Interconnection Allocation Between Functional Units and Registers in High-Level Synthesis

Cong Hao, Jianmo Ni, Nan Wang, and Takeshi Yoshimura, *Member, IEEE*

***Abstract*— Data path interconnection on VLSI chips usually consumes a significant amount of both power and area. In this paper, we focus on the port assignment problem for binary commutative operators for interconnection complexity reduction. First, the port assignment problem is formulated on a constraint graph, and a practical method is proposed to find a valid and initial solution. For solution optimization, an elementary spanning-tree-transformation-based local search algorithm is proposed. To improve the efficiency of optimization, a matrix formulation, which meets the simplex tabuleau format, is proposed and thus the simplex method is adopted for optimization. Moreover, operation pivoting and successive pivoting are discussed for algorithm speedup. The experimental results show that on the randomly generated test cases, the matrix-based algorithm shows the highest solution optimality and is five times faster than the elementary transformation method. On the real high-level synthesis benchmarks, the matrix-based method reduced 14% interconnections, while the previous greedy algorithm reduced 8% on average.**

***Index Terms*— Graphical methodology, high-level synthesis (HLS), interconnection, simplex method.**

I. Introduction

IN HIGH-LEVEL synthesis (HLS), interconnections have become one of the key features that influences the performance of integration designs. As shown in [4], interconnections consume a considerable fraction of total circuit power, and [5] shows that interconnections count for more than 50% of dynamic power of Intel microprocessors. Meanwhile, a multiplexer (MUX) is usually expensive in terms of both area and power consumption, especially the one with a large number of input ports. As pointed out in [6], the area and power consumption of a 32-to-1 MUX are almost equivalent to a 18-bit multiplier. Therefore, it is important to reduce the complexity of the interconnections that stitch the functional units (FUs) and registers, as well as MUX power and area.

The port assignment is an important step in connectivity binding in the HLS to reduce interconnection complexity. Given a fixed binding of FUs and registers, the interconnection is allocated from the registers to FUs through MUXes, which is called port assignment problem. It is first addressed in [7] and is proved to be NP-complete. When the operators are all binary commutative, it is more accurately defined as the port assignment problem for binary commutative operators (PAP-BCO) in [11]. Since the port assignment is performed on each FU after binding, its high-quality solutions are highly expected since they directly determine the interconnection complexity and MUX usage. Besides, if the port assignment can be combined into register binding step, it can help evaluate the solution quality of register binding in terms of interconnection complexity. Therefore, the execution time of port assignment algorithm must be short when it is combined into an iterative register binding algorithm [18] and being solved repeatedly. Consequently, producing optimal solutions and estimating interconnection complexity quickly and accurately are the essential demands for port assignment algorithms.

There are already several literature works that deal with the port assignment problem. The work in [8] performed global permutation of all the inputs of an FU during MUX generation, and the work in [9] designed an integer linear programming (ILP) algorithm; for both algorithms, the complexity is a concern. Chen *et al.* [10] proposed a greedy algorithm by swapping the operands of an operator if some operands are assigned to the registers that drive both ports. However, it has a limitation that operand swapping will not help when there is a series of operations that has circular dependencies. The work in [11] tried to decrease the MUX size discrepancy between the two ports. It reformulated the PAP-BCO problem as PAP-BCO*, which accounted for evenly distributing input signals among two MUXs, but it did not propose practical algorithms for PAP-BCO problem.

We also have published several works for port assignment problem [1]–[3]. Cong *et al.* [1] were the first to propose a practical spanning-tree-based algorithm, where elementary tree transformation is adopted for solution optimization. Cong *et al.* [2] extended the algorithm from [2] to also take MUX power into consideration. Cong *et al.* [3] still adopted the spanning tree for initial solution generation, but used a Fiduccia and Mattheyses (FM)-based method for optimization instead of elementary tree transformation. All these proposed algorithms excel other existing works [10], but they still have problems. For example, in [1] and [2], the elementary tree transformation used for solution optimization was time consuming, which degraded the algorithm efficiency. In [3], for

Manuscript received January 5, 2016; revised May 27, 2016 and August 8, 2016; accepted September 5, 2016. This work was supported by KAKENHI under Grant 26420323.

C. Hao and T. Yoshimura are with the Graduate School of Information, Production and System, Kitakyushu 808-0135, Japan (e-mail: cong.hao@akane.waseda.jp; tyoshimura@waseda.jp).

J. Ni is with the School of Shanghai Jiao Tong University, Shanghai 200240, China.

N. Wang is with the School of Information Science and Engineering, East China University of Science and Technology, Shanghai 200237, China.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2016.2607758

port assignment solution perturbation, the FM-based algorithm allowed only one vertex being moved or two vertices being swapped in each iteration of local search; thus the search area was small, which limited the solution optimality.

Therefore, in this paper, we improve [1] and [2] by proposing a new matrix-based algorithm and compare it with [3] and [10]. The contributions of this paper are stated as follows, among which 1) and 2) have been proposed in [1] and [2] and 3) to 5) are additional contributions.

1) A *spanning tree and conflict graph* based method is first proposed to obtain feasible solutions for the port assignment problem. It not only generates high-quality initial solutions but can also estimate interconnection complexity during register binding with little time cost.
2) An *elementary-tree-transformation*-based method is proposed for solution optimization. In each iteration, the spanning tree structure is changed by elementary tree transformation to get a smaller conflict graph.
3) To improve the efficiency of solution optimization, a *matrix formulation* is proposed by defining two operations $\oplus$ and $\otimes$; because the coefficient matrix agrees with the *simplex tableau* format, the *simplex method* is adopted and *pivotings* are performed for optimization.
4) To improve the efficiency of *pivot selection*, the properties of pivotings are studied and *successive pivotings* are proposed; three pivoting rules are proposed to filter unpromising or redundant pivoting calculations, which significantly speeds up the pivoting selection.
5) Our algorithm is tested on real benchmarks to evaluate the improvements in terms of power, area, and delay; moreover, the testbench FFT is implemented on an FPGA to get actual improvements of power, area, and clock frequency achieved by our algorithm.

The rest of this paper is organized as follows. Section II describes the problem formulation, and Section III introduces the initial solution generation. Section IV illustrates the elementary-tree-transformation-based solution optimization. In Section V, the matrix formulation is proposed and the simplex method is applied for optimization speedup. Section VI shows the experimental results and is followed by the conclusion and future work in Section VII.

II. Problem Formulation

After scheduling and FU and register binding in HLS, the final step is to connect allocated registers to FUs through MUXes. For each allocated FU, denoted by fu, a number of operations $\{op_1, op_2, \ldots, op_k\}$ are carried on it sequentially. The input operands of op_1–op_k are stored in the allocated registers, which are to be connected to fus input ports through MUXes. The step is called *port assignment*, and it is performed on one FU each time.

When fu is a binary commutative operator with two input ports, say "+" or "×," the problem is called the PAP-BCO. In this paper, we consider only the binary commutative FUs. Therefore, each fu has two ports and a register can be connected to the fu's left or right port or to both ports.

The objective of port assignment is to minimize the number of interconnections that connect the two ports of an FU to its input registers through MUXes. Naturally, the widths of two MUXes are also minimized.

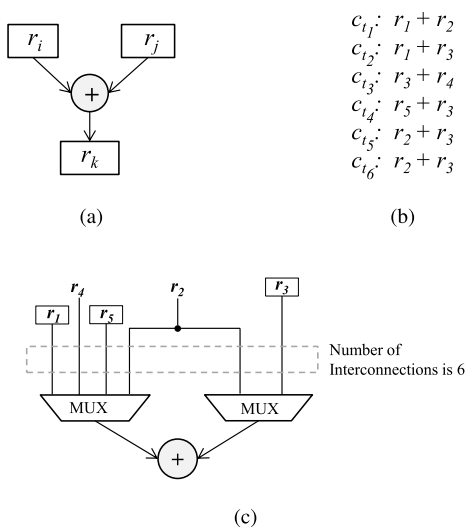


Fig. 1. Example of a binary commutative FU with the interconnection allocation between the registers and MUXes. (a) Example of an FU that carries "+" operation. (b) Operations executed in sequential control steps. (c) Interconnections allocated between registers and fu ports.

Fig. 1(a) and (b) shows an example of an FU that carries "+" operation with two input ports, and the sequential operations that are executed on it, respectively. In control step c_{t_1}, the operands stored in registers r_1 and r_2 are executed on the FU, and in c_{t_2}, the operands stored in r_1 and r_3 are executed. Fig. 1(c) shows the port assignment result for this example. In this case, the number of interconnections between the registers and the FU ports is 6, which is the objective to be minimized.

Besides, since wider MUXes have larger signal delays that may affect the critical path delay of a circuit, the size of the largest MUX in a circuit is also minimized as a secondary objective in this paper.

In the remaining of this section, first, the graph-based formulation is introduced and followed by an ILP formulation.

A. Graph-Based Formulation

In this paper, the port assignment problem is formulated on an undirected simple graph $G = (V, E)$, called the *constraint graph*. $V = \{v_1, v_2, \ldots, v_m\}$ is the set of registers that store input operands of fu and are to be connected to fu; $E = \{e_1, e_2, \ldots, e_n\}$, where each $e = (v_i, v_j) \in E$ means that registers r_i and r_j are providing two operands for fu in one operation.

To indicate the port which a register is connected to, the vertex set V is denoted by $V = V_L \cup V_R \cup V_B$ ($V_L \cap V_R = \varnothing$, $V_L \cap V_B = \varnothing$, $V_R \cap V_B = \varnothing$); a vertex $v_i \in V_L$ if its corresponding register r_i is connected to the fu's left port, $v_i \in V_R$ if r_i is connected to the fu's right port, or $v_i \in V_B$ if r_i is connected to both ports. Therefore, the port assignment problem is also regarded as a *vertex partition problem*, and hereafter, the vertex partition and port assignment are used indiscriminately.

For each edge $e = (v_i, v_j)$, their corresponding registers r_i and r_j shall be connected to different ports of fu separately,

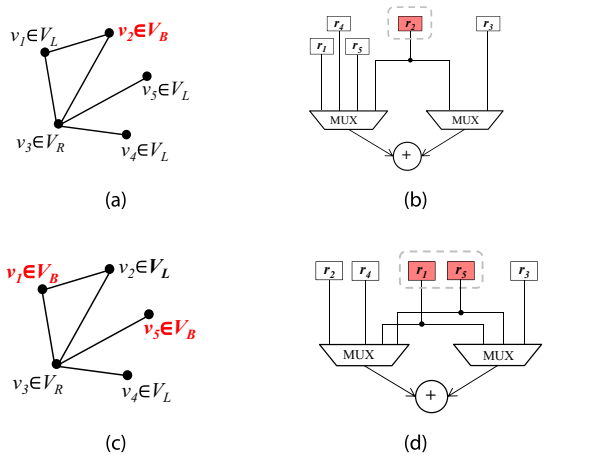


Fig. 2. Examples of vertex partitions and the corresponding port assignment solutions. (a) One valid vertex partition. (b) Port assignment from (a) with six interconnections. (c) Another valid vertex partition. (d) Port assignment from (c) with seven interconnections.

or at least one of them is connected to both ports, like r_2 shown in Fig. 1(c), to guarantee that the operands stored in r_i and r_j can be provided to fu in the same control step. Therefore, the vertex partition is valid if and only if when each edge $e = (v_i, v_j)$ satisfies the following *partition rules*:

$$v_i \in V_L, v_j \in V_R \tag{1}$$

$$v_i \in V_R, v_j \in V_L \tag{2}$$

$$v_i \in V_B \text{ or } v_j \in V_B. \tag{3}$$

As proved in [11], minimizing the interconnections between the fu and MUXes equals minimizing the number of registers that are connected to both ports, i.e., the value of $|V_B|$. For example, in Fig. 2(a) and (c), two different vertex partitions are shown, whose port connections are shown in Fig. 2(b) and (d). In Fig. 2(b) and (d), there is only one register r_2 being connected to both ports and there are two registers r_1 and r_5 being connected to both ports, respectively. The number of registers being connected to both ports is to be minimized.

Therefore, the port assignment problem is formulated as follows.

1) *Given:* An undirected simple graph $G = (V, E)$, where $v_i \in V$ stands for register r_i, which is to be connected to functional unit fu, and $e = (v_i, v_j) \in E$ stands for an execution of r_i and r_j in a particular control step.
2) *Goal:* To find a vertex partition on constraint graph G, where each $v \in V$ is partitioned to V_L, V_R, or V_B following the partition rules, to minimize $|V_B|$; after that, $max\{|V_L|, |V_R|\}$ is minimized as a secondary objective.

B. ILP Formulation

For each vertex $v_i \in V$, three 0–1 variables δ_l^i, δ_r^i, δ_b^i are defined to represent its partition, for example, if $v_i \in V_L$, then $\delta_1^i = 1$; otherwise, $\delta_1^i = 0$. They are stated as

$$\delta_l^i = 1 \text{ if } v_i \in V_L; \quad \delta_l^i = 0 \text{ if } v_i \notin V_L \tag{4}$$

$$\delta_l^i = 1 \text{ if } v_i \in V_R; \quad \delta_r^i = 0 \text{ if } v_i \notin V_R \tag{5}$$

$$\delta_b^i = 1 \text{ if } v_i \in V_B; \quad \delta_b^i = 0 \text{ if } v_i \notin V_B. \tag{6}$$

Therefore, the ILP formulation is written as

$$\text{Min: } \sum_{v_i \in V} \delta_b^i$$

$$\text{s.t. for } \forall v_i \in V : \delta_l^i, \delta_r^i, \delta_b^i \in \{0, 1\}$$

$$\text{for } \forall v_i \in V : \delta_l^i + \delta_r^i + \delta_b^i = 1$$

$$\text{for } \forall e = (v_i, v_j) \in E : \delta_l^i + \delta_l^j \le 1, \delta_r^i + \delta_r^j \le 1. \tag{7}$$

This ILP formulation can be solved using a general linear programming tool *lp_solver* [12].

In the following, first, an efficient algorithm is proposed to *generate feasible initial solutions* for the port assignment problem in Section III. Given an initial solution, an *elementary-tree-transformation-based optimization* is proposed to improve the solution quality, as discussed in Section IV. To improve the efficiency of tree-transformation-based optimization, and another optimization method, *the simplex-method-based optimization*, is proposed in Section V. The properties of pivotings and successive pivotings are studied as well to speed up the simplex-method-based optimization.

III. Conflict-Graph-Based Initial Solution Generation

In this section, a heuristic method is proposed to find a valid initial port assignment solution. First, vertices are initially partitioned by a spanning tree, which is introduced in Section III-A, and then the vertex partition is legalized by a *minimum-vertex-cover*-based method on a *conflict graph*, which is introduced in Section III-B.

A. Initial Vertex Partition on the Spanning Tree

We first give some graph notations from [14], which will be used in this paper.

1) A *spanning tree* of graph G is denoted by T. The *tree edges* on the spanning tree are denoted by e_t and the *nontree edge* are denoted by e_c.
2) A *fundamental cycle*, also called *fundamental loop*, is denoted by FL. Each fundamental loop FL has only one nontree edge e_c and each e_c has a unique fundamental loop, denoted by $FL(e_c)$.
3) A *fundamental cut set* is denoted by FC. Similarly, each tree edge e_t has a unique fundamental cut set, denoted by $FC(e_t)$.
4) The fundamental loop size refers to the number of edges in the loop. The *parity* of a nontree edge e_c, denoted by $p(e_c)$, is defined on the size of fundamental loop $FL(e_c)$: if the loop size is odd, the parity of e_c is *odd* and e_c is an *odd edge*; otherwise, the parity of e_c is *even* and e_c is an *even edge*.

On the given constraint graph $G = (V, E)$, a spanning tree T_0 rooted v_r is first built and E is written as $E = E_e \cup E_o \cup E_e$, where E_t is the set of tree edges, E_o is the set of odd nontree edges, and E_e is the set of even nontree edges. The *vertex level*, denoted by $lvl(v)$, is defined as the distance from vertex v to root v_r on the tree. Then the initial vertex partition is applied according to the vertex levels as

$$v \to V_L \text{ if } lvl(v) \text{ is even} \tag{8}$$

$$v \to V_R \text{ if } lvl(v) \text{ is odd.} \tag{9}$$

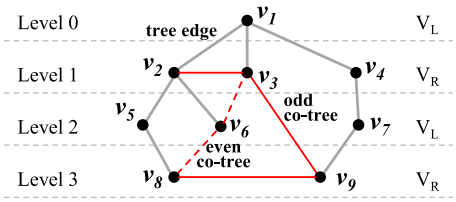


Fig. 3. Example of tree edges, odd nontree edges, and even nontree edges.

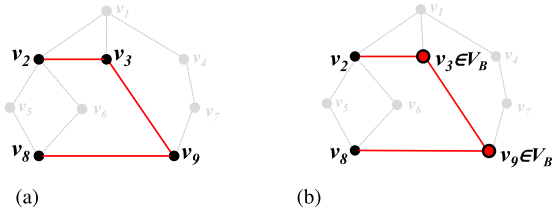


Fig. 4. Example of initial vertex partition legalization. (a) Conflict graph. (b) Vertex cover VC on the conflict graph.

Fig. 3 shows an example of initial vertex partition on a spanning tree. The gray edges are tree edges; the level of vertex v_1 is 0, the level of vertices v_2, v_3, and v_4 is 1, and so on. The vertices whose levels are 0 and 2 are initially partitioned to V_L, like v_1, v_5, v_6, and v_7, and the vertices whose levels are 1 and 3 are partitioned to V_R.

Given the initial partition, we have the following theorem.

Theorem 1: Each odd nontree edge violates the partition rules, while tree edges and even nontree edges do not.

The proof is straightforward and is omitted. For example, in Fig. 3, the odd nontree edge (v_3, v_9) violates the partition rules because both v_3 and v_9 are partitioned to set V_R, while the even nontree edge (v_6, v_8) does not.

B. Solution Legalization: Vertex Cover on Conflict Graph

The initial vertex partition is usually nonvalid because odd nontree edges violate the partition rules; these edges are defined as *conflict edges*. To record all conflict edges, a conflict graph composed of conflict edges is defined as follows.

Definition 1: A graph $G_c = (V_c, E_c)$ is defined as the *conflict graph*, where $e \in E_c$ is the set of conflict edges and $v \in V_c$ are the endpoints of conflict edges.

Fig. 4(a) shows the conflict graph built from Fig. 3, where the edges (v_2, v_3), (v_3, v_9), and (v_8, v_9) are conflict edges.

To legalize the initial vertex partition, some vertices on the conflict graph are needed to be repartitioned to to V_B, to make all conflict edges satisfy the partition rules. For each conflict edge $e_c = (v_i, v_j)$, if either v_i or v_j or both of them are repartitioned to V_B, edge e_c is legalized according to partition rule 3. On the other hand, since the objective of vertex partition is to minimize $|V_B|$, the number of vertices that are repartitioned to V_B is expected to be as small as possible.

Consequently, a *minimum vertex cover* is proposed to determine the vertices to be repartitioned on the conflict graph.

Definition 2: A *vertex cover* of a graph G is a set of vertices that each edge of the graph is adjacent to at least one vertex in the set. A *minimum vertex cover*, denoted by VC, is a vertex cover with the smallest number of vertices in it.

Theorem 2: Given a conflict graph G_c and its minimum vertex cover VC, the initial vertex partition is legalized by repartitioning all the vertices $v \in VC$ to V_B.

Proof: According to the definition of a vertex cover, each edge $e = (v_i, v_j)$ of G_c is adjacent to at least one vertex $v \in VC$ and $v \in V_B$; therefore, for each $e = (v_i, v_j)$, either v_i or v_j is in V_B, which no longer violates the partition rule. □

Though finding a minimum vertex cover is NP-hard, many high-quality algorithms have been proposed these years. Moreover, if minimum vertex cover is to be found on a graph that is a tree, it is reduced to a class-P problem and optimal solutions can be easily obtained. For the port assignment problem, from our pre-experiments, it shows that the conflict graphs are mostly sparse graphs and almost 90% of them are trees. Therefore, in most cases, optimal minimum vertex cover can be obtained efficiently. In this paper, a simple greedy heuristic in [13] is adopted to solve minimum vertex cover problem.

Fig. 4 shows an example of vertex partition legalization. Fig. 4(a) and (b) shows the conflict graph built from Fig. 3 and its minimum vertex cover including v_3 and v_9, respectively. By repartitioning v_3 and v_9 to V_B, the vertex partition is legalized.

In this section, the algorithm to produce a valid port assignment solution is proposed. As discussed in Section I, this is practical for interconnection complexity estimation during the register binding because of its high efficiency. Moreover, as to be shown in the experiments, the initial solutions have only 20% overhead compared with optimum solutions by ILP, which can be regarded as a good approximation for interconnections.

Given the NP-completeness of the port assignment problem, however, further optimizations are still needed to get near-optimal or optimal solutions. Our proposed solution optimizations are introduced in the following sections.

IV. Elementary-Tree-Transformation-Based Solution Optimization

In this section, the elementary-tree-transformation-based optimization is introduced. First, some definitions and the objective of solution optimization are given in Section IV-A. Then an *elementary-tree-transformation*-based local search method is discussed in Sections IV-B and IV-C. In addition, two important properties of elementary tree transformation to speed up the optimization are given.

A. Solution Optimization

Given a specific spanning tree T on the constraint graph, a unique conflict graph G_c is determined; then the minimum vertex cover is found on G_c and a valid vertex partition solution is obtained. Since most of the conflict graphs are trees or near trees, the chance to get an optimal minimum vertex cover is high, so that the vertex partition solution is predominantly determined by spanning tree. Therefore, changing spanning tree structure greatly affects solution quality.

On the other hand, calculating the minimum vertex cover every time is time consuming. Since empirically the minimum

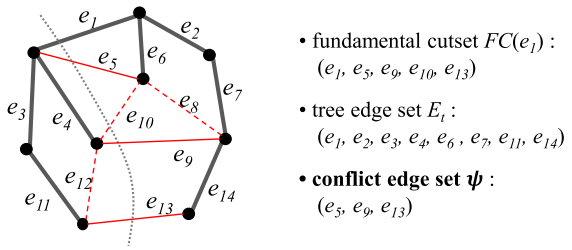


Fig. 5. Example of fundamental cut set FC, tree edge set E_t, and conflict edge set ψ.

vertex cover size is positively correlated with the number of conflict edges, the objective of solution optimization is to minimize the number of conflict edges for efficiency, as defined in the following.

Definition 3: A *conflict edge set*, denoted by ψ, is the set of all the conflict edges, i.e., the odd nontree edges. The number of edges in ψ is defined as the size of ψ, denoted by $\|\psi\|$.

Fig. 5 shows examples of fundamental cut set, tree edge set, and conflict edge set. The edges of the spanning tree are bold ones in the set E_t; the fundamental cut set of tree edge e_1 is $FC(e_1)$, including e_1, e_5, e_9, e_{10}, and e_{13}; and the conflict edge set ψ includes odd nontree edges e_5, e_9, and e_{13}, and $\|\psi\|$ is 3.

To sum up, the solution optimization is stated as follows: by iteratively changing the structure of the spanning tree on the constraint graph, the value of $\|\psi\|$ is minimized.

B. Elementary Tree Transformation

For solution optimization, a local-search-based algorithm is proposed; in each iteration, the structure of spanning tree is changed using *elementary tree transformation*. It is one of the most widely used transformations when computing minimum cost tree problems as explained in [17], which is defined in the following.

Definition 4: The *elementary transformation* of a spanning tree T refers to the process that a tree edge e_t is being removed out of T, resulting in the disjoint of two subtrees, and an edge $e_c \neq e_t$ is being added to into the tree so that the two subtrees are connected as a new spanning tree T' [13].

Accordingly, an elementary transformation can be uniquely defined by two specific edges, a tree e_t and a nontree edge in the fundamental cut set of e_t, i.e., $e_c \in FC(e_t)$; thus, an elementary transformation is denoted by $ET(e_t, e_c)$.

Specific to this problem, there are two properties of the elementary tree transformation, which can be used to speed up the local-search-based optimization by avoiding redundant calculations of $ET(e_t, e_c)$ or $FC(e_t)$.

Property 1: For an elementary transformation $ET(e_t, e_c)$, the conflict edge set ψ is updated as

$$\psi' = \begin{cases} \psi, & p(e_c) = 0 \\ \psi \triangle FC(e_t), & p(e_c) = 1 \end{cases} \tag{10}$$

where $p(e_c)$ is the parity of nontree edge e_c and the $\triangle$ is the *symmetric difference* between two sets defined as

$$A \triangle B = (A \cup B) - (A \cap B). \tag{11}$$

Algorithm 1 Local-Search-Based Port Assignment Algorithm

Require: Constraint graph $G = (V, E)$
Ensure: Find a valid vertex partitioning of each v
1: **while** stop criteria of random start NOT met **do**
2: Build an initial spanning tree T
3: Calculate the initial port assignment solution
4: **while** 1 **do**
5: **for** each tree edge $e_t \in E_t$ **do**
6: **for** each $e_c \in FC(e_t)$ **do**
7: $k \leftarrow$ the value of $\|\psi'\|$ computed from Eq.10
8: If k is small enough, keep (e_t, e_c) in set Cdt
9: **end for**
10: **end for**
11: **for** each candidate pair $(e_t, e_c) \in Cdt$ **do**
12: Perform transformation $ET(e_t, e_c)$ without updating T
13: Find minimum vertex cover on the new conflict graph
14: Keep the best solution as (e_{t_0}, e_{c_0})
15: **end for**
16: **if** current solution not improved **then** Break
17: Choose an odd non-tree $e_{c_0} \in FC(e_{t_0})$ randomly
18: Apply transformation $ET(e_c, e_{c_0})$, update tree T
19: **end while**
20: **end while**

Property 2: For an elementary transformation $ET(e_t, e_c)$, the fundamental cut set for each tree edge e_t is updated as

$$FC'(e_t) = \begin{cases} FC(e_t), & e_t \notin FL(e_c) \\ FC(e_t) \triangle FC(e_t), & e_t \in FL(e_c) \end{cases} \tag{12}$$

where $FL(e_c)$ is the fundamental loop of nontree edge e_c.

Property 1 shows that when performing an elementary tree transformation $ET(e_t, e_c)$, if the nontree edge e_c is an even edge, the conflict edge set ψ remains unchanged; therefore, these transformations shall be avoided. Property 2 shows that after performing an elementary tree transformation, the fundamental cut sets of tree edges can be calculated through set operations, and only a part of the fundamental cut sets are needed to be updated. The proofs are given in Section V through matrix formulations and operations.

C. Local-Search-Based Optimization Algorithm

Adopting the elementary tree transformation as local transformation, a local-search-based algorithm is proposed shown in Algorithm 1, including both initial solution generation and solution optimization.

First, the outermost cycle is the random start as shown in line 1. In each iteration of the random start, an initial spanning tree is first built on the constraint graph $G = (V, E)$ and an initial port assignment solution is obtained, shown in lines 2 and 3, and then, local-search-based solution optimization is performed shown in lines 4–20.

In each iteration of local search, first, α candidate edge pairs for elementary transformation are collected as shown in lines 5–9. For each tree edge e_t and all the nontree edges $e_c \in FC(e_t)$, the value of $\|\psi'\|$ is computed for each edge pair (e_t, e_c) according to (10), but the elementary transformation is not actually performed. Total α candidate edge pairs in set Cdt are kept with smallest $\|\psi'\|$.

Then as shown in lines 11–15, the elementary transformation is performed on each edge pair in Cdt, and the minimum vertex cover on the conflict graph is found. The best solution

(e_{t_0}, e_{c_0}) with the minimum $|V_B|$ is kept. The solution (e_{t_0}, e_{c_0}) is accepted if it improves the current global best solution; otherwise, the local search stops.

D. Critical Path Optimization

As described in Section II, critical path delay is the secondary objective function of the port assignment problem. Therefore, after $|V_B|$ being minimized, the size of the largest MUX among all MUXes is minimized, because among all control steps, the one with the largest MUX has a much higher possibility to be the critical path.

Critical path optimization can be performed by selecting from a group of port assignment solutions. During the optimization, many solutions may be found that have the same value of $|V_B|$ but different values of V_L and V_R; therefore, from the solutions that already have minimized $|V_B|$, the one with a minimized $max\{|V_L|,\ |V_R|\}$ is chosen.

V. Simplex-Method-Based Solution Optimization

In Section IV, the elementary-tree-transformation-based solution optimization is introduced. Although the two properties can speed up the algorithm, the tree transformation is still time consuming. To further improve the efficiency of solution optimization, in this section, a *new matrix formulation* is first proposed by defining two mathematic operations, $\oplus$ and $\otimes$, which is introduced in Section V-B. Since the coefficient matrix of our proposed formulation meets the *simplex tableau format*, the *simplex method* is adopted to efficiently perform the optimization, which is introduced in Section V-C. Moreover, to improve both efficiency and optimality, the properties of *pivotings and successive pivotings* specific to the port assignment problem are discussed. Based on these properties, a *pivoting selection* method and a *two-step pivoting selection* method are proposed, which significantly contribute to algorithm speedup and optimality improvement.

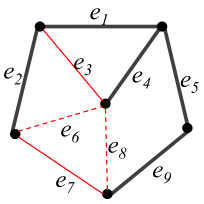
A. Preliminaries and Mathematic Formulation

1) Fundamental Cut Set and Edge Status Vector: In Section IV, the fundamental cut set FC, fundamental loop set FL, and conflict edge set ψ are defined. In this section, in order to organize the formulation into a matrix, *vectors* are proposed to instead *edge sets*. Without confusion, the same notations of FC, FL, and ψ are still adopted to represent edge vectors.

Definition 5: A *fundamental cut set vector* of tree edge e_t, denoted by $FC(e_t)$, is a 0–1 column vector with an entry for each edge on the constraint graph $G = (V, E)$: if edge e is in the fundamental cut set of e_t, its entry is 1; otherwise, it is 0.

Definition 6: An *edge status vector*, denoted by ψ, is also a 0-1 column vector with an entry for each edge: if edge e is a tree edge or even nontree edge, its entry is 0; if e is an odd nontree edge, i.e., the conflict edge, its entry is 1. The number of 1-entries is the *size* of ψ, denoted by $\|\psi\|$.

Fig. 6 shows examples of edge status vector ψ and fundamental cut set vectors $FC(e_1)$ and $FC(e_2)$.



Edge Status Vector ψ

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	$\|\|\psi\|\|$
ψ	0	0	1	0	0	0	1	0	0	2

Fundamental Cut Vector FC

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9
$FC(e_1)$	1	0	1	0	0	1	1	0	0
$FC(e_2)$	0	1	0	0	0	1	1	0	0
...					...	...			

Fig. 6. Examples of fundamental cut vector FC and edge status vector ψ.

2) Mathematic Formulation: To formulate the port assignment optimization mathematically, an exclusive-OR operation is first defined.

Definition 7: The $\oplus$ operation between two 0–1 values is defined as

$$x \oplus y = \begin{cases} 1, & x = 1, y = 0 \text{ or } x = 0, y = 1 \\ 0, & x = 0, y = 0 \text{ or } x = 1, y = 1. \end{cases} \tag{13}$$

Given an initial spanning tree and initial vertex partition on the constraint graph $G = (V, E)$, where each $v \in V$ has been assigned to V_L or V_R, the variables v and e are defined as

$$v = \begin{cases} 0, & v \in V_L \\ 1, & v \in V_R \end{cases} \tag{14}$$

$$e = \begin{cases} 0, & e \text{ is tree edge or even non-tree edge} \\ 1, & e \text{ is odd non-tree edge (conflict edge).} \end{cases} \tag{15}$$

The definition of e is consistent with the parity of non-tree edges, and the parity of tree edges are regarded as 0. In addition, it is consistent with the definition of the edge status vector ψ.

For each conflict edge where $e = 1$, it has

$$\text{for each } e_k = (v_i, v_j) \in E : v_i \oplus v_j \oplus e_k = 1. \tag{16}$$

If $\oplus$ operations are performed among the formulas in (16), where all edges exactly form a *loop* in the graph G, the variables v_i can be completely eliminated because $v_i \oplus v_i = 0$. Therefore, for each fundamental loop $FL(e_c)$, it has

$$\text{for each } FL(e_c) = \{e_{i_1}, e_{i_2} \ldots, e_{i_k}\} : \\ e_{i_1} \oplus e_{i_2} \oplus \ldots \oplus e_{i_k} = p(e_c). \tag{17}$$

Equation (17) shows the mathematic constraints of port assignment optimization; the objective is to minimize $\|\psi\|$. Fig. 7(a) shows an example. For fundamental loop $FL(e_3) = \{e_1, e_3, e_4\}$, it has a constraint of $e_1 \oplus e_3 \oplus e_4 = 1 = p(e_3)$, where the loop size of $FC(e_3)$ is 3 and the parity of e_3 is $p(e_3) = 1$.

B. Matrix Formulation

Definition 8: The $\oplus$ operation between two matrices A and B, where A and B are both $m \times n$ with only 0–1 entries, is defined as

$$(A \oplus B)_{ij} = A_{ij} \oplus B_{ij}. \tag{18}$$

Definition 9: The $\otimes$ operation between two matrices A and B, where A is a $n \times m$ matrix and B is a $m \times p$ matrix with 0–1 entries, is defined as the ordered multiplication and exclusive-OR operations as follows:

$$(A \otimes B)_{ij} = (A_{i1} \cdot B_{1j}) \oplus (A_{i2} \cdot B_{2j}) \oplus \cdots \oplus (A_{im} \cdot B_{mj}). \tag{19}$$

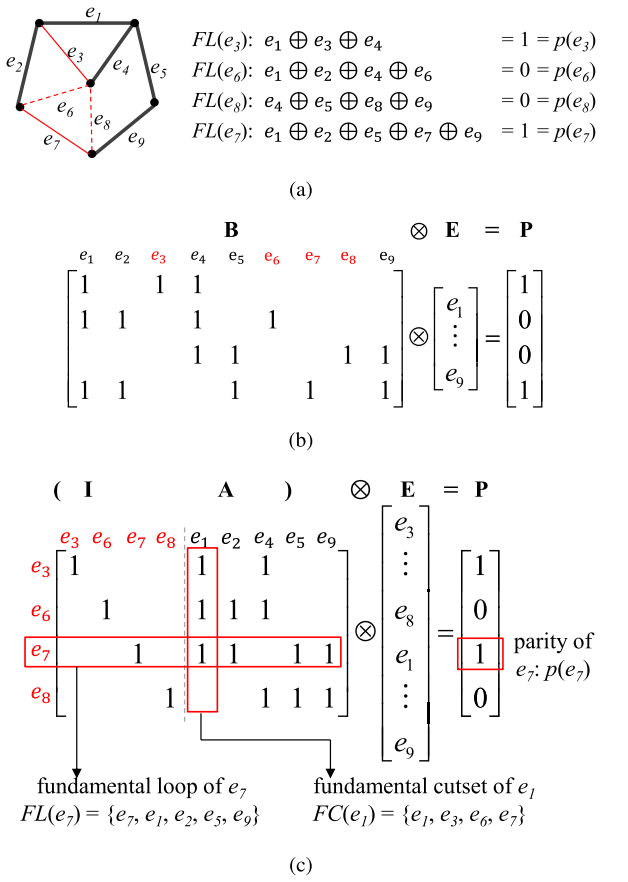


Fig. 7. Example of mathematic formulation and its matrix formulation. (a) Tree and nontree edges are formulated by ⊕ operations. (b) Write into matrix format. (c) Write into FCM format (simplex tableau format).

According to the theory of fundamental cycle matrix (FCM) theory [14], we can write the formulas in (17) into the following equation:

$$B \otimes E = P \tag{20}$$

where B is the coefficient matrix, E is the variable matrix of e_i, and P is the parity matrix of $p(e_i)$. Fig. 7(b) shows the matrix formula organized from Fig. 7(a) written as (20).

The coefficient matrix B is an FCM, which can be organized as the combination of a $n_c \times n_c$ square identity matrix I and a $n_c \times n_t$ matrix A, and is written as

$$B = (I \ A) = (b^1 \ \cdots \ b^p \ \cdots \ b^{n_c} \ b^{n_c+1} \ \cdots \ b^q \ \cdots \ b^n)$$

$$= \begin{array}{c} \\ e_{c_1} \\ \vdots \\ \boldsymbol{e_{c_p}} \\ \vdots \\ e_{c_{n_c}} \end{array} \begin{array}{c} \begin{array}{cccccccccc} e_{c_1} & \cdots & \boldsymbol{e_{c_p}} & \cdots & e_{c_{n_c}} & e_{t_1} & \cdots & \boldsymbol{e_{t_q}} & \cdots & e_{t_{n_t}} \end{array} \\ \begin{bmatrix} 1 & & & & & b_{11} & & b_{1q} & & b_{1n_t} \\ & \ddots & & & & \vdots & & \vdots & & \vdots \\ & & 1 & & & b_{p1} & & b_{pq} & & b_{pn_t} \\ & & & \ddots & & \vdots & & \vdots & & \vdots \\ & & & & 1 & b_{n_c1} & & b_{n_cq} & & b_{n_cn_t} \end{bmatrix} \end{array} \tag{21}$$

where the identity matrix I represents the coefficients of the nontree edge variables, i.e., e_{c_1}–$e_{c_{n_c}}$, and the matrix A represents the coefficients of the tree edge variables, i.e., e_{t_1}–$e_{t_{n_t}}$. The column vector $b^i (n_c + 1 \le i \le n)$ represents the fundamental cut set of the tree edge e_i, and the row vector $r^j (1 \le j \le n_c)$ represents the fundamental loop of the nontree edge e_i.

Letting E_c represent the nontree edge variables and E_t represent the tree edge variables, E is written as $E = (E_c \ E_t)^T$ and (20) is written as

$$[I \quad A] \otimes \begin{bmatrix} E_c \\ E_t \end{bmatrix} = P. \tag{22}$$

Fig. 7(c) shows the matrix format of (22), where the coefficient matrix B is organized as an FCM.

Therefore, the matrix formulation of port assignment optimization is stated as

$$\begin{aligned} &\text{Min: } \|\psi\| = \sum_{e_i \in E_c} p(e_i) \\ &\text{s.t.: } B \otimes E = P. \end{aligned} \tag{23}$$

C. Simplex Tableau Format

The formulation shown in (23) agrees with the linear programming problem (LPP) standard form because of the following reasons.

1) All variables are non-negative, i.e., all 0–1 variables.
2) The coefficient matrix $B = (I \ A)$ agrees with the simplex tableau format, where the matrix I is an identity submatrix, whose variables are *basic variables*, and the variables in A are *nonbasic variables.*
3) When all nonbasic variables are 0, the right-hand values are determined by basic variables: in this formulation, the nonbasic variables are tree edges and the basic variables are nontree edges; the edge parities, i.e., the right-hand values, are determined by the ⊕ operations of all nontree variables.

The differences between our formulation and LPP are as follows.

1) Our objective function is not a linear combination of all variables while the LPP is.
2) Our formulation is composed of ⊕, ⊗, and multiplications, while LPP is composed of additions, substractions, and multiplications.

Therefore, our formulation is not guaranteed to get optimum solutions as the linear programming using the simplex method. However, since it agrees the LPP form, the similar *pivoting* of the simplex method can be adopted.

D. Pivoting

The general *pivoting* of the simplex method refers to the process of improving the current solution (if is not optimal) by moving a nonbasic variable (entering variable) into the basis and moving a basic variable (leaving variable) out of the basis. Similarly, the *pivoting* of our matrix formulation is defined as follows.

Definition 10: A *pivoting* refers to the process that moving a nontree edge variable e_{c_p} out of the basis and moving a tree edge variable e_{t_q} into the basis, where $b_{pq} = 1$ (b_{pq} is the

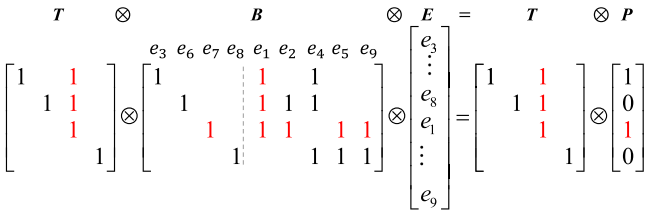


Fig. 8. Example of the pivoting using pivoting matrix T.

entry of the coefficient matrix B). A pivoting between leaving variable e_c and entering variable e_t is denoted as $PV(e_c, e_t)$.

The definition specifies that the tree edge e_{t_q} must be in the fundamental loop of the nontree edge e_{c_p} or the nontree edge e_{c_p} must be in the fundamental cut set of the tree edge e_{t_q}, which is equivalent to an elementary tree transformation.

To perform a pivoting, a *pivoting matrix* T is introduced as

$$T \otimes B \otimes E = T \otimes P. \tag{24}$$

T is a $n_c \times n_c$ square matrix written as

$$T = \begin{bmatrix} & t^1 & \cdots & & t^p & & \cdots & t^{n_c} \\ & 1 & & & b_{1q} & & & \\ & & \ddots & & \vdots & & & \\ & & & 1 & \vdots & & & \\ & & & & b_{pq} & & & \\ & & & & \vdots & 1 & & \\ & & & & \vdots & & \ddots & \\ & & & & b_{n_c q} & & & 1 \end{bmatrix} \tag{25}$$

where its pth column is copied from the qth column of matrix B, i.e., $t^p = b^q$.

The matrix B' after the pivoting is calculated as

$$B' = T \otimes B = (b'^1 \ \cdots \ b'^p \ \cdots \ b'^q \ \cdots \ b'^n)^T \tag{26}$$

where each new column vector b'^i is calculated as

$$b'^i = \begin{cases} b^i, & b_{pi} = 0 \\ b^i \oplus t^q = b^i \oplus b^q, & b_{pi} = 1. \end{cases} \tag{27}$$

And the matrix P' after the pivoting is calculated as

$$P' = T \otimes P = \begin{cases} P, & p_p = 0 \\ P \oplus b^q, & p_p = 1. \end{cases} \tag{28}$$

This calculation can be easily proved given the definition of $\oplus$ and $\otimes$ operations, so the detailed steps are omitted.

Fig. 8 shows an example of a pivoting $PV(e_3, e_5)$ using a pivoting matrix T. In this example, the entering variable is e_5 and the leaving variable is e_3.

According the definition of v and e in (14) and (15), the definition of the fundamental cut set vector exactly agrees with the column vector b^i and the definition of edge status vector ψ also agrees with the matrix P. Therefore, from (27) and (28), we have

$$FC'(e_i) = \begin{cases} FC(e_i), & e_i \in FL(e_{c_p}) \\ FC(e_i) \ \triangle \ FC(e_{t_q}), & e_i \notin FL(e_{c_p}). \end{cases} \tag{29}$$

$$\psi' = \begin{cases} \psi, & p(e_{c_p}) = 0 \\ \psi \ \triangle \ FC(e_{t_q}), & p(e_{c_p}) = 1. \end{cases} \tag{30}$$

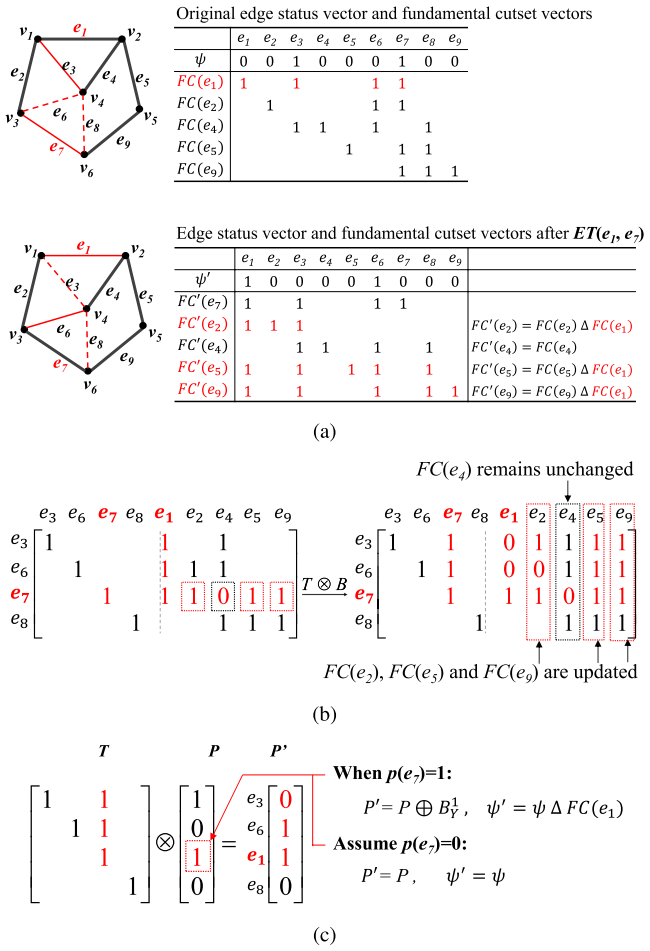


Fig. 9. Example of the pivoting using pivoting matrix T. (a) Tree transformation on the constraint graph. (b) Calculation of coefficient matrix B' from B. (c) Calculation of parity matrix P' from P.

The above equations (29) and (30) are exactly the same as the elementary tree transformation properties discussed in Section IV-B and the examples are shown in Fig. 9. Fig. 9(a) shows the example of the elementary transformation between edges e_1 and e_7, and Fig. 9(b) and (c) shows the corresponding matrix calculations.

Fig. 9(b) illustrates (29) and shows that if a tree edge e_i is not in the fundamental loop of the leaving nontree edge e_{c_p}, its fundamental cut set keeps unchanged, say edge e_4; otherwise, the fundamental cut set is updated using $\triangle$ operations between two vectors, as the columns for edges e_2, e_5, and e_9.

Fig. 9(c) illustrates (30) and shows how the edge status vector ψ changes. When the chosen leaving edge e_3 is an odd edge, i.e., $p(e_3) = 1$, the ψ is updated through $\triangle$ operation; otherwise, if $p(e_3) = 0$, both the P and ψ remain unchanged.

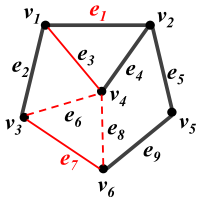
Naturally, derived from (29) and (30), we have the following definitions and theorems.

Definition 11: For pivoting $PV(e_c, e_t)$, if $p(e_c) = 1$, it is called as an *effective pivoting*; otherwise, it is called an *ineffective pivoting*. We ignore all the ineffective pivots, and hereafter, all the pivotings we mention are effective ones.

Theorem 3: For pivotings $PV_1(e_{c_1}, e_t)$ and $PV_2(e_{c_2}, e_t)$ with the same entering variable e_t, $\psi'_1 = \psi'_2$; the

Algorithm 2 General Pivot Selection

Ensure: Initial matrix formulation $B_0 \otimes E_0 = P_0$
Require: Pivoting $PV(e_x, e_y)$ to be performed
1: Build initial edge status vector ψ and FC vectors
2: Calculate Edge Status Table for each tree edge e_t
3: Choose $PV(x, e_i)$ with the smallest $\|\psi'\|$ in Edge Status Table, $e_y = e_i$
4: Choose $e_j \in FC(e_i)$ randomly, $e_x = e_j$



Edge Status Table – Level 1

ψ_1	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	$\|\psi_1\|$
$PV(x,e_1)$	1					1				2
$PV(x,e_2)$		1	1			1				3
$PV(x,e_4)$				1		1	1	1		4
$PV(x,e_5)$			1		1			1		3
$PV(x,e_9)$			1					1	1	3

Fig. 10. Pivot selection using an edge status table.

two pivotings are regarded as *equivalent* in terms of the edge status vector ψ'.

From Theorem 3, we know that for a specified entering variable e_t, all the pivotings $PV(e_c, e_t)$, where $e_c \in FC(e_t)$, are equivalent and undistinguished; so we denote a pivoting as $PV(x, e_t)$ if we do not care the leaving variables for an entering variable e_t.

E. Pivot Selection

Pivot selection is a crucial step in the simplex method since good choices can lead to a significant speed up in finding the optimal solutions. However, in our formulation, a pivoting is not guaranteed to always improve the solution, so that in order to make a pivot selection that results in the smallest number of odd nontree edges, all potential pivotings are needed to be checked, which is obviously time consuming.

Note that each pivoting $T \otimes B \otimes E = T \otimes P$ consists of two steps: the calculation of coefficient matrix $B' = T \otimes B$ and parity matrix $P' = T \otimes P$. Obviously, the size of matrix B is much larger than P and the calculation of B' is more time consuming than P'. Therefore, it is expected that P' can be calculated first without the calculating B'.

Based on this motivation, an *edge status table* is proposed, where each row of the table is an edge status vector obtained after one pivoting $PV(x, e_{t_i})$. Fig. 10 shows an example of the edge status table, which shows the possible pivotings for all the tree edges. Then a pivoting selection method based on the edge status table is proposed shown in Algorithm 2. Before each pivoting, the edge status table is first calculated. For the entering variable, e_i of pivoting $PV(x, e_i)$, which has the smallest $\|\psi\|$, is chosen; for the leaving variable, one variable $e_j \in FC(e_i)$ is randomly chosen.

F. Successive Pivotings

Definition 12: Two adjacent pivotings, for example, pivoting PV_1 is performed and is followed by pivoting PV_2, are regarded as *successive pivotings*, denoted by $PV_1 + PV_2$.

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	$\|\psi\|$
$PV_1(e_1, e_7)+PV_2(e_7, e_3)$	1					1				2
$PV(e_1, e_3)$	1					1				2
$PV_1(e_1, e_3)+PV_2(e_6, e_4)$				1		1	1	1		4
$PV(x, e_4)$				1		1	1	1		4

Fig. 11. Examples of redundant successive pivotings.

The original edge status vector is denoted by ψ, the one after PV_1 is ψ_1, and the one after PV_2 is ψ_2.

When applying Algorithm 2 for optimization, in each iteration, all possible pivotings are calculated; however, there are some redundant calculation between two successive pivotings.

1) The result of two successive pivotings may be the same as before pivoting (see Fig. 11), the edge status vector ψ after $PV(e_1, e_3)$ is the same as the vector after two successive pivotings $PV_1(e_1, e_7) + PV_2(e_7, e_3)$.
2) The result of two successive pivotings may have already been calculated: for example, ψ after $PV_1(e_1, e_3) + PV_2(e_6, e_4)$ is the same as $PV(x, e_4)$.

Therefore, for two successive pivotings, the following two theorems are proposed to avoid redundant pivoting calculations in the early stage, before the first pivoting being performed.

Theorem 4: For two successive pivotings $PV_1(e_x, e_y)$ followed by $PV_2(e_y, e_z)$, they have:

$$PV_1(e_x, e_y) + PV_2(e_y, e_z) = PV(e_x, e_z). \tag{31}$$

This theorem can be easily explained by tree transformation so that the proof is omitted.

Theorem 5: For successive pivotings $PV_1(e_{p_1}, e_{q_1})$ followed by $PV_2(e_{p_2}, e_{q_2})$, where $p(e_{p_1}) = 1$ and $p(e_{p2}) \oplus b_{p_2q_1} = 1$, they have the following conditions.

1) When $b_{p_1q_2} = 0$ and $b_{p_2q_2} = 1$

$$\psi_2 = \psi \triangle FC(e_{q_1}) \triangle FC(e_{q_2}). \tag{32}$$

2) When $b_{p_1q_2} = 1$ and $b_{p_2q_2} \oplus b_{p_2q_1} = 1$

$$\psi_2 = \psi \triangle FC(e_{q_2}). \tag{33}$$

3) *Others:* Successive pivotings cannot be applied.

Proof: First, the conditions $p(e_{p_1}) = 1$ and $p(e_{p2}) \oplus b_{p_2q_1} = 1$ are to guarantee that both PV_1 and PV_2 are effective pivotings, which are the same as the conditions in (30).

Second, the condition that pivoting PV_2 can be performed is that $b'_{p_2q_2} = 1$, which refers to the entry of $b_{p_2q_2}$ after PV_1. When $b'_{p_2q_2} = 1$ is ensured, we have

$$\psi_1 = \psi \triangle FC(e_{q_1}) \tag{34}$$

$$\psi_2 = \psi_1 \triangle FC'(e_{q_2}) \tag{35}$$

$$FC'(e_{q_2}) = \begin{cases} FC(e_{q_2}) \triangle FC(e_{q_1}), & b_{p_1q_2} = 1 \\ FC(e_{q_2}), & b_{p_1q_2} = 0. \end{cases} \tag{36}$$

$$\therefore \psi_2 = \begin{cases} \psi \triangle FC(e_{q_2}) \triangle FC(e_{q_1}), & b_{p_1q_2} = 1 \\ \psi \triangle FC(e_{q_2}), & b_{p_1q_2} = 0. \end{cases} \tag{37}$$

Since $b'_{p_2q_2}$ is calculated as

$$b'_{p_2q_2} = \begin{cases} b_{p_2q_2}, & b_{p_1q_2} = 0 \\ b_{p_2q_2} \oplus b_{p_1q_2}, & b_{p_1q_2} = 1 \end{cases} \tag{38}$$

Edge Status Table – Level 2

PV_1	PV_2	ψ_2	x	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	
$PV_1(x,e_1)$	$PV_2(y,e_2)$	$\psi_1 \triangle FC(e_2)$	e_3	1	1					1			Need calculation
	$y \neq e_1$	$\psi \triangle FC(e_2)$	e_7		1	1			1				$= PV_1(x,e_2)$
	$PV_2(y,e_4)$	$\psi_1 \triangle FC(e_4)$	e_7			1				1			Need calculation
	$y \neq e_1$	$\psi \triangle FC(e_4)$	e_3				1		1	1	1		$= PV_1(x,e_4)$
	$PV_2(y,e_5)$	$\psi_1 \triangle FC(e_5)$	e_3	1					1				Need calculation
	$y \neq e_1$	$\psi \triangle FC(e_5)$	e_7			1		1			1		$= PV_1(x,e_5)$
	$PV_2(y,e_9)$	$\psi_1 \triangle FC(e_9)$	e_3	1					1	1	1	1	Need calculation
	$y \neq e_1$	$\psi \triangle FC(e_9)$	e_7			1					1	1	$= PV_1(x,e_9)$
$PV_1(x,e_2)$	$PV_2(y,e_4)$	$\psi_1 \triangle FC(e_4)$	e_7		1		1				1		Need calculation
	$y \neq e_2$	–	–										
	$PV_2(y,e_5)$	–	–										
	$y \neq e_2$	$\psi_1 \triangle FC(e_5)$	e_7			1		1			1		$= PV_1(x,e_5)$
	$PV_2(y,e_9)$	–	–										
	$y \neq e_2$	$\psi \triangle FC(e_9)$	e_7			1					1	1	$= PV_1(x,e_9)$
.....													

Fig. 12. Example of the edge status table of level 2.

to ensure $b'_{p_2q_2} = 1$, we shall have $b_{p_1q_2} = 0$ and $b_{p_2q_2} = 1$, which is the condition for (32), or we shall have $b_{p_1q_2} = 1$ and $b_{p_2q_2} \oplus b_{p_1q_2} = 1$, which is the condition for (33). □

From (32) and (33) in Theorem 5, it shows that when performing two successive pivotings PV_1 and PV_2, the edge status vector ψ_2 of the PV_2 can be directly calculated using the original fundamental cut set vectors like $FC(e_{q_1})$ and $FC(e_{q_2})$. It means the result of two successive pivotings $PV_1 + PV_2$ can be calculated without actually performing pivoting PV_1. Besides, (33) shows that in some cases, the successive pivotings $PV_1 + PV_2$ may get the same result as PV_1, and the calculations for these cases can be avoided.

Based on Theorems 4 and 5, an *edge status table of level* 2 is proposed to calculate the results of the successive pivotings based on the edge status table of level 1. Fig. 12 shows an example of edge status table of level 2. Given the edge status table of level 1, the successive pivotings of each two tree edges e_{q_1} and e_{q_2} (order is neglected) are calculated, denoted by $PV_1(e_{p_1}, e_{q_1})$ and $PV_2(e_{p_2}, e_{q_2})$. For each PV_1, according to (32) and (33), there are two possible solutions: if $b_{p_1q_2} = 1$, like shown in the first row where e_{p_1} is e_3, the ψ_2 is calculated as $\psi_2 = \psi \triangle FC(e_1) \triangle FC(e_2)$ according to (32), while if $b_{p_1q_2} = 0$, like shown in the second row where e_{p_1} is e_7, ψ_2 is calculated as $\psi_2 = \psi \triangle FC(e_2)$ according to (33). Moreover, in the second situation, the result of ψ_2 has already been calculated in the edge status table of level 1, so that no calculation is needed. It can be seen from the table shown in Fig. 12 that for all possible successive pivotings, only a portion of them need calculation and most successive pivotings can be skipped. Therefore, the pivot selection efficiency is expected to be greatly enhanced.

G. Two-Step Pivot Selection

Based on the edge status tables, a two-step pivoting selection method is proposed, as shown in Algorithm 3.

The major improvements from Algorithm 2 include the following.

1) Determining two successive pivotings in one iteration is more efficient than that in Algorithm 2, because the execution of a pivoting is time consuming due to the large size of coefficient matrix B.
2) It improves the optimality of the pivoting selections, because the selection of PV_1 may be blind if the results of PV_2 are unpredictable.

Algorithm 3 Two-Step Pivot Selection

Ensure: Initial matrix formulation $B_0 \otimes E_0 = P_0$
Require: Pivotings PV_1 and PV_2 to be performed
1: Build initial edge status vector ψ
2: **Step 1:** Calculate Edge Status Table of Level 1
3: **Step 2:** Calculate Edge Status Table of Level 2
4: Choose k successive pivotings with the smallest $\|\psi_2\|$ in Edge Status Table of Level 2, solve the exact minimum vertex cover
5: Perform pivoting PV_1 and pivoting PV_2 with the smallest vertex cover solution

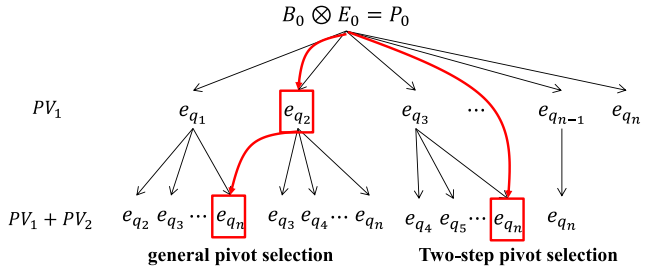


Fig. 13. Two-step pivot selection.

Fig. 13 shows the major differences of the two pivot selection methods. Starting from a matrix formula $B_o \otimes E_0 = P_0$, the two-step pivoting selection is able to forecast the results of $PV_1 + PV_2$ and skip the calculation of PV_1 and choose a best solution of $PV_1 + PV_2$.

H. Multistep Pivotings

In Section V-F, two-step successive pivoting is discussed; in this section, multistep successive pivotings are discussed.

For successive pivotings $PV_1(e_{p_1}, e_{q_1})$, $PV_2(e_{p_2}, e_{q_2})$, and $PV_3(e_{p_3}, e_{q_3})$ with edge status vectors ψ (before PV_1), we have ψ_1, ψ_2, and ψ_3. Similar to (32) and (33), ψ_3 can be directly computed from ψ and $FC(e_{q_1})$–$FC(e_{q_3})$ under some special conditions as

$$\psi_3 = \begin{cases} \psi \triangle FC(e_{q_3}) \triangle FC(e_{q_1}) \triangle FC(e_{q_2}), & cond.\ 1 \\ \psi \triangle FC(e_{q_3}) \triangle FC(e_{q_2}), & cond.\ 2 \\ \psi \triangle FC(e_{q_3}) \triangle FC(e_{q_1}), & cond.\ 3 \\ \psi \triangle FC(e_{q_3}), & cond.\ 4 \\ \text{succesive pivotings cannot be performed}, & others \end{cases} \tag{39}$$

where $cond.\ 1$ is

$$b_{p_1q_2} = 0, \quad b_{p_1q_3} = 0, \ b_{p_2q_3} = 0, \ b_{p_2q_2} = 1, \ b_{p_3q_3} = 1 \tag{40}$$

and conditions $cond.\ 2$–$cond.\ 4$ are more complicated. Besides, the conditions to guarantee PV_1–PV_3 are all effective pivotings and similar to the ones in Theorem 5 are also needed. If all conditions are not met, successive pivotings cannot be applied. The proof is omitted due to page limit.

Knowing from (39) and (40), first, the conditions for three-step successive pivoting is much more strict than for two-step pivoting as in (40), so that the situations that three-step pivotings cannot be applied are much more than two-step pivotings. Second, the condition check, i.e., the calculation of $cond.1$–$cond.4$ is also much more time consuming than two-step pivotings. Consequently, we believe that multistep pivotings do not contribute to optimality more than two-step pivoting and the time consumption will be much larger.

VI. Experimental Results

The proposed algorithms are implemented in C on the platform of Windows 7 with a 2.8-GHz CPU and a 8-GB memory. The algorithms are evaluated on randomly generated constraint graphs and real HLS benchmarks from [18].

A. Optimality and Runtime Test on Random Constraint Graph

In Table III, our proposed algorithms are first evaluated on the randomly generated constraint graphs with the different number of vertices and graph densities. The column Vtx. shows the number of vertices of the constraint graph from 40 to 600 and the column Den. shows the graph density varying from 2.0 to 5.0. For each pair of Vtx. and Den., one graph is randomly generated as the constraint graph. On each constraint graph, our algorithm is executed for 1000 times and the number of interconnections is the average value of 1000 runs. The # of INT shows the total number of interconnections. The columns of Cmp refer to the comparisons.

Our algorithms are compared with the optimum solutions generated by ILP, shown in the column of ILP, solved by *lp_solver* [12], including the following.

1) The initial solutions, obtained by conflict graph and minimum vertex cover discussed in Section III, are shown in the column of Initial Solution.
2) The elementary-tree-transformation-based algorithm, Algorithm 1, discussed in Section IV, is shown in the column of Tree Based Opt.
3) The matrix-based optimization discussed in Section V, where the general pivoting selection of Algorithm 2 is adopted, is shown in the column of Marix Based Opt. I.
4) The matrix-based optimization, where the two-step pivoting selection of Algorithm 3 is adopted, is shown in the column of Matrix Based Opt. II.

In terms of solution optimality, i.e., the number of interconnects, compared with ILP, it shows that the initial solutions are 1.221, the tree-based optimization solutions are 1.069, the matrix-based optimization solutions using general pivot selection are 1.045, and the solutions using two-step pivot selection are 1.032 times overhead, respectively. The matrix-based optimization using two-step pivot selection shows the highest optimality, because it is able to predict two successive pivotings in one iteration.

In terms of algorithm efficiency, the execution times of three optimization algorithms are shown in milliseconds. The matrix-based optimization using general pivoting is taken as the baseline for comparison since it shows the shortest execution time. The elementary-transformation-based optimization runs 5.7 times slower, because the tree transformations consume most of the execution time. The two-step pivoting optimization has a slight overhead compared with the general pivoting optimization, 1.16 on average. It is mainly because of the condition checks in (32) and (33). Even then, compared with the elementary-tree transformation-based optimization, the matrix-based optimization still shows 4.9 times speedup.

TABLE I
INFORMATION OF TEST BENCHES

DFG	V	E	T_{lp}*	DFG	V	E	T_{lp}*
arf	29	42	11	rand0	92	136	17
ewf	35	55	17	rand1	158	234	19
ar	29	42	11	rand2	632	943	30
ae	54	143	60	rand3	658	1392	42
ad2	47	110	47	rand4	917	2601	189
ellip	35	67	25	rand5	1916	5387	251
mpeg	54	114	36	rand6	1832	5428	583
fft	134	234	20				

* The longest path in DFG

TABLE II
AREA, POWER, AND DELAY OF MUXs FROM [20]

MUX	Power(μW)	Area(μm^2)	Delay(ns)
2 to 1	27.58	130	0.81
4 to 1	44.24	317	1.05
8 to 1	78.82	748	1.40
16 to 1	138.99	1315	1.74
32 to 1	257.82	2464	2.08

B. Optimality and Runtime Test on DFGs

Our port assignment algorithms are also applied on real HLS benchmarks, by comparing with [3] and [10], shown in Table IV. The benchmarks are given in the format of data flow graphs (DFGs) as shown in Table I. The first eight benchmarks are taken from [18], and benchmarks from rand0 to rand6 are randomly generated using a DFG generator $TGFF$ [19]. Each benchmark is given the number of vertices in Column V and the number of edges in Column E, and the length of longest path in DFG is shown in Column T_{lp}. The scheduling, FU, and register binding are first performed using the method in [18]; the number of interconnections is shown in Column INT with runtime measured in milliseconds. Then port assignment is performed to minimize the number of interconnections. The greedy [10] shows the results of the algorithm proposed in [10]; FM [3] shows the results of the FM-based algorithm proposed in [3]. Tree Based Opt. shows the results of the algorithm proposed in [1] and [2], in which the elementary tree transformation is adopted for optimization. Matrix Based Opt. I and Matrix Based Opt. II show the results of our matrix-based algorithm using pivotings, where Opt. I uses the general pivot selection shown in Algorithm 2 and Opt. II uses the two-step pivot selection shown in Algorithm 3.

It shows that the greedy algorithm in [10] reduces the number of interconnections by 7.7%, with a time overhead of 0.34%; it shows high efficiency because of the algorithm simplicity. The FM-based algorithm [3] reduces the number

TABLE III

COMPARISON BETWEEN ELEMENTARY-TREE-TRANSFORMATION-BASED AND MATRIX-TRANSFORMATION-BASED OPTIMIZATION METHODS

TestBench		**ILP**		**Initial Solution**			**Tree Based Opt.**				**Matrix Based Opt. I**[1]				**Matrix Based Opt. II**[2]			
Vtx.	Den.	# of INT	Cmp	# of INT	Cmp	Time (ms)	# of INT	Cmp	Time	Cmp	# of INT	Cmp	Time	Cmp	# of INT	Cmp	Time	Cmp
40	2.0	9	1	10.77	1.197	0.027	9.01	1.001	9.281	4.57	9.054	1.006	2.03	1	9.01	1.001	2.55	1.26
40	5.0	18	1	20.69	1.149	0.011	18.19	1.011	46.81	6.55	18.03	1.002	7.15	1	18.02	1.001	7.98	1.12
50	2.5	12	1	15.10	1.258	0.050	12.16	1.013	24.32	2.48	12.14	1.012	9.82	1	12.04	1.003	8.65	0.88
50	3.0	16	1	18.42	1.151	0.063	16.04	1.003	32.06	3.87	16.04	1.002	8.28	1	16.02	1.001	8.72	1.05
50	4.0	18	1	21.94	1.219	0.268	18.62	1.034	64.93	4.91	18.47	1.026	13.23	1	18.39	1.022	20.71	1.57
70	2.0	14	1	17.78	1.270	0.192	14.04	1.003	24.59	6.60	14.02	1.002	3.88	1	14.01	1.001	4.28	1.10
70	3.0	20	1	26.30	1.315	0.489	20.96	1.048	109.03	3.64	20.81	1.041	29.99	1	20.54	1.027	35.55	1.19
70	4.0	26	1	31.49	1.211	0.688	26.88	1.034	154.73	4.51	26.17	1.007	34.31	1	26.04	1.002	38.20	1.11
100	2.0	17	1	23.91	1.406	0.488	18.15	1.068	105.03	6.89	17.86	1.051	15.24	1	17.58	1.034	17.44	1.14
100	3.0	29	1	37.53	1.294	0.950	32.09	1.107	213.65	9.06	30.87	1.064	23.58	1	30.56	1.054	28.59	1.21
100	4.0	36	1	44.42	1.234	1.387	39.43	1.095	316.97	9.99	38.11	1.059	31.72	1	37.77	1.049	39.02	1.23
200	2.0	32*	1	47.37	1.480	1.94	40.12	1.254	163.85	2.55	35.62	1.113	64.25	1	35.28	1.103	63.16	0.98
200	4.0	77*	1	87.66	1.138	4.75	84.04	1.091	529.66	5.13	80.86	1.050	103.29	1	77.92	1.082	92.28	0.89
300	2.5	72*	1	91.60	1.272	5.14	83.45	1.159	636.07	6.25	80.90	1.124	101.80	1	78.01	1.083	122.99	1.21
300	4.5	129*	1	140.67	1.090	11.80	135.15	1.048	1439.5	6.15	135.34	1.049	233.98	1	133.27	1.033	298.00	1.27
400	2.5	104*	1	124.26	1.195	7.10	114.35	1.100	1170.9	5.88	111.03	1.068	198.99	1	109.05	1.049	257.98	1.30
400	4.5	174*	1	190.91	1.097	15.49	185.60	1.067	2658.5	6.03	181.51	1.043	440.58	1	177.45	1.020	576.44	1.31
500	5.0	219*	1	253.22	1.156	24.29	249.10	1.137	5462.6	6.57	243.18	1.110	831.23	1	226.05	1.032	980.51	1.18
600	5.0	285*	1	304.55	1.069	43.57	300.20	1.053	9866.0	8.00	292.99	1.028	1233.3	1	289.05	1.014	1260.2	1.02
AVG			**1**		**1.221**			**1.069**		**5.77**		**1.045**		**1**		**1.032**		**1.16**

[1] The general pivot selection shown in Algorithm 2 is used in the local search optimization.
[2] The two-step successive pivot selection shown in Algorithm 3 is used in the local search optimization.
* For testcases with more than 200 vertices the ILP failed to produce solutions within limited time. The values here are the minimum numbers of interconnections among 1000 runs produced by our algorithm.

TABLE IV

PORT ASSIGNMENT ON HLS TESTBENCHES

Test	**W/O PA**		**Greedy [10]**		**FM [3]**		**Tree Based Opt.**		**Matrix Based Opt. I**		**Matrix Based Opt. II**	
Bench	INT	Time	# of INT	Time(ms)	# of INT	Time(ms)	# of INT	Time(ms)	# of INT	Time(ms)	# of INT	Time(ms)
arf	19	4	18(94.74%)	+0.022(0.55%)	18(94.74%)	+0.09(2.25%)	18(94.74%)	+0.39(9.75%)	17(89.47%)	+0.17(4.25%)	17(89.47%)	+0.18(4.50%)
ewf	17	4	16(94.12%)	+0.021(0.53%)	16(94.12%)	+0.08(2.00%)	15(88.24%)	+0.67(16.75%)	16(94.12%)	+0.14(3.50%)	15(88.24%)	+0.15(3.75%)
ar	31	6	28(90.32%)	+0.019(0.32%)	28(90.32%)	+0.05(0.83%)	28(90.32%)	+0.41(6.83%)	28(90.32%)	+0.07(1.17%)	28(90.32%)	+0.07(1.17%)
ae	35	16	32(91.43%)	+0.048(0.30%)	34(97.14%)	+0.11(0.69%)	31(88.57%)	+0.87(5.44%)	31(88.57%)	+0.31(1.94%)	27(77.14%)	+0.22(1.38%)
ad2	37	16	33(89.19%)	+0.035(0.22%)	30(81.08%)	+0.09(0.56%)	31(83.78%)	+1.41(8.81%)	31(83.78%)	+0.22(1.38%)	30(81.08%)	+0.29(1.81%)
ellip	45	14	42(93.33%)	+0.019(0.14%)	42(93.33%)	+0.03(0.21%)	40(88.89%)	+0.82(5.86%)	40(88.89%)	+0.05(0.36%)	40(88.89%)	+0.05(0.36%)
mpeg	35	20	32(91.43%)	+0.055(0.28%)	31(88.57%)	+0.02(0.10%)	30(85.71%)	+1.00(5.00%)	29(82.86%)	+0.10(0.50%)	29(82.86%)	+0.11(0.55%)
fft	75	46	67(89.33%)	+0.095(0.21%)	65(86.67%)	+2.19(4.76%)	63(84.00%)	+10.01(21.76%)	63 (84.00%)	+4.44(9.65%)	62(82.67%)	+4.52(9.83%)
rand0	81	21	74(91.36%)	+0.048(0.23%)	74(91.36%)	+0.32(1.52%)	73(90.12%)	+1.15(5.48%)	74(91.36%)	+0.40(1.90%)	72(88.89%)	+0.39(1.86%)
rand1	129	54	123(95.35%)	+0.053(0.10%)	118(91.47%)	+0.99(1.83%)	120(93.02%)	+4.47(8.28%)	118(91.47%)	+1.09(2.02%)	118(91.47%)	+1.28(2.37%)
rand2	458	371	418(91.27%)	+0.591(0.16%)	415(90.61%)	+15.22(4.80%)	409(89.30%)	+92.70(24.99%)	405(88.43%)	+17.24(4.65%)	402(87.77%)	+19.69(5.31%)
rand3	554	260	513(92.60%)	+1.81(0.70%)	499(90.07%)	+5.26(2.02%)	487(87.91%)	+82.18(31.61%)	485(87.55%)	+9.09(3.50%)	480(86.64%)	+12.86(4.95%)
rand4	692	417	654(94.51%)	+2.05(0.49%)	636(91.91%)	+9.90(2.37%)	630(91.04%)	+48.77(11.70%)	623(90.03%)	+15.02(3.60%)	621(89.74%)	+18.09(4.34%)
rand5	1158	2137	1030(88.95%)	+5.84(0.27%)	1022(88.26%)	+60.27(2.82%)	1020(88.08%)	+237.8(11.13%)	1019(88.00%)	+88.50(4.14%)	1006(78.50%)	+94.22(4.41%)
rand6	1113	2088	1068(95.96%)	+12.14(0.58%)	1050(94.34%)	+87.20(4.18%)	1044(93.80%)	+426.2(20.41%)	1033(92.81%)	+108.2(5.18%)	1031(92.63%)	+121.7(5.83%)
AVG			**92.26%**	**+0.34%**	**90.97%**	**+2.06%**	**89.17%**	**+12.92%**	**88.78%**	**3.18%**	**86.42%**	**+3.49%**

of interconnections by 9% with a 2.06% time overhead. The elementary-tree-transformation-based algorithm reduces the number of interconnections by 10%, which is slightly higher than FM but with a large time overhead of 13%. The matrix-based optimization using general pivot selection reduces the interconnections by 11.2% with a 3.18% time overhead; the two-step pivot selection shows the highest optimality, reducing the interconnections by 13.7% with a 3.49% time overhead.

The results show that the matrix method proposed in this paper excels the FM-based algorithm in terms of optimality, and we think the possible reasons are the following.

1) The FM-based algorithm in [3] is a local-search-based optimization method, where a bipartite graph is used for vertex partition. In each iteration of local search, on the bipartite graph, either one vertex is moved from one partition to the other or two vertices are swapped to change their partitions. Thus, each time the number of vertices whose partitions are changed is limited to 1 or 2. It means that when the solutions in the neighborhood are being searched by local search, only the ones that are quite close to the current solution are investigated, that is, the search area is very narrow. In the actual experiments, it shows that the possibility that a better solution exists nearby the current solution is low; as a result, the local search mostly terminates only after a few iterations, which is fast but limits the optimality of the algorithm. It implies that the diversification of search area for this problem is necessary.
2) In the matrix-based algorithm, on the other hand, during the optimization, the edge status tables are built and pivoting operations are performed on the matrix. In each

TABLE V
ESTIMATED AREA, POWER, AND CRITICAL PATH DELAY OF ALL MUXes

Test	W/O Port Assignment			Greedy [10]			Ours			
Bench	Power(μW)	Area(μm^2)	Delay(nm)	Power(μW)	Area(μm^2)	Delay(nm)	Power(μW)	Area(μm^2)	Delay(w/o cp)	Delay(with cp)
arf	256.29	2030	1.05	247.78 (96.68%)	1937 (95.42%)	1.05 (100.0%)	239.31 (93.37%)	1829 (90.10%)	1.05 (100.0%)	0.93 (88.57%)
ewf	220.37	1807	1.05	211.73 (96.08%)	1699 (94.23%)	0.93 (88.57%)	203.26 (92.24%)	1591 (88.05%)	0.93 (88.57%)	0.93 (88.57%)
ar	356.16	3198	1.14	322.48 (90.54%)	2949 (92.21%)	1.05 (92.11%)	322.48 (90.54%)	2949 (92.21%)	1.05 (92.11%)	1.05 (92.11%)
ae	391.85	3666	1.23	367.05 (93.67%)	3380 (92.20%)	1.23 (100.0%)	324.00 (82.68%)	2841 (77.50%)	1.14 (92.68%)	1.05 (85.37%)
ad2	406.89	3808	1.44	374.57 (92.06%)	3451 (90.63%)	1.23 (85.42%)	349.76 (85.96%)	3164 (83.09%)	1.23 (85.42%)	1.14 (79.17%)
ellip	416.60	3949	1.53	394.12 (94.60%)	3735 (94.58%)	1.49 (97.39%)	379.17 (91.02%)	3593 (90.99%)	1.44 (94.12%)	1.31 (85.62%)
mpeg	411.47	3697	1.23	385.68 (93.73%)	3388 (91.64%)	1.23 (100.0%)	359.92 (87.47%)	3165 (85.61%)	1.23 (100.0%)	1.05 (85.37%)
fft	720.84	6830	1.49	660.77 (91.67%)	6262 (91.68%)	1.49 (100.0%)	622.14 (86.31%)	5870 (85.94%)	1.44 (96.64%)	1.44 (96.64%)
rand0	867.57	8149	1.31	812.68 (93.67%)	7579 (93.01%)	1.31 (100.0%)	797.63 (91.94%)	7437 (91.26%)	1.23 (93.89%)	1.05 (80.15%)
rand1	1151.00	10930	1.66	1106.34 (96.12%)	10510 (96.16%)	1.61 (96.99%)	1069.01 (92.88%)	10153 (92.89%)	1.61 (96.99%)	1.57 (94.58%)
rand2	4489.64	42507	1.78	4183.37 (93.18%)	39486 (92.89%)	1.76 (98.87%)	4062.10 (90.47%)	38313 (90.13%)	1.74 (97.75%)	1.61 (90.45%)
rand3	5416.90	51224	1.87	5106.87 (94.28%)	48238 (94.17%)	1.83 (97.86%)	4854.54 (89.62%)	45748 (89.31%)	1.80 (96.26%)	1.76 (94.12%)
rand4	6454.39	61163	2.02	6170.18 (95.60%)	58453 (95.57%)	1.95 (96.53%)	5923.31 (91.77%)	56092 (91.71%)	1.91 (94.55%)	1.85 (91.58%)
rand5	10254.15	97466	2.08	9301.22 (90.71%)	88288 (90.58%)	2.02 (97.12%)	9122.93 (88.97%)	86569 (88.82%)	2.00 (96.15%)	1.89 (90.87%)
rand6	9919.28	94226	2.04	9584.13 (96.62%)	91014 (96.59%)	2.02 (99.02%)	9308.06 (93.84%)	88346 (93.76%)	1.95 (95.59%)	1.87 (91.67%)
AVG	**1**	**1**	**1**	**93.94%**	**93.41%**	**96.65%**	**89.93%**	**88.75%**	**94.71%**	**88.98%**

iteration of local search, the number of variables whose values are changed is the size of fundamental cut set $FC(e_t)$. This corresponds to the operation that changes the partitions of the descendant vertices of tree edge e_t on the graph. Thus, the number of vertices whose partitions are changed is much larger than that in [3]. It means that in the matrix-based algorithm, the solutions far from the current solution are also investigated and the search area of our proposal is broader than [3]. Moreover, by proposing the successive pivoting method, the search area is further broadened by predicting the solutions of the next iteration. Benefiting from the diversification of search area, the matrix-based algorithm shows a higher optimality than [3].

C. Power, Area, and Delay Evaluation

In this section, we show the estimated improvements of area, power, and critical path delay achieved by our port assignment algorithm. Reducing the number of interconnections by port assignment leads to the reduction in MUX size before the input ports of FUs; smaller MUXes result in smaller area, lower power, and shorter critical path delay. In this paper, we estimate the reduction of area, power, and critical path delay by calculating all the MUXes before input ports of FUs, because our work does not change the FU and register binding but only affects the MUX size. The area, power, and delay information of MUXes of different sizes are obtained from [20] and shown in Table II. Since only 2-to-1, 4-to-1, 8-to-1, 16-to-1, and 32-to-1 MUXes are given, we combine them to make MUXes of other sizes like 23-to-1. The total area and power are computed by summarizing all the MUXes in a benchmark and the critical path delay is approximately estimated using the largest MUX in a benchmark.

Table V shows the estimated reduction of area, power, and critical path delay of all MUXes after port assignment, by comparing with the results without port assignment. It shows that the work in [10] is able to reduce power by 6% and area by 6.6% of MUXes on average and our work is able to reduce power by 10% and area by 11.2% on average. As for the critical path delay, the work in [10] reduces the delay by 3.5% and ours reduces the delay by 5.3% if no special optimization for critical path is conducted, as shown in the column Delay(w/o cp). If the critical path is considered, the delay is reduced by 11%, as shown in the column Delay(with cp), which is achieved by reducing the size of the largest MUX.

TABLE VI
IMPLEMENTATION OF TESTBENCH FFT ON FPGA PLATFORM UNDER THE 50-MHz CLOCK FREQUENCY

	Power(mW)			Resource (Area)		Delay(ns)	
	Total	*dyn*	*leak*	# of Reg.	# of LUTs	Max. Frq.[1]	Min. Prd.[2]
W/O PA	327	286	41	2282	3598	57.607	17.359
Greedy [10]	313	272	41	2092	3518	58.517	17.089
Ours w/o cp	289	249	40	1900	3453	60.140	16.628
Ours	289	249	40	1908	3446	60.190	16.614

[1] Max. Frq: maximum frequency (MHz)
[2] Min. Prd: minimum period (ns)

Moreover, we pick up a most typical testbench FFT and implement it on the FPGA platform to evaluate the actual improvement in terms of power, area, and delay. The simulation is conducted using Xillinx ISE 14.7 on the evaluation platform Spartan-6 SP605 evaluation platform under a clock frequency of 50 MHz. The results are shown in Table VI, including the result without port assignment, the result produced by [10], and the result produced by ours without and with critical path consideration. It shows that the port assignment indeed reduces the power and area (reflected by the resources on FPGA) and increases the maximum clock frequency as well. Compared with the work in [10], our algorithm achieves more power and area reduction and higher clock frequency.

VII. CONCLUSION

In this paper, we presented a port assignment algorithm for interconnect reduction after the FU and register binding in HLS. We formulated the problem on a constraint graph and proposed an algorithm to generate feasible initial solutions using conflict graph and minimum vertex cover. We also proposed two optimization methods: the elementary tree opti-

mization and the matrix-based optimization to improve the solution quality. Given the properties of the matrix formulation, we adopted the simplex method and pivoting operations to perform optimizations. We proposed two methods for pivot selection, the general one and the two-step successive pivotings, to improve the efficiency and optimality of pivot selection. The experimental results showed that on the randomly generated test cases, the matrix-based port assignment algorithm using two-step pivot selection showed the highest optimality. On the real benchmarks, our proposal reduced 14% interconnections, while the previous work reduced 8% on average.

The current algorithm proposed in this paper assumes a fixed FU and register binding; it is expected that if FU and register binding are not fixed and being solved with port assignment jointly, and the number of interconnections can be further reduced. For this purpose, our proposed conflict graph is expected to be adopted to estimate the number of interconnections rather than actually computing the vertex cover, because vertex cover is much more time consuming than conflict graph generation. Another future research direction is that the port assignment problem is expected to be solved on chained FUs with more than two input ports, say three. In this case, the set cover problem can be first solved to determine the registers that are connected to the third port, and then our algorithm can be applied on the remaining two ports. When the number of ports is more than 3, more explorations are needed.

ACKNOWLEDGMENT

This work is extended from the papers published in VLSI-DAT2012 [1] and ASQED2013 [2].

REFERENCES

[1] H. Cong, S. Chen, and T. Yoshimura, "Port assignment for interconnect reduction in high-level synthesis," in *Proc. Int. Symp. VLSI Design, Autom., Test (VLSI-DAT)*, Apr. 2012, pp. 1–4.

[2] C. Hao, H.-R. Zhang, S. Chen, T. Yoshimura, and M.-Y. Wu, "Port assignment for multiplexer and interconnection optimization," in *Proc. 5th Asia Symp. Quality Electron. Design (ASQED)*, Aug. 2013, pp. 136–143.

[3] C. Hao, N. Wang, S. Chen, T. Yoshimura, and M.-Y. Wu, "Interconnection allocation between functional units and registers in high-level synthesis," in *Proc. IEEE 10th Int. Conf. ASIC (ASICON)*, Oct. 2013, pp. 1–4.

[4] D. Liu and C. Svensson, "Power consumption estimation in CMOS VLSI chips," *IEEE J. Solid-State Circuits*, vol. 29, no. 6, pp. 663–670, Jun. 1994.

[5] N. Magen, A. Kolodny, U. Weiser, and N. Shamir, "Interconnect-power dissipation in a microprocessor," in *Proc. Int. Workshop Syst. Level Interconnect Predict.*, 2004, pp. 7–13.

[6] D. Chen, J. Cong, Y. Fan, and L. Wan, "LOPASS: A low-power architectural synthesis system for FPGAs with interconnect estimation and optimization," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 18, no. 4, pp. 564–577, Apr. 2010.

[7] B. M. Pangrle, "On the complexity of connectivity binding," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 10, no. 11, pp. 1460–1465, Nov. 1991.

[8] S. Raje and R. A. Bergamaschi, "Generalized resource sharing," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, Nov. 1997, pp. 326–332.

[9] S. Yamada, "An optimal block terminal assignment algorithm for VLSI data path allocation," *IEICE Trans. Fundam. Electron., Commun. Comput. Sci.*, vol. E80-A, no. 3, pp. 564–566, 1997.

[10] D. Chen and J. Cong, "Register binding and port assignment for multiplexer optimization," in *Proc. Asia South Pacific Design Autom. Conf.*, Jan. 2004, pp. 68–73.

[11] P. Brisk and P. Ienne, "On the complexity of the port assignment problem for binary commutative operators in high-level synthesis," in *Proc. Int. Symp. VLSI Design, Autom. Test (VLSI-DAT)*, Apr. 2009, pp. 339–342.

[12] *LP-Solver*. [Online]. Available: http://lpsolve.sourceforge.net/5.5/

[13] V. N. Kasyanov and V. A. Evstigneev, *Graph Theory for Programmers: Algorithms for Processing Trees*, vol. 515. Tokyo, Japan: Springer, 2000.

[14] L. R. Foulds, *Graph Theory Applications*. Tokyo, Japan: Springer, 2012.

[15] J. R. Bunch and D. J. Rose, Eds., *Sparse Matrix Computations*. San Francisco, CA, USA: Academic, 2014.

[16] H. S. Kasana and K. D. Kumar, *Introductory Operations Research: Theory and Applications*. Tokyo, Japan: Springer, 2013.

[17] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. North Chelmsford, MA, USA: Courier Corporation, 1998.

[18] C. Hao, J.-M. Ni, H.-T. Wang, and T. Yoshimura, "Simultaneous scheduling and binding for resource usage and interconnect complexity reduction in high-level synthesis," in *Proc. 11th Int. Conf. ASIC (ASICON)*, Nov. 2015, pp. 1–4.

[19] *TGFF*. accessed on Sep. 18, 2016. [Online]. Available: http://ziyang.eecs.umich.edu/~dickrp/tgff/

[20] P. Balasubramanian and D. A. Edwards, "Power, delay and area efficient self-timed multiplexer and demultiplexer designs," in *Proc. 4th Int. Conf. Design Technol. Integr. Syst. Nanoscal Era*, Apr. 2009, pp. 173–178.

Cong Hao received the B.S. degree in computer science and the M.S. degree from Shanghai Jiao Tong University, Shanghai, China, in 2011 and 2014, respectively, and the M.S. degree from the Graduate School of Information, Production and System, Waseda University, Kitakyushu, Japan, in 2012, where she is currently pursuing the Ph.D. degree.

Her current research interests include very large scale integration design automation, network-on-chip, and reconfigurable architecture.

Jianmo Ni received the B.S. degree from Shanghai Jiao Tong University, Shanghai, China, in 2013. He attended the dual-master program at Shanghai Jiao Tong University and Waseda University, Kitakyushu, Japan, in 2012, and received the M.E. degree from Waseda University in 2014. Currently, he is pursuing the M.E. degree with Shanghai Jiao Tong University.

His current research interests include smart grid, very large scale integration computer-aided design, and Internet-of-Things.

Nan Wang received the B.S. degree from Nanjing University, Nanjing, China, in 2009, and the M.S. and Ph.D. degrees from the Graduate School of Information, Production and System, Waseda University, Kitakyushu, Japan, in 2011 and 2014, respectively.

He is currently a Lecturer of Electronics and Communication Engineering with the East China University of Science and Technology, Shanghai, China. His current research interests include very large scale integration design automation, low power design techniques, network-on-chip, and reconfigurable architectures.

Takeshi Yoshimura (M'16) received the B.E., M.E., and Dr.Eng. degrees from Osaka University, Osaka, Japan, in 1972, 1974, and 1997, respectively.

He joined NEC Corporation, Tokyo, Japan, in 1974, where he was involved in the research and development of computer application systems for communication network design, hydraulic network design, and very large scale integration (VLSI) CAD. From 1979 to 1980, he was on leave at the Electronics Research Laboratory, University of California at Berkeley, Berkeley, CA, USA, where he worked on VLSI CAD. In 2003, he joined the Graduate School of Information, Production and Systems, Waseda University, Tokyo.

Dr. Yoshimura is a fellow of IEICE and a member of the Information Processing Society of Japan.