# Low-Power Gated Bus Synthesis for 3D IC via Rectilinear Shortest-path Steiner Graph

Chung-Kuan Cheng[1,2], Peng Du[1], Andrew B. Kahng[1,2] and Shih-Hung Weng[1]
[1]Dept. of Computer Science and Engineering, University of California San Diego, La Jolla, CA, 92093
[2]Dept. of Electrical and Computer Engineering, University of California San Diego, La Jolla, CA, 92093
ckcheng@ucsd.edu, pedu@ucsd.edu, abk@ucsd.edu, s2weng@ucsd.edu

## ABSTRACT

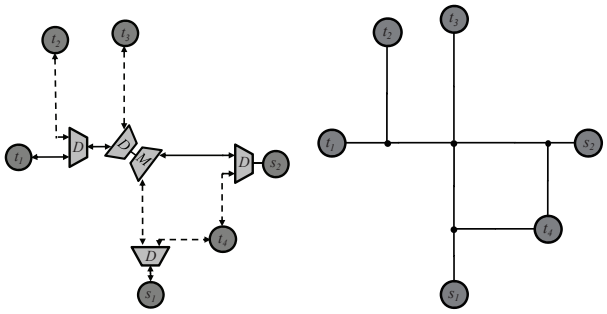
In this paper, we propose a new approach for gated bus synthesis [16] with minimum wire capacitance per transaction in three-dimensional (3D) ICs. The 3D IC technology connects different device layers with through-silicon vias (TSV), which need to be considered differently from metal wire due to reliability issues and a larger footprint. Practically, the number of TSVs is bounded between layers; thus, we first devise dynamic programming and local search techniques to determine the optimal TSV locations. We then employ two approximation algorithms to generate a rectilinear shortest-path Steiner graph in each device layer. One algorithm extends the well-known greedy heuristic for the Rectilinear Steiner Arborescence problem and handles large cases with high efficiency. The other algorithm utilizes a linear programming relaxation and rounding technique which costs more time and generates a nearly-optimal Steiner graph. Experimental results show that our algorithms can construct shortest-path Steiner graphs with 22% less total wire length than the previous method of Wang et al. [16].

**Categories and Subject Descriptors**
B.7.2[**Integrated Circuits**]:Design Aids–placement and routing
**General Terms**
Algorithm
**Keywords**
3D IC, TSV, shortest-path Steiner graph, Gated Bus

## 1. INTRODUCTION

Three-dimensional (3D) IC technology offers the potential for improving performance and power consumption for bus architectures in SoC design [4]. 3D IC technology uses through-silicon vias (TSVs) to connect device layers. In SoC designs, TSVs can potentially reduce global interconnect among IPs, improving communication power efficiency and performance of bus-based architectures. For example, Pathak et al. [12] show that the timing of a 3D LEON3 multi-core design with AMBA bus architecture connected by TSVs is better than that of a 2D design by about 79%.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
*ISPD'12,* March 25–28, 2012, Napa, California, USA.
Copyright 2012 ACM 978-1-4503-1167-0/12/03 ...$10.00.




(a) Bus implementation. (b) Shortest-path Steiner graph.

**Figure 1: Gated bus architecture.**

A 3D IC design needs many TSVs for data bus, address bus and control wires to connect IPs that are on different layers. However, the number of TSVs must be limited since TSV is costly to fabricate, and a large number of TSVs may degrade manufacturing yield, test cost efficiencies and available layout area on-chip. Given this consideration, designers cannot generously use TSVs for communication between device layers. Therefore, selection of TSV positions becomes important when designers seek to increase performance and reduce power consumption of bus architectures while using a restricted number of TSVs.

Many researchers [5][11][16] have worked on low-power bus architectures, reducing unnecessary wire loading or signal switching when the bus transfers data. Bus segmentation [5] and bus splitting [11] methods reduce wire load by masking off certain bus segments, but only consider where to mask off within a given bus topology to achieve maximum power savings. In the gated bus architecture [16], Wang *et al.* synthesize a gated bus topology to reduce the wire capacitance by using distributed multiplexers and demultiplexers.

Figure 1(a) shows an implementation of the gated bus architecture with two masters $s_1, s_2$ and four slaves $t_1, \ldots, t_4$. The gray multiplexers and demultiplexers mask off the unused path when transferring data from $s_2$ to $t_1$ so that $s_2$ only needs to drive the path consisting of the three solid arrows. Therefore, driven wire capacitance is greatly reduced. Figure 1(b) shows the Steiner graph representation of Figure 1(a). The Steiner points of Figure 1(b) indicate where to place multiplexers/demultiplexers in the bus architec-

ture. To have the minimum wire capacitance for every data transaction, the Steiner graph must contain a shortest path with length equal to the Manhattan distance between each master-slave pair. We call such a Steiner graph a *shortest-path Steiner graph*; our objective is to find a shortest-path Steiner graph with smallest total wire length. In [16], Wang *et al.* proposed a heuristic method that first constructs a minimum rectilinear Steiner arborescence [8][13] starting from a master, and then adds other masters one by one in each iteration to obtain a shortest-path Steiner graph.

In this paper, we investigate the problem of gated bus synthesis in 3D ICs to minimize total power consumption of the gated bus architecture. Since constraints of TSVs are different from these of on-die vias, e.g., larger footprint and keep-out zones, we must consider TSVs separately in this problem. We first develop dynamic programming and local search algorithms to determine TSV locations that minimize the sum of weighted shortest distances over all master-slave pairs in a 3D IC design. Then, given the TSV locations thus determined, we propose two approximation algorithms to synthesize a shortest-path Steiner graph with smallest total wire length on each layer of the 3D IC stack; this graph determines locations of multiplexers and demultiplexers for the gated bus architecture. The two algorithms include a greedy heuristic that can handle large instances and a linear programming relaxation and rounding method [15] that is more accurate and suitable for solving cases with relatively small scale. Overall, our method can reduce wire length by up to 22% when compared to [16].

The remainder of this paper is organized as follows. In Section 2, we introduce the problem of gated bus synthesis in 3D ICs and formally state two problem formulations on TSV location determination and construction of shortest-path Steiner graphs. In Section 3, we give two algorithms for determination of TSV locations; these address the cases of one TSV and multiple TSVs between adjacent layers, respectively. Section 4 describes our approximation algorithms to generate a shortest-path Steiner graph with minimum total wire length. Experimental results are shown in Section 5, and conclusions are given in Section 6.

## 2. PROBLEM FORMULATION

We assume that we are given locations of masters and slaves on device layers, and the communication frequency for each master-slave pair. Our goal is to minimize total power consumption over all master-slave pairs while keeping the total wire length as small as possible. Total power is estimated as the summation of frequency multiplied by capacitance, i.e., only dynamic power. Note that since introduced auxiliary gates in our bus architecture only take a small percentage of total gate count in a design, the increase of leakage power is negligible (ignoring possible effects of power-gating).

As noted above, we cannot place an arbitrary number of TSVs between adjacent device layers. Hence, we assume that there is a given constant $B$ indicating the maximum number of TSVs available for connecting masters and slaves between two device layers. Our experimental results (below, Section 5) show that when $B > 2$, this trade-off achieves nearly the optimal power consumption when the number of TSVs between layers is unbounded ($B = \infty$). Besides, with so few TSVs, the power impact of large capacitance TSV is insignificant. The overhead of the control wires in the gated bus architecture is small compared to the data bus and the address bus because the number of such wires is much less than the bus width. Moreover, since the control signals do not switch during data transaction, the dynamic power associated with control signal is small. We therefore ignore effects of control wiring in this study.

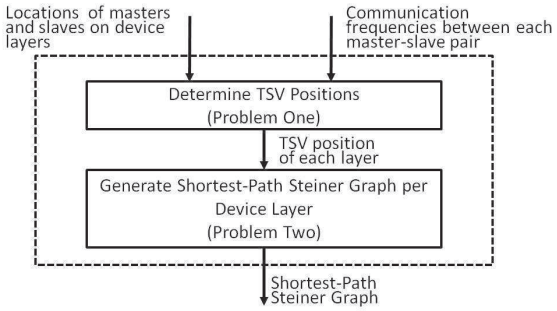



**Figure 2: Overall flow.**

Figure 2 shows the overall flow of our method. The locations of masters and slaves can be obtained after performing 3D floorplanning [2][6]. The communication frequency can be calculated from gate-level simulation. We address the following two sub-problems in Sections 3 and 4, respectively.

DEFINITION 1. *A rectilinear shortest path between a master-slave pair in the same layer is a shortest path with length equal to the Manhattan distance between them.*

- Problem One: Find locations of TSVs between adjacent layers so that the total length of weighted shortest paths between master-slave pairs is minimized. We take frequencies to be weights and only $B$ TSV locations can be assigned between two adjacent layers.
- Problem Two: Given optimal (fixed, from Problem One) locations of TSVs, construct a rectilinear shortest-path Steiner Graph in each layer that minimizes the total wire length subject to the existence of a layer-wise rectilinear shortest path for each master-slave pair. This is the *minimal rectilinear shortest-path Steiner graph* (RSSG) problem, which was first considered in [16].

## 3. PROBLEM ONE: DETERMINATION OF TSV LOCATIONS

The input of our problem includes the locations of $n$ masters $\{s_1, s_2, \ldots, s_n\}$ and $m$ slaves $\{t_1, t_2, \ldots, t_m\}$ on $L$ device

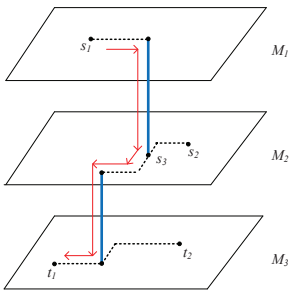



**Figure 3: Two TSVs between three device layers.**

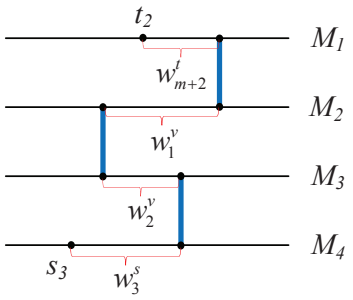



**Figure 4: Contribution of an s-t path $s_3 - t_2$ to weights.**

layers from $M_1$ to $M_L$, and the communication frequency $c_{i,j}$ for each master-slave pair $(s_i, t_j)$. We assume that all device layers have the same size $W \times H$. Given the upper bound $B$ of the number of TSVs between adjacent device layers, our first objective is to determine the best locations of TSVs so that the sum of weighted shortest-path lengths over all $s_i - t_j$ pairs, for $1 \le i \le n$ and $1 \le j \le m$, is as small as possible. The weight for the shortest path between $s_i$ and $t_j$ is chosen to be $c_{i,j}$. For example, Figure 3 shows a possible assignment of two sets of TSVs in blue between three device layers where the shortest path through TSVs from $s_1$ to $t_1$ is denoted by red arrows. The length of TSVs will be neglected in the following discussion since it does not affect the objective value. We denote the planar coordinates and layers of $s_i$ and $t_j$ for $1 \le i \le n$ and $1 \le j \le m$ by $(x^s_i, y^s_i, l^s_i)$ and $(x^t_j, y^t_j, l^t_j)$ respectively. The candidate positions of the TSVs are determined by the following theorem.

THEOREM 1. Given $X = \{x^s_1, \ldots, x^s_n, x^t_1, \ldots, x^t_m\}$ and $Y = \{y^s_1, \ldots, y^s_n, y^t_1, \ldots, y^t_m\}$, the optimal TSV positions can be chosen from the set $\{(x, y) : x \in X, y \in Y\}$, i.e., the Hanan grid [10] formed by union of masters and slaves.

PROOF. For simplicity, we only consider the case when $B = 1$. The proof can be extended to the case of larger $B$ value. By the additivity of rectilinear distance for horizontal and vertical directions, we can determine the $x$ and $y$ coordinates of TSVs separately. Without loss of generality, we only consider $x$ coordinates of TSV locations. Suppose $x^v_1, x^v_2, \ldots, x^v_{L-1}$ is an optimal assignment of TSVs where $x^v_k$ denotes the $x$ coordinate of a (set of) TSV(s) connecting adjacent layers $M_k$ and $M_{k+1}$, for $1 \le k \le L-1$. Let $k_0$ be the smallest layer index such that $x^v_{k_0} \notin X$. We choose $x^1 = \max\{x : x < x^v_{k_0} \text{ and } x \in X\}$ and $x^2 = \min\{x : x > x^v_{k_0} \text{ and } x \in X\}$. Using the property of linear length metric, we can show that there exists $x' \in \{x^1, x^2, x^v_{k_0+1}\}$ so that moving $x^v_{k_0}$ to $x'$ will not increase the objective value. If $x' = x^1$ or $x^2$, we achieve an optimal solution where the positions of the first $k_0$ TSVs are chosen from the Hanan grid instead of $k_0 - 1$. Otherwise if $x' = x^v_{k_0+1}$, we move $x^v_{k_0}$ to $x'$ and repeat the previous process by moving $x^v_{k_0}$ and $x^v_{k_0+1}$ together. By induction, we have that there exists an optimal solution in which all TSVs are located on the Hanan grid. ■

As noted in [9], restriction to Hanan grids may be suboptimal for minimizing the maximum delay between source-terminal pairs. We assume that timing constraints can still be met by adding buffers in buses. In the following two subsections, we present an efficient dynamic programming algorithm to determine the optimal TSV location assignment when $B = 1$ and a local search heuristic to determine the approximately optimal TSV location assignment when $B > 1$.

## 3.1 One TSV between Adjacent Layers

In the case where $B = 1$, we can determine the $x$ and $y$ coordinates of TSVs separately by coordinate-wise additivity of rectilinear distance. Here, we give the algorithm for finding $x$ coordinates of TSVs. We again represent an assignment of TSV locations by $x^v_1, x^v_2, \ldots, x^v_{L-1}$ where $x^v_k$ denotes the $x$ coordinate of the single TSV between $M_k$ and $M_{k+1}$. Now our objective can be expressed as

$$\begin{aligned}
&\min_{x^v_1,\ldots,x^v_{L-1}} \sum_{i=1}^{n}\sum_{j=1}^{m} c_{i,j} d(s_i, t_j) \\
= &\min_{x^v_1,\ldots,x^v_{L-1}} \Big(\sum_{k=1}^{L-2} w^v_k |x^v_{k+1} - x^v_k| \\
+ &\sum_{i=1}^{n} w^s_i |x^s_i - x^v_{l^s_i - 1}| + \sum_{i=1}^{n} w^s_{n+i} |x^s_i - x^v_{l^s_i}| \\
+ &\sum_{j=1}^{m} w^t_j |x^t_j - x^v_{l^t_j - 1}| + \sum_{j=1}^{m} w^t_{m+j} |x^t_j - x^v_{l^t_j}| \\
+ &\sum_{i=1}^{n}\sum_{j=1}^{m} \chi(l^s_i = l^t_j) c_{i,j} |x^s_i - x^t_j|\Big), \qquad (1)
\end{aligned}$$

where $d(s_i, t_j)$ is the length of a shortest path from $s_i$ to $t_j$; $w^s_i, w^t_j$ and $w^v_k$ indicate constant weights corresponding to each term; and $\chi$ is a 0-1 indicator function. We explain the meaning of each term of (1) as follows.

- $|x^v_{k+1} - x^v_k|$: The segment between two TSVs connecting $M_{k+1}$ with $M_{k+2}$ and $M_k$ with $M_{k+1}$, respectively.
- $|x^s_i - x^v_{l^s_i - 1}|$: The segment between $s_i$ and the TSV connecting the layer of $s_i$ with the layer above.
- $|x^s_i - x^v_{l^s_i}|$: The segment between $s_i$ and the TSV connecting the layer of $s_i$ with the layer below.
- $|x^t_j - x^v_{l^t_j - 1}|$: The segment between $t_j$ and the TSV connecting the layer of $t_j$ with the layer above.
- $|x^t_j - x^v_{l^t_j}|$: The segment between $t_j$ and the TSV connecting the layer of $t_j$ with the layer below.
- $\chi(l^s_i = l^t_j)|x^s_i - x^t_j|$: The segment between $s_i$ and $t_j$ if they are on the same layer.

If the shortest path from $s_i$ to $t_j$ passes through a segment listed above, it will contribute $c_{i,j}$ to the corresponding weight constant. Figure 4 gives an example where the shortest path between $s_3$ and $t_2$ passes through four segments whose corresponding weight constants are labeled below. Therefore, the shortest path will contribute $c_{3,2}$ to $w^s_3, w^v_2, w^v_1$ and $w^t_{m+2}$, respectively.

Since the last term of (1) is constant, we remove it from our objective for simplicity. Let $\vec{x} = (x^s_1, \ldots, x^s_n, x^t_1, \ldots, x^t_m)$ and let $\vec{x}(r)$ be the $r^{th}$ coordinate of $\vec{x}$ for $1 \le r \le n+m$. Let $S(k)$ and $T(k)$ respectively denote the sets of masters and slaves in layer $M_k$, for $1 \le k \le L$. To derive our dynamic programming algorithm, we use a function $OPT$ to indicate

partial solutions, defined as

$$\begin{aligned} OPT(k_c, r_c) &= \min_{x_1^v,\ldots,x_{k_c}^v=\vec{x}(r_c)} \Big( \sum_{k=1}^{k_c-1} w_k^v |x_{k+1}^v - x_k^v| \\ &+ \sum_{k=1}^{k_c} \sum_{i\in S(k)} (w_i^s |x_i^s - x_{k-1}^v| + w_{n+i}^s |x_i^s - x_k^v|) \\ &+ \sum_{k=1}^{k_c} \sum_{j\in T(k)} (w_j^t |x_j^t - x_{k-1}^v| + w_{m+j}^t |x_j^t - x_k^v|)\Big), \\ &1 \le k_c \le L-1; 1 \le r_c \le n+m. \end{aligned} \quad (2)$$

The initial values of function $OPT$ can be evaluated as

$$\begin{aligned} OPT(1, r_c) &= \sum_{i\in S(1)} w_{n+i}^s |x_i^s - \vec{x}(r_c)| \\ &+ \sum_{j\in T(1)} w_{m+j}^t |x_j^t - \vec{x}(r_c)|, 1 \le r_c \le n+m. \end{aligned} \quad (3)$$

The value of $OPT(k_c, r_c)$ indicates the minimal total length of weighted shortest paths between masters and slaves in layers $\{M_1, \ldots, M_{k_c}\}$ plus weighted shortest paths from masters/slaves in $\{M_1, \ldots, M_{k_c}\}$ to the set of TSVs with position $\vec{x}(r_c)$ between $M_{k_c}$ and $M_{k_c+1}$. The key step for computing $OPT$ is expressed by the following recursive formula:

$$\begin{aligned} OPT(k_c, r_c) &= \min_{1\le r\le n+m} (OPT(k_c-1, r) + w_{k_c-1}^v |\vec{x}(r_c) - \vec{x}(r)| \\ &+ \sum_{i\in S(k_c)} (w_i^s |x_i^s - \vec{x}(r)| + w_{n+i}^s |x_i^s - \vec{x}(r)|) \\ &+ \sum_{j\in T(k_c)} (w_j^t |x_j^t - \vec{x}(r)| + w_{m+j}^t |x_j^t - \vec{x}(r)|)), \\ &2 \le k_c \le L-1; 1 \le r_c \le n+m. \end{aligned} \quad (4)$$

Intuitively, since all communications from masters/slaves in $\{M_1, \ldots, M_{k_c-1}\}$ to masters/slaves in $M_{k_c}$ and the TSV between $M_{k_c}$ and $M_{k_c+1}$ must go through the TSV between $M_{k_c-1}$ and $M_{k_c}$, we only need to enumerate all possible positions $\vec{x}(r)$ between $M_{k_c-1}$ and $M_{k_c}$ and choose the best one to achieve $OPT(k_c, r_c)$. Finally, our objective (1) will be computed as

$$\begin{aligned} \min_{1\le r\le n+m} \quad & (OPT(L-1, r) + \textstyle\sum_{i\in S(L)} w_i^s |x_i^s - \vec{x}(r)| \\ + \quad & \textstyle\sum_{j\in T(L)} w_j^t |x_j^t - \vec{x}(r)|). \end{aligned} \quad (5)$$

THEOREM 2. The time complexity of the dynamic programming algorithm for determination of TSV locations is $O((n+m)^2 L)$.

PROOF. As in recursive formula (4), each $OPT(k_c, r_c)$ can be evaluated in time $O(n+m)$ where the summation term can be gradually updated in constant time as $r$ is increasing. Since there are $O((n+m)L)$ number of $OPT$ values, we can obtain all of them in time $O((n+m)^2 L)$. Computing the final solution by (5) takes additional $O(n+m)$ time. Therefore, the total running time of our algorithm is $O((n+m)^2 L) + O(n+m) = O((n+m)^2 L)$. ■

## 3.2 Multiple TSVs between Adjacent Layers

When multiple TSVs between adjacent layers are allowed (i.e., $B > 1$), we cannot determine TSV locations for $x$ and $y$ coordinates separately. Therefore, we represent the TSVs' locations by $\{(x_{kb}^v, y_{kb}^v) : 1 \le k \le L-1, 1 \le b \le B\}$ where $(x_{kb}^v, y_{kb}^v)$ denotes the location of the $b^{th}$ TSV between layers $M_k$ and $M_{k+1}$. We describe a local search heuristic to determine locations of TSVs in Algorithm 1. It first finds a best assignment of TSVs on a coarse grid $Z$ by exhaustive search and then updates it locally to find better solutions until there is no improvement of the objective function. If $Z$ is a candidate set of positions for TSVs, each element $V$ in the set $Z^{(L-1)B}$ corresponds to an assignment of TSVs on $Z$. We use $V(g)$ to denote the $g^{th}$ element of $V$ by taking $V$ as a sequence of elements in $Z$ with length $(L-1)B$. If $V$ is the assignment of TSVs, the function $TotShortestPaths(V)$ returns the total length of shortest paths weighted by communication frequencies between each pair of masters and slaves, .

**Algorithm 1:** Finding multiple TSVs by local search

**Input**: $n, m, L, B, \{(x_i^s, y_i^s, l_i^s) : 1 \le i \le n\}, \{(x_j^t, y_j^t, l_j^t) : 1 \le j \le m\}$
**Output**: TSV positions $(x_{kb}^v, y_{kb}^v)$ where $1 \le k \le L-1, 1 \le b \le B$

1 $prt$ = a small integer (e.g., 5);
2 $d_x = W/prt$;
3 $d_y = H/prt$;
4 $Z = \{0, d_x, 2d_x, \ldots, W\} \times \{0, d_y, 2d_y, \ldots, H\}$;
5 $V_0$ = the best assignment $V \in Z^{(L-1)B}$ such that $TotShortestPaths(V)$ is smallest;
6 $X = \{x_i^s : 1 \le i \le n\} \cup \{x_j^t : 1 \le j \le m\}$;
7 $Y = \{y_i^s : 1 \le i \le n\} \cup \{y_j^t : 1 \le j \le m\}$;
8 **while** *true* **do**
9 $\quad update_flag = false$;
10 $\quad$ **for** $1 \le g \le (L-1)B$ **do**
11 $\quad\quad V_1$ =the best assignment $V$ with $V(g') = V_0(g')$ for $g' \ne g$ and $V(g) \in X \times Y$ such that $TotShortestPaths(V)$ is smallest;
12 $\quad\quad$ **if** $V_1 \ne V_0$ **then**
13 $\quad\quad\quad update_flag = true$;
14 $\quad\quad$ **end**
15 $\quad\quad V_0 = V_1$;
16 $\quad$ **end**
17 $\quad$ **if** $update_flag = false$ **then**
18 $\quad\quad$ break;
19 $\quad$ **end**
20 **end**
21 return $V_0$;

We describe the details of Algorithm 1 as follows:

- Lines 1-5: We partition the rectangle $W \times H$ into $prt \times prt$ parts with the size of each part to be $d_x \times d_y$. Let $Z$ be the set of lattice points in this partition. We first assume TSVs can only take positions from $Z$ and obtain the best assignment $V_0$ by exhaustive search in Line 5. $V_0$ will be updated to a better solution afterward in Lines 8-20.
- Lines 6-7: We define the Hanan grid formed by positions of masters and slaves as $X \times Y$.
- Lines 8-20: In each iteration of the while loop, if there exists a better solution $V_1$ than $V_0$, we update $V_0$ to be $V_1$ and continue the loop. Otherwise, we stop and output $V_0$ as the final solution.
- Lines 10-16: For each TSV $g$, let $V_1$ be the best assignment by moving TSV $g$ in $V_0$ to a new position as in Line 11. If $V_1$ is better than $V_0$, we update $V_0$ in Lines 12-15.

Based on our experimental results, the local search algorithm achieves nearly optimal solution by comparing with the lower bound obtained from the case where arbitrary TSVs are allowed between adjacent layers (i.e., $B = \infty$).

**Algorithm 2:** Greedy RSSG heuristic

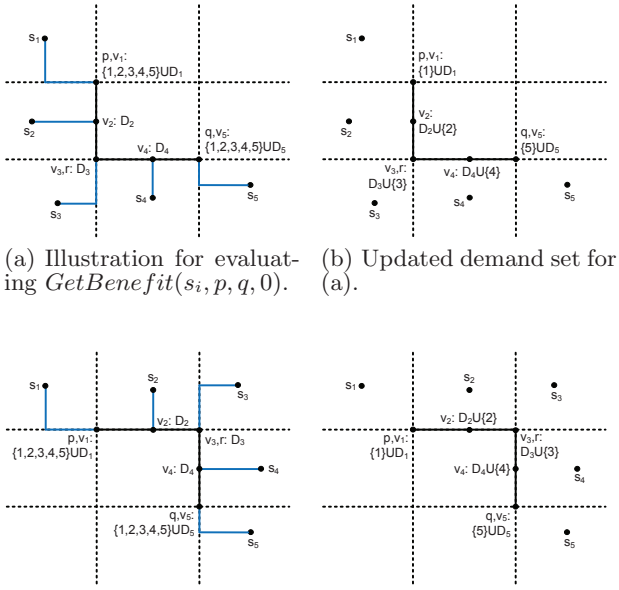
**Input**: A set of $n$ masters $s_1, \ldots, s_n$ and $m$ slaves $t_1, \ldots, t_m$ in rectilinear plane.
**Output**: A subgraph $G(V, E)$ of Hanan grid $H_g$ formed by masters and slaves so that $G$ contains a rectilinear shortest path for each master-slave pair.

1 $V = \{s_1, \ldots, s_n, t_1, \ldots, t_m\}$;
2 $E = \emptyset$;
3 $P =$ set of vertices in Hanan grid $H_g$;
4 $D_p = \{1, 2, \ldots, n\}$ for $p \in \{t_1, \ldots, t_m\}$;
5 $D_p = \emptyset$ for $p \in P$ and $p \notin \{t_1, \ldots, t_m\}$;
6 **while** $\exists p, q \in P$ *s.t.* $D_p \cap D_q \neq \emptyset$ **do**
7  $BestBenefit = 0$;
8  **for** $(p, q \in P$ *s.t.* $D_p \cap D_q \neq \emptyset)$ *and* $d \in \{0, 1\}$ **do**
9  $Benefit = \sum_{i \in D_p \cap D_q} GetBenefit(s_i, p, q, d)$;
10  **if** $Benefit > BestBenefit$ **then**
11  $BestBenefit = Benefit$;
12  $p^* = p,\ q^* = q,\ d^* = d$;
13  **end**
14  **end**
15  **for** $i \in D_{p^*} \cap D_{q^*}$ **do**
16  $v_i = GetSteinerPoint(s_i, p^*, q^*, d^*)$;
17  $V = V \cup \{v_i\}$;
18  $D_{p^*} = D_{p^*} \backslash \{i\}$, $D_{q^*} = D_{q^*} \backslash \{i\}$;
19  $D_{v_i} = D_{v_i} \cup \{i\}$;
20  **end**
21  $r = GetMergePoint(p^*, q^*, d^*)$;
22  $E = E \cup ShortestPath(p^*, r)$;
23  $E = E \cup ShortestPath(q^*, r)$;
24 **end**
25 **for** $p \in P$ *s.t.* $D_p \neq \emptyset$ **do**
26  **for** $i \in D_p$ **do**
27  $E = E \cup ShortestPath(s_i, p)$;
28  **end**
29 **end**
30 $RemoveRedundantEdges(G)$;
31 return $G(V, E)$;

# 4. PROBLEM TWO: APPROXIMATION ALGORITHMS FOR GENERATING RECTILINEAR A SHORTEST-PATH STEINER GRAPH

In this section, we build a network to connect masters, slaves and TSVs in each device layer. Since signals are delivered on one path with distributed multiplexers/demultiplexers, we formulate our problem as the RSSG synthesis first considered in [16]. We assume that locations of a set of masters $s_1, \ldots, s_n$ and slaves $t_1, \ldots, t_m$ in a fixed layer are given. Notice that TSVs connected with this layer are considered as both master and slave. A solution of RSSG is a rectilinear routing containing all master-to-slave rectilinear shortest paths, with total wire length as small as possible. RSSG is a generalization of the minimum rectilinear Steiner arborescence (RSA) problem [8][13] and a relaxation of the Minimum Manhattan network (MMN) problem [3], both of which have been proven to be NP-Complete in [14] and [7] respectively. The RSA problem corresponds to the case where there is only one master $s$. We will introduce two approximation algorithms for RSSG in the following subsections. One is a greedy heuristic which extends the insight of a 2-approximation algorithm given in [13] for solving the RSA problem. It requires only $O(nm)$ extra memory beyond inputs and easily handles large-scale RSSG instances. Our algorithm uses linear programming (LP) relaxation and rounding to solve small cases with higher accuracy. In practice, we observe that the solution of LP relaxation and rounding is within a factor 1.0005 of optimal. However, the LP formulation of RSSG has $O(nm(n+m)^2)$ number of variables and constraints which makes it suitable only for solving instances with $< 1000$ master-slave pairs on a computer with 4GB RAM.

## 4.1 Greedy Heuristic




(a) Illustration for evaluating $GetBenefit(s_i, p, q, 0)$.

(b) Updated demand set for (a).




(c) Illustration for evaluating $GetBenefit(s_i, p, q, 1)$.

(d) Updated demand set for (c).

**Figure 5: Contributions of two possible cases for merging $p$ and$q$.**

The greedy heuristic starts with $m$ slaves as $m$ subtrees and iteratively merges a pair of subtree roots $p^*$ and $q^*$ with merging point $r$ is farthest from masters. We can think of this heuristic as obtaining the largest possible benefit, i.e., the sum of Manhattan distances from masters to $p^*$ and $q^*$, in each iteration. Algorithm 2 provides the details of the greedy heuristic, with explanations as follows.

- Lines 1-3: The subgraph $G$ and the set of vertices $P$ of Hanan grid $H_g$ are initialized.
- Lines 4-5: For each slave $p \in \{t_1, \ldots, t_m\}$, we define a demand set $D_p$ initialized to be $\{1, \ldots, n\}$ which means $s_i - p$ rectilinear shortest path needs to be constructed

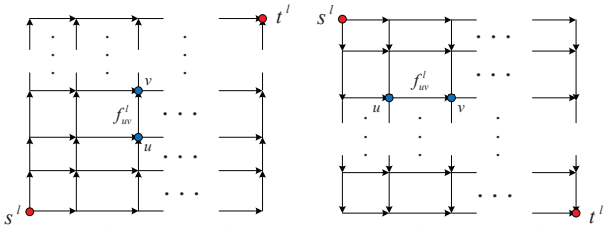
in graph $G$ for every $i \in D_p$. We also set the demand set to be empty for other non-slave points in $P$.

- Lines 6-14: In Line 6, if there exist two slaves or Steiner points $p$ and $q$ such that their demand sets have non-empty intersection, we will choose one such pair, i.e., $p^*$ and $q^*$, with largest benefit for merging them using direction $d^*$ between Lines 7 and 13. The function $GetBenefit(s_i, p, q, d)$ returns the benefit we can obtain in terms of $s_i$ by merging $p$ and $q$ using direction $d$. Without loss of generality, we assume that $p$ is above and to the left of $q$ as illustrated in Figure 5. Two possible directions of merging $p$ and $q$ are shown in Figures 5(a) and 5(c). The benefit obtained by $GetBenefit(s_i, p, q, d)$ is the length of the blue line connecting $s_i$ with a slave or Steiner point $v_i$. For other positions of $s_i$ other than those shown in Figure 5, the function $GetBenefit(s_i, p, q, d)$ will return zero. Notice that after $p$ and $q$ are merged, the rectilinear shortest paths from $s_i$ to $p$ and from $s_i$ to $q$ can share the path from $s_i$ to $v_i$; this is the benefit we achieve compared with connecting $s_i$ with $p$ and $q$ separately.
- Lines 15-20: We update the demand sets after merging $p^*$ and $q^*$ using direction $d^*$. For each master $s_i$ such that $i$ belongs to the intersection of $D_{p^*}$ and $D_{q^*}$, the slave or Steiner point $v_i$ is found by the function $GetSteinerPoint$ and we henceforth only need to connect $s_i$ with $v_i$ by a rectilinear shortest path, instead of connecting with $p^*$ and $q^*$. The corresponding update of demand sets is described in Lines 18-19 and illustrated in Figures 5(b) and 5(d).
- Lines 21-23: We get the merging point $r$ and update the graph $G$ according to the merging operation. The function $ShortestPath$ gives a rectilinear shortest path between two points using smallest extra edges not in $G$. It is implemented by a simple dynamic programming algorithm.
- Lines 25-29: When there are no intersecting demand sets, we fulfill the remaining connection requirements.
- Lines 30: We delete all redundant edges in $G$, i.e., as long as removing them does not change the length of any $s - t$ shortest path.

## 4.2 LP Relaxation and Rounding

In this section, we first give an integer linear programming (ILP) formulation of the RSSG problem and relax it into an instance of general linear programming (LP). Then, we devise a rounding technique to obtain an approximation solution of RSSG from the optimal solution of the LP relaxation. By comparing with the objective value of the LP relaxation, our results show that this approach achieves nearly optimal solutions.

The ILP formulation for the RSSG problem is given in (6) and the details are described as follows. Suppose there are $Q$ pairs of masters and slaves where $Q = n \times m$. We use $(s^l, t^l)$ to denote the $l^{th}$ master-slave pair. To transform the RSSG problem into an ILP, we construct a directed graph $N_l$ as in Figure 6 for $(s^l, t^l)$ on the Hanan grid formed by masters and slaves. Without loss of generality, assume $s^l$ is to the left of $t^l$. Figures 6(a) and 6(b) show the directed graph $N_l$ for the cases where $s^l$ is below and above $t^l$, respectively. We notice that an $s^l - t^l$ rectilinear shortest path exists



(a) $s^l$ is below $t^l$. (b) $s^l$ is above $t^l$.

**Figure 6: Directed network $N_l$ on the Hanan grid.**

if and only if an integer $s^l - t^l$ flow with value one exists in $N_l$. Let $E_l$ be the set of arcs of $N_l$ and let $f^l_{uv}$ be a variable denoting the flow from $u$ to $v$ where $(u, v) \in E_l$. We use the first three constraints in (6) to guarantee the existence of a valid flow for each $(s^l, t^l)$ pair. Now let $E_H$ be the set of undirected edges in the Hanan grid and let the binary variable $x_{uv}$ denote whether the edge $(u, v) \in E_H$ is selected in the RSSG solution. If a rectilinear shortest path from $s^l$ to $t^l$ includes the edge $(u, v)$ for any $1 \le l \le Q$, then $x_{uv} = 1$. We use the fourth constraint in (6) to denote such constraints. Finally, if $d_{uv}$ denotes the length of edge $(u, v)$, our objective is to minimize the total wire length which can be described by the objective function of (6). Since there are $O((n + m)^2)$ edges in the Hanan grid and $O(nm)$ master-slave pairs, it is easy to see that the ILP formulation contains $O(nm(n + m)^2)$ variables and $O(nm(n + m)^2)$ constraints.

$$
\begin{aligned}
\min \quad & \sum_{(u,v)\in E_H} d_{uv} x_{uv} \\
s.t. \quad & \sum_{(u,v)\in E_l} f^l_{uv} - \sum_{(v,u)\in E_l} f^l_{vu} \ge 0, \\
& 1 \le l \le Q \text{ and } v \ne s^l, t^l; \\
& \sum_{(s^l,u)\in E_l} f^l_{s^l u} \ge 1, \quad 1 \le l \le Q; \\
& \sum_{(u,t^l)\in E_l} f^l_{u t^l} \ge 1, \quad 1 \le l \le Q; \\
& x_{uv} - f^l_{uv} \ge 0, \\
& (u, v) \in E_H \text{ and } l \in \{l_0 : (u, v) \in E_{l_0}\}; \\
& f^l_{uv} \in \{0, 1\}, \quad 1 \le l \le Q \text{ and } (u, v) \in E_l; \\
& x_{uv} \in \{0, 1\}, \quad (u, v) \in E_H. \qquad (6)
\end{aligned}
$$

Since solving ILP (6) is time-consuming and even intractable when $n$ and $m$ are large, we propose a LP relaxation and rounding technique described in Algorithm 3. We begin by relaxing the binary constraints on variables $f^l_{uv}$ and $x_{uv}$ so that they can take any nonnegative real values. We thus obtain an LP instance that can be solved efficiently. The variables $x_{uv}$ in the optimal solution of the LP relaxation are in the range $[0, 1]$ based on other constraints, but may be fractional, which does not yield a feasible RSSG solution. To construct an RSSG from the LP solution, we adopt usual approach of viewing each $x_{uv}$ as the probability of selecting edge $(u, v)$. Hence, we start with a graph $G$ containing all edges in the Hanan grid and try to delete edges as long as the remaining graph still contains a rectilinear shortest path for each master-slave pair. The order of deletion depends on the optimal $x_{uv}$ values. An edge with smaller $x_{uv}$ will be

deleted from graph $G$ earlier if possible. Finally, the remaining graph $G$ will be a feasible RSSG solution. The optimal objective function value of the LP relaxation lower-bounds the optimal objective function value of the ILP (6), and hence optimal RSSG solution. We can evaluate the quality of the rounding solution by comparing it with the lower bound; experimental results below show that the largest ratio between them in practice is 1.0005.

**Algorithm 3:** LP relaxation and rounding for RSSG

**Input**: A set of $n$ masters $s_1, \ldots, s_n$ and $m$ slaves $t_1, \ldots, t_m$ in the rectilinear plane.
**Output**: A subgraph $G(V, E)$ of the Hanan grid $H_g$ with edge set $E_H$, such that $G$ contains a rectilinear shortest path for each master-slave pair.

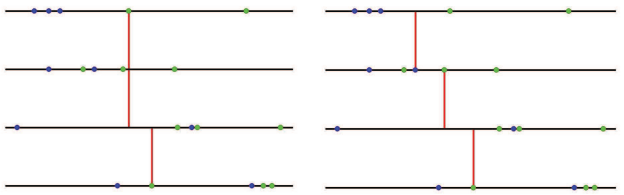
```
V = {s_1, ..., s_n, t_1, ..., t_m};
E = E_H;
n_e = |E_H|;
Solve LP relaxation of (6) to obtain x_uv for every (u, v) ∈ E_H;
(e_1, e_2, ..., e_{n_e}) = edges in E_H sorted from smallest to largest according to the value of x_uv;
for e = e_1, e_2, ..., e_{n_e} do
    if E − e contains a rectilinear shortest path for each master-slave pair then
        E = E − e;
    end
end
return G(V, E);
```

# 5. EXPERIMENTAL RESULTS

## 5.1 TSV Location

We first show the TSV location results with different communication frequencies and different upper bounds $B$ on the number of available TSVs between adjacent layers. Let $L$ be the number of device layers. We randomly distribute $N$ masters and slaves in each layer. Figure 7 shows two optimal assignments of TSVs with different communication frequencies and the same distribution of masters and slaves. TSV locations are plotted by red lines. Blue dots and green dots denote masters and slaves respectively. For clarity, we simplify each layer as a straight line. Here we assume $B = 1$ and the dynamic programming approach is adopted to obtain optimal solutions. We notice that in Figure 7(b), TSV locations connecting the first three layers are to the left of those in Figure 7(a). At the same time, masters in bottom two layers are farther away from slaves in the top layer in Figure 7(b). These observations reflect the fact that master-slave pairs in the first two layers communicate more frequently.

Table 1 compares the power consumption results for various bounds $B$. We assume that communication frequencies for all master-slave pairs are the same in this experiment. The power consumption is defined as the total length of shortest paths normalized by the number of master-slave pairs. Notice that when $B = \infty$, every master-slave pair can be reached by a path with shortest rectilinear distance which gives us a lower bound on power consumption. In each cell of



(a) $c_{i,j} = 1.0$ for all master-slave pairs $(s_i, t_j)$.

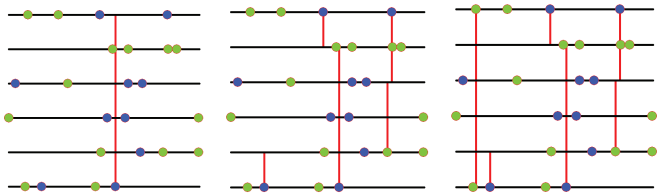
(b) $c_{i,j} = 5.0$ for master-slave pairs $(s_i, t_j)$ in first two layers and $c_{i,j} = 1.0$ for other pairs.

**Figure 7: Single TSV assignments for $(L, N) = (4, 5)$.**

the table, the first value represents the power consumption and the second value is the increase of (as a percentage) of power consumption relative to the case $B = \infty$ in the same row. We can see that the lower bound can be almost achieved by only using three TSVs between adjacent layers. Figure 8 gives an example of the TSV assignment results along with their power consumption for bound $B = 1, 2, 3$ on the same setup of masters and slaves.

**Table 1: Power consumption with varying $B$.**

| $(L, N)$ | $B = \infty$ | $B = 1$ | $B = 2$ | $B = 3$ |
|---|---|---|---|---|
| (3,10) | 346.36 | 456.46 (31.79%) | 375.93 (8.54%) | 360.86 (4.19%) |
| (3,20) | 292.39 | 384.34 (31.45%) | 336.13 (14.96%) | 305.50 (4.49%) |
| (3,50) | 337.27 | 453.09 (34.34%) | 393.07 (16.55%) | 355.22 (5.32%) |
| (4,20) | 343.93 | 476.15 (38.45%) | 413.28 (20.16%) | 367.62 (6.89%) |
| (5,20) | 346.68 | 492.00 (41.92%) | 431.04 (24.33%) | 367.39 (5.97%) |
| (5,50) | 323.76 | 452.50 (39.77%) | 401.23 (23.93%) | 346.31 (6.96%) |



(a) Single TSV - 439.

(b) Two TSVs - 395.

(c) Three TSVs - 348.

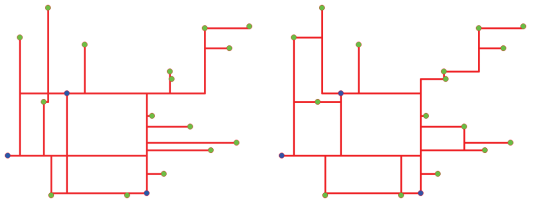
**Figure 8: Wire length with varying TSV budgets.**

## 5.2 RSSG Construction

Table 2 gives the total wire length improvement of our algorithms when compared with the RSSG result in [16]. The column "LP(obj)" denotes the optimal objective value of the LP relaxation in (6). The column "LP(Round)" denotes the total wire length of the LP relaxation and rounding result. Notice that in most cases, the rounding result and the optimal LP objective are the same, which means that we already obtain the optimal RSSG (since LP relaxation gives a

Table 2: Total wire length results.

| $(n, m)$ | RSSG | Greedy | LP(Obj) | LP(Round) | Ratio |
|---|---|---|---|---|---|
| $(3, 16)$ | 5388 | 5068 | 4882 | 4882 | 9.39% |
| $(5, 15)$ | 7024 | 6586 | 6342 | 6342 | 9.71% |
| $(12, 6)$ | 6698 | 6306 | 5915 | 5915 | 11.69% |
| $(6, 12)$ | 7127 | 6160 | 5575 | 5575 | 21.78% |
| $(12, 12)$ | 11559 | 10385 | 10319 | 10319 | 10.73% |
| $(20, 20)$ | 16236 | 15921 | 13847 | 13847 | 14.71% |
| $(30, 30)$ | 302619 | 287042 | 251841 | 251968 | 16.78% |



(a) Greedy heuristic. (b) LP relaxation and rounding.

**Figure 9: Comparison of RSSG results.**

lower bound of smallest total wire length). In practice during our experiments, the largest gap between "LP(round)" and "LP(obj)", i.e., LP(round)/LP(obj), is 1.0005 which indicates that our rounding method can achieve nearly optimal solutions. The column "Ratio" gives the improvement of our RSSG generated by LP relaxation and rounding relative to RSSG generated by the algorithm in [16]. We can see that our algorithm can reduce the total wire length by up to 21.78%. Figure 5.2 plots examples of our RSSG solutions generated by the greedy heuristic and by LP relaxation and rounding. There are three masters and sixteen slaves, which are represented by blue dots and green dots, respectively. The LP relaxation and rounding method for this example achieves an optimal RSSG with total wire length of 5683, 5.38% better than the greedy solution whose total wire length is 6006. Table 3 shows the time and memory complexity of our LP relaxation and rounding algorithm. We use the Gurobi Optimizer [1] running on a machine with Intel Core i3 2.4GHz CPU and 4GB memory to solve LP relaxation of (6). Based on the memory limit, we can solve RSSG instances with up to 1000 master-slave pairs by LP relaxation and rounding.

Table 3: Scaling of LP relaxation and rounding.

| $(n, m)$ | # Variables | # Constraints | Time | Memory |
|---|---|---|---|---|
| $(3, 16)$ | 4326 | 6209 | <1s | <50MB |
| $(5, 15)$ | 8019 | 11458 | <1s | <50MB |
| $(20, 20)$ | 155396 | 239037 | 3m43s | 234MB |
| $(30, 30)$ | 778358 | 1175974 | 4h22m | 1.2GB |

# 6. CONCLUSION

In this paper, we construct a framework and algorithms for synthesis of gated buses in 3D ICs. The power consumption of on-chip communication is considered to be the first objective and we achieve it by optimizing TSV locations between device layers. In each device layer, we build a rectilinear shortest-path Steiner graph to connect masters and slaves whose objective is to minimize the total wire length. Experimental results show that our dynamic programming (resp. local search) algorithm efficiently determines the optimal (resp. nearly optimal) TSV locations, and that the approximation algorithms for generating RSSG reduce the total wire length by up to 22% compared with [16].

# 7. ACKNOWLEDGMENTS

The authors would like to acknowledge the support of NSF CCF-1017864 and CCF-1116667, the DARPA/MARCO Gigascale Systems Research Center, as well as SRC and DOE support. C.K. Cheng would like to acknowledge the support of National Science Council of Taiwan Grant No.: NSC 100-2811-E-002-034. We thank Prof. Ion Mandoiu of the University of Connecticut for pointers to the MMN problem and related references.

# 8. REFERENCES

[1] Gurobi optimizer 4.0. `http://www.gurobi.com/`.

[2] K. Bazargan, R. Kastner and M. Sarrafzadeh. 3-D floorplanning: simulated annealing and greedy placement methods for reconfigurable computing systems. *IEEE Int. Workshop on Rapid System Prototyping*, pp. 38–43, 2002.

[3] M. Benkert, A. Wolff, F. Widmann and T. Shirabe. The minimum Manhattan network problem: approximations and exact solutions. *Computational Geometry: Theory and Applications*, 35(3), pp. 188–208, 2006.

[4] B. Black, D. W. Nelson, C. Webb and N. Samra. 3D processing technology and its impact on IA32 microprocessors. *Int. Conference on Computer Design*, pp. 316–318, 2004.

[5] J. Y. Chen, W. B. Jone, J. S. Wang, H. I. Lu and T. Chen. Segmented bus design for low power system. *IEEE Trans. on VLSI Systems*, 7(1), pp. 25–29, 1999.

[6] L. Cheng, L. Deng and M. Wong. Floorplanning for 3-D VLSI design. *Asia and South Pacific Design Automation Conference*, pp. 405–411, 2005.

[7] F. Y. Chin, Z. Guo and H. Sun. Minimum Manhattan network is NP-complete. *Proc. of the 25th Annual Symposium on Computational Geometry*, pp. 393–402, 2009.

[8] J. Cong, A. B. Kahng and K. S. Leung. Efficient algorithms for the minimum shortest path Steiner arborescence problem with applications to VLSI physical design. *IEEE Trans. on Computer-Aided Design*, 17(1), pp. 24–39, 1998.

[9] K. D. Boese, A. B. Kahng, B. A. McCoy and G. Robins. Near-optimal critical sink routing tree constructions *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 14(12), pp. 1417–1436, 1995.

[10] M. Hanan. On Steiner's problem with rectilinear distance. *SIAM Journal on Applied Mathematics*, 14, pp. 255–265, 1966.

[11] C.-T. Hsieh and M. Pedram. Architectural power optimization by bus splitting. *Design, Automation and Test in Europe*, pp. 612–616, 2000.

[12] M. Pathak, Y. Lee, T. Moon and S. Lim. Through-silicon-via management during 3D physical design: when to add and how many? *Int. Conference on Computer-Aided Design*, 2010.

[13] S. K. Rao, P. Sadayappan, F. K. Hwang and P. W. Shor. The rectilinear Steiner arborescence problem. *Algorithmica*, 7, pp. 277–288, 1992.

[14] W. Shi and C. Su. The rectilinear Steiner arborescence problem is NP-complete. *Proc. of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 780–787, 2000.

[15] V. V. Vazirani. *Approximation algorithms*. Springer, 2010.

[16] R. Wang, N. C. Chou, B. Salefski and C. K. Cheng. Low power gated bus synthesis using shortest-path Steiner graph for system-on-chip communications. *Design Automation Conference*, pp. 166–171, 2009.