdot x_i^d)$. After all these swappings, we can first sort the sequence in a non-decreasing order of their discrete TDM ratios. For tdmNets that have the same discrete TDM ratios, they can also be sorted in a non-increasing order of their Ks without changing the objective value. In this way, we can get a partitioning of a non-increasing sequence of the tdmNets' Ks while the objective is still optimal. □

In our implementation, we traverse the discrete TDM choices of each tdmNet starting from the nearest one of its continuous solution instead of one (line 9). Even though this will change the optimality of the result, it may give a better system clock period because a TDM ratio far from the continuous solution will make the slack ratio estimation very inaccurate and reduce the correlation between slack ratio and system clock period.

4 EXPERIMENTAL RESULTS

In this work, all algorithms are implemented in C++ and embedded into a multi-FPGA system compilation flow. In Section 3.1, we use WNLIB [12] as our non-linear CG solver. In Section 4.1, GLPK [11] is used as our ILP solver. The experiments are performed on a Linux machine with an Intel Xeon CPU with 20 cores (no hyper-thread) and 248GB memory. Details of the benchmarks are shown in Table 2, where the constant edges represent the intra-FPGA connections and the TDM edges represent the tdmNets. In the followings, we will first introduce the formulation of our baseline algorithm and then analyze the performance of our proposed framework.

4.1 Integer Linear Programming Baseline

Since there is no prior work that directly works on the same problem, an ILP-based method is proposed as our baseline for comparison. The objective and constraints of our ILP are listed in Equation (9) and the notations are shown in Table 1.

$$\min \quad at_{\text{sink}} \tag{9a}$$

$$s.t. \quad \sum_{tc_j \in tc} x_i^{tc_j} = 1, \forall i \tag{9b}$$

$$n_{k,tc_j}^{dir} \geq \frac{\sum_{i \in t_k^{dir}} x_i^{tc_j}}{tc_j}, \forall k \in T; \forall tc_j \in tc; dir \in \{f, b\} \tag{9c}$$

$$\sum_{tc_j \in tc} n_{k,tc_j}^{b} + n_{k,tc_j}^{f} \leq p_k, \forall k \in T \tag{9d}$$

$$at_q \geq at_p + delay_{p,q}, \forall e_{p,q}^{c} \tag{9e}$$

$$at_q \geq at_p + b_{p,q} \cdot \sum_{tc_j \in tc} x_{p,q}^{tc_j} \cdot tc_j + c_{p,q}, \forall e_{p,q}^{x}. \tag{9f}$$

Equation (9b) ensures that each tdmNet can only have one TDM ratio. Equations (9c) and (9d) ensure that the total usage of forward and backward wires of each FPGA-pair is under the wire limit. Equations (9e) and (9f) are the constraints for timing propagation.

Table 2: Statistics of our designs.

Design	#Nodes	#TDM edges	#Constant edges
design1	194239	111242	189832
design2	321629	187230	325239
design3	479359	245961	480321
design4	194694	110760	234820
design5	174099	103691	194692
design6	339004	197672	346480
design7	94510	53404	82609
design8	129189	76644	106004
design9	216803	127224	181525
design10	19147	11751	15060
design11	35796	19396	37438
design12	2586	1677	3961
design13	232	96	201
design14	845	326	788

Table 3: Comparison with ILP-based method on the small designs.

Design	System clock period (ns)			Runtime (s)	
	Inter-FPGA routing	ILP	TM-flow	ILP	TM-flow
design12	393	N/A	393	>2hrs	17
design13	87	87	87	52	0.1
design14	137	137	137	49.2	0.1

4.2 Results Analysis

Our TDM optimization is performed after inter-FPGA routing whose result is the starting point of our algorithm. It is worth mentioning that the inter-FPGA result is from a well-optimized compilation flow which already selects a TDM ratio for each tdmNet while considering timing. It is a relatively good starting point but our method can still further improve on it. Note that, the continuous solver is followed by timing-driven discretization in the TM-flow. On the other hand, in the DISP-flow, the continuous solver is followed by the displacement-driven discretization.

For small designs, we can obtain the same results as the optimal ILP-based method with much shorter runtime as shown in Table 3. Moreover, the ILP-based method reaches the time limit (2 hours) in a slightly larger design, which shows that the ILP-based method is not scalable in this extended formulation where wires in the same FPGA-pairs can have different TDM ratios. Note that, in our targeting architecture, given 201 discrete TDM choices, the number of $x_i^{tc_j}$ for each tdmNet is 201. Hence, the number of variables increases dramatically as the number of tdmNets increases. For small designs, the inter-FPGA routing may already get the optimal TDM assignment, which is the reason that both ILP and our framework have the same system clock period as the inter-FPGA routing result.

In our experiment, the ILP-based method cannot generate a feasible solution for any of those large scale designs. Hence, we

Table 4: System clock period of our methods on the large designs with more than 10K nodes.

Design	System clock period (ns)			
	Inter-FPGA routing	Continuous solver	Discretization	
			TM-flow	DISP-flow
design1	359 (1)	328 (0.913)	333 (0.928)	335 (0.931)
design2	526 (1)	494 (0.939)	494 (0.940)	494 (0.940)
design3	617 (1)	575 (0.931)	579 (0.938)	597 (0.967)
design4	520 (1)	445 (0.855)	451 (0.868)	458 (0.881)
design5	485 (1)	419 (0.864)	422 (0.871)	436 (0.900)
design6	227 (1)	202 (0.890)	223 (0.983)	214 (0.943)
design7	179 (1)	154 (0.860)	162 (0.906)	162 (0.906)
design8	275 (1)	240 (0.874)	265 (0.963)	265 (0.964)
design9	242 (1)	226 (0.935)	226 (0.937)	230 (0.953)
design10	171 (1)	162 (0.943)	171 (0.996)	164 (0.959)
design11	210 (1)	197 (0.936)	197 (0.938)	197 (0.938)
Average		(0.904)	(0.933)	(0.935)

Table 5: Displacement and runtime of our approach with different discretization methods on the large designs with more than 10K nodes.

Design	Avg. displacement		Runtime (s)	
	TM-flow	DISP-flow	TM-flow	DISP-flow
design1	0.29 (1)	0.13 (0.448)	1402 (1)	1790 (1.277)
design2	0.29 (1)	0.11 (0.379)	295 (1)	552 (1.873)
design3	1.05 (1)	0.07 (0.067)	3661 (1)	3672 (1.003)
design4	1.11 (1)	0.11 (0.099)	815 (1)	907 (1.113)
design5	2.64 (1)	0.15 (0.057)	1093 (1)	1126 (1.031)
design6	0.35 (1)	0.07 (0.200)	2158 (1)	2247 (1.041)
design7	0.34 (1)	0.04 (0.118)	632 (1)	673 (1.065)
design8	6.25 (1)	0.23 (0.037)	845 (1)	927 (1.098)
design9	0.58 (1)	0.08 (0.138)	132 (1)	321 (2.427)
design10	0.9 (1)	0.42 (0.467)	90 (1)	89 (0.990)
design11	0.26 (1)	0.18 (0.692)	15 (1)	25 (1.721)
Average		(0.246)		(1.331)

only show the results of our method in Table 4. In our experiment, the number of solver iterations and CG iterations are set to 20 and 200 respectively. We set the timeout of the continuous solver to be 4000 seconds. As one can see from the table, our method is scalable on large scale design and can have about 7% improvement on average in both flows compared to the original inter-FPGA routing which has already been well optimized.

In the previous section, we introduce two discretization methods, which optimize total displacement and timing respectively. As shown in Tables 4 and 5, the TM flow has better average performance in runtime and system clock period although some cases have worse system clock period, which are caused by the inaccurate estimation of slack ratio. However, compared with the DISP flow, it has a much bigger displacement. One of the reasons that the DISP flow has worse system clock period is that the displacement-driven discretization does not consider timing. For example, given two solutions with the same displacement, it will not choose the one with better timing. In terms of runtime, the DISP flow is slower since its discretization explores a larger solution space compared to the one in the TM flow. However, we can easily trade off runtime and quality by changing the number of TDM ratio candidates.

4.3 Extension to Other Architectures

Although only connections between FPGA-pairs are considered in this work, it can be easily extended to architectures where FPGAs can communicate with each other through multiple intermediate hops. Since our optimization is performed after routing, how the fpga pair communicate is determined. Given this assumption, the timing graph can be built according to the routing topology and the wire limit constraints between hops and FPGAs can also be modeled similarly as those between FPGAs.

5 CONCLUSION

In this paper, we extend the TDM optimization problem of previous works to meet more general constraints in multi-FPGA based systems. We propose a novel method to optimize the system clock period of a multi-FPGA system. To be specific, we propose a two-step analytical framework that consists of a non-linear CG-based continuous solver and a DP-based discretization. Several techniques are used to improve the efficiency and effectiveness of the algorithms. Experimental results show that our approach is scalable and effective in large scale multi-FPGA based designs. About 7% improvement can be gained after our TDM optimization. Future works will include preconditioning in CG solver, considering discretization effect in the continuous solver, etc.

REFERENCES

[1] J. Babb, R. Tessier, M. Dahl, S. Z. Hanono, D. M. Hoki, and A. Agarwal. Logic emulation with virtual wires. *IEEE TCAD*, 16(6):609–626, 1997.

[2] S.-C. Chen, R. Sun, and Y.-W. Chang. Simultaneous partitioning and signals grouping for time-division multiplexing in 2.5 d fpga-based systems. In *Proc. ICCAD*, page 4, 2018.

[3] G. A. Constantinides. FPGAs in the cloud. In *Proc. FPGA*, pages 167–167, 2017.

[4] S. Hauck. The roles of fpgas in reprogrammable systems. *Proceedings of the IEEE*, 86(4):615–638, 1998.

[5] W. N. Hung and R. Sun. Challenges in large fpga-based logic emulation systems. In *Proc. ISPD*, pages 26–33, 2018.

[6] 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 Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 98(12):2572–2583, 2015.

[7] M. Inagi, Y. Takashima, and Y. Nakamura. Globally optimal time-multiplexing in inter-FPGA connections for accelerating multi-FPGA systems. In *Proc. FPL*, pages 212–217, 2009.

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

[9] W.-S. Kuo, S.-H. Zhang, W.-K. Mak, R. Sun, and Y. K. Leow. Pin assignment optimization for multi-2.5 d FPGA-based systems. In *Proc. ISPD*, pages 106–113, 2018.

[10] A. Ling and J. Anderson. The role of FPGAs in deep learning. In *Proc. FPGA*, pages 3–3, 2017.

[11] A. Makhorin. Glpk. *http://www.gnu.org/s/glpk/glpk.html*, 2008.

[12] W. Naylor and B. Chapman. WNLIB. http://www.willnaylor.com/wnlib.html.

[13] J. R. Shewchuk et al. An introduction to the conjugate gradient method without the agonizing pain, 1994.