Improved Performance of 3DIC Implementations Through Inherent Awareness of Mix-and-Match Die Stacking

Kwangsoo Han[†], Andrew B. Kahng[†+] and Jiajia Li[†]
UC San Diego, [†]ECE and [+]CSE Depts., La Jolla, CA 92093, {kwhan, abk, jil150}@ucsd.edu

***Abstract*—3D logic-logic integration is an important future lever for continued cost and density scaling value propositions in the semiconductor industry. In the 3DIC context, several works have proposed "mix-and-match" of multiple stacked die, according to binning information, to improve overall product yield. However, each of the stacked die in these works is independently designed: there is no holistic "design for eventual stacking" of any of the die. Separately, many approaches have been proposed for design partitioning and implementation with multiple die, including 3D stacked-die implementation. However, the signoff criteria used to implement such a multi-die solution must necessarily validate timing correctness for all combinations of process conditions on the multiple die. To our knowledge, no previous work has examined the fundamental issue of design partitioning and signoff specifically for mix-and-match die stacking. In this work, we study performance improvements of 3DIC implementation that leverage knowledge of mix-and-match die stacking during manufacturing. We propose partitioning methodologies to partition timing-critical paths across tiers to explicitly optimize the signed-off timing across the reduced set of corner combinations that can be produced by the stacked-die manufacturing. These include both an ILP-based methodology and a heuristic with novel maximum-cut partitioning, solved by semidefinite programming, and a signoff timing-aware FM optimization. We also extend two existing 3DIC implementation flows to incorporate mix-and-match-aware partitioning and signoff, demonstrating the simplicity of adopting our techniques. Experimental results show that our optimization flow achieves up to 16% timing improvement as compared to the existing 3DIC implementation flow in the context of mix-and-match die stacking.**

I. Introduction

Small footprint and high transistor density in three-dimensional integrated circuits (3DICs) make 3D logic-logic integration an important future lever for cost and density scaling. Specific to 3DICs, a number of works [2] [5] [7] [12] have pointed out that "mix-and-match" of multiple stacked die, according to binning information, can improve overall product yield.[1] Without loss of generality, assuming that dies are classified into two process bins, SS and FF, the example in Figure 1 shows that mix-and-match die stacking can offer 75ps timing improvement for a small 28nm FDSOI block as compared to the conventional worst-case analysis.[2] However, in the previous works each of the stacked die is independently designed, that is, there is no holistic "design for eventual stacking" of any of the die.

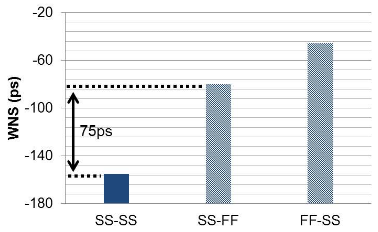


Fig. 1: Worst negative slack (WNS) of design AES [25] at 28FDSOI technology. Clock period = 1.2ns. SS-SS, SS-FF and FF-SS respectively indicate SS Tier 0 + SS Tier 1, SS Tier 0 + FF Tier 1 and FF Tier 0 + SS Tier 1. Mix-and-match die stacking offers 75ps improvement of WNS over the conventional worst-case analysis. In this example, the AES implementation was simply bipartitioned for minimum net cut using MLPart [1] [24].

Separately, many works [3] [10] [11] [18] [20] [21] have suggested approaches for partitioning of logic into multiple die, e.g., to obtain the wirelength (hence, power and delay) savings implied by implementing a 1 x 1 die area into two stacked 0.7 x 0.7 dies. However, the signoff criteria used to implement such a multi-die solution must necessarily validate timing correctness for all combinations of process conditions on the multiple die – e.g., the four combinations { SS Tier 0 + SS Tier 1, SS Tier 0 + FF Tier 1, FF Tier 0 + SS Tier 1, FF Tier 0 + FF Tier 1 }.[3] Satisfying this combinatorial number of signoff constraints induces area and power overheads as a result of the sizing and buffering operations needed to close timing.

To our knowledge, no previous work has examined the fundamental issue of *design partitioning and signoff specifically for mix-and-match die stacking*. In particular, if we know *a priori* that, say, SS Tier 0 and SS Tier 1 die will never be stacked together, or that FF Tier 0 and FF Tier 1 die will never be stacked together, this changes our signoff criteria. Even more, this *a priori* knowledge allows us to partition timing-critical paths across tiers to explicitly optimize the design's performance in the regime of mix-and-match stacking. The simple example in Figure 2, explained in the figure caption, illustrates how the partitioning solution can impact design signoff timing in the regime of mix-and-match stacking.

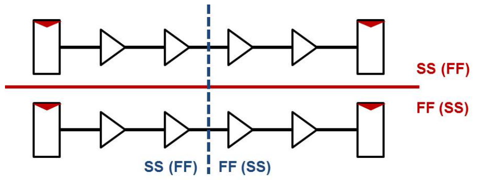


Fig. 2: Partitioning solutions affect a design's performance in the regime of mix-and-match stacking. Assuming that SS Tier 0 + FF Tier 1 and FF Tier 0 + SS Tier 1 are utilized for die stacking, the partitioning solution indicated by the blue dotted line has the maximum timing slack, while the partitioning solution indicated by the red solid line has the minimum timing slack.

In this work, we propose partitioning methodologies and signoff flows that are aware of mix-and-match die stacking to improve design timing (i.e., to improve worst negative slack (WNS)). However, 3D partitioning for mix-and-match die stacking is nontrivial. First, the optimal cut locations on one timing path might conflict with those on other timing paths. Thus, the partitioning optimization must trade off timing optimizations among timing paths. This can be quite challenging in a design with a large number of potentially critical paths and shared logic cones among multiple pairs of timing startpoints-endpoints. Further, the partitioning optimization must comprehend the timing impact of *vertical interconnects* or *VIs* (i.e., the vertical electrical connections (vias) between tiers, such as through-silicon vias), and can no longer "freely" partition a timing path into segments. In addition, delay variations across different process conditions can be different for cells of different types (e.g., INV, NAND or NOR), sizes and VT flavors. Last, asymmetric distribution of process bins (e.g., 3σ FF + 2σ SS) as discussed in [14] will also increase the difficulty of the partitioning optimization. Figure 3 shows a simple example with different optimal partitioning solutions that respectively minimize (a) delay of path A-C, (b) delay of path B-C, and (c) the worst case over the two paths. Moreover, the optimal partitioning solution changes with increased VI delay impact, as shown in Figure 3(d).

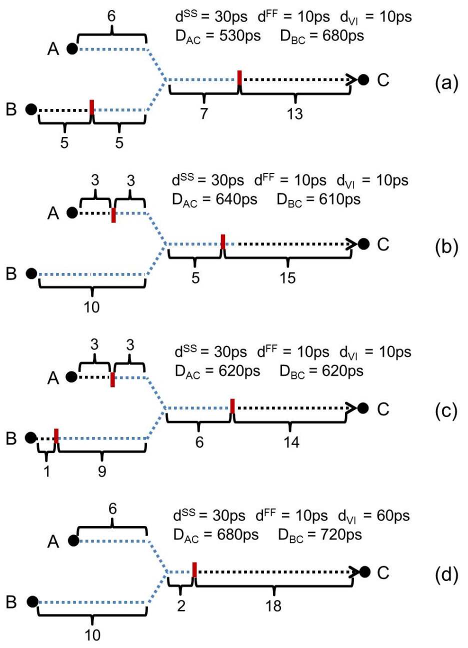
Our contributions in this work are as follows.

- We are the first to study design-stage optimization specifically for mix-and-match die stacking.
- We develop partitioning methodologies that are inherently aware of mix-and-match die stacking. Our approaches achieve up to 16% timing improvement as compared to a min-cut based partitioning approach.
- We extend the existing 3DIC implementation flows to incorporate mix-and-match-stacking-aware partitioning and signoff, demonstrating the simplicity of adopting our techniques.

[1]The mix-and-match stacking optimization is also applicable to wafer-to-wafer bonding integration where SS wafers are integrated with FF wafers, and to monolithic 3D integration with adaptive adjustment of the top-tier process according to the bottom-tier process condition. For simplicity, we use "(die) stacking" to refer collectively to these multiple contexts.

[2]In the following discussions and our experiments, we assume that dies are classified into two process bins, SS and FF. However, given matched pairs of process bins based on die-level and/or wafer-level stacking optimization, our approaches can be extended to scenarios with > 2 process bins, e.g., additional combinations can be { SS Tier 0 + TT Tier 1, TT Tier 0 + SS Tier 1, FF Tier 0 + TT Tier 1, TT Tier 0 + FF Tier 1, TT Tier 0 + TT Tier 1 } when we also consider the TT process bin.

[3]Here, a *tier* refers to one stacked die in a 3DIC. In a two-tier 3DIC, Tier 0 is the bottom tier and Tier 1 is the top tier.



d^{SS} = 30ps d^{FF} = 10ps d_{VI} = 10ps
D_{AC} = 530ps D_{BC} = 680ps
A
B
C
6
5
5
7
13
(a)
d^{SS} = 30ps d^{FF} = 10ps d_{VI} = 10ps
D_{AC} = 640ps D_{BC} = 610ps
A
B
C
3
3
10
5
15
(b)
d^{SS} = 30ps d^{FF} = 10ps d_{VI} = 10ps
D_{AC} = 620ps D_{BC} = 620ps
A
B
C
3
3
1
9
6
14
(c)
d^{SS} = 30ps d^{FF} = 10ps d_{VI} = 60ps
D_{AC} = 680ps D_{BC} = 720ps
A
B
C
6
10
2
18
(d)

Fig. 3: Area-balanced partitioning solutions on path A-C (26 stages) and path B-C (30 stages) which respectively minimize (a) delay of path A-C (D_{AC}), (b) delay of path B-C (D_{BC}), (c) worst-case delay over the two paths, and (d) worst-case delay over the two paths in the regime of large VI delay impact (d_{VI}). Red bars are VIs. We assume the same stage delay (30ps at SS, 10ps at FF) for every stage in the two paths. Timing analysis is aware of mix-and-match stacking (i.e., { SS Tier 0 + FF Tier 1, FF Tier 0 + SS Tier 1 }) and assumes ideal clock.

II. Related Works

We classify related works into two categories: (i) mix-and-match die stacking optimization, and (ii) 3D netlist partitioning.

Mix-and-match optimization. Several works propose approaches for mix-and-match die stacking optimization. Ferri et al. [5] propose methodologies to benefit from the flexibility of die-to-die and/or die-to-wafer 3D integration with awareness of the inter-die process variation. Their optimization improves performance and parametric yield of 3DICs with one CPU die and one L2 cache die. Garg et al. [7] formulate mathematical programs to improve the performance yield of 3DICs via mix-and-match die stacking. Chan et al. [2] propose an integer linear programming-based method as well as a heuristic method to optimize reliability of 3DICs (i.e., to improve the mean time to failure). To avoid the large runtime of thermal simulation, Juan et al. [12] develop a learning-based model for temperature prediction in 3DICs. Based on the model, they perform thermal-aware matching and stacking of dies to improve thermal yield. These optimization approaches operate at die level or wafer level (essentially, post-manufacturing). By contract, our work addresses design-stage optimization and signoff for mix-and-match die stacking.

3D netlist partitioning. As mentioned above, quite a few works study 3D partitioning. Li et al. [18] use a simulated annealing engine to partition blocks across tiers during the floorplanning stage to minimize wirelength. Several works cast 3D partitioning as a form of standard hypergraph partitioning. Thorolfsson et al. [21] use hMetis [16] to partition the design into balanced halves while minimizing the number of cuts. A multilevel partitioning methodology is proposed in [10], which first applies Hyperedge Coarsening (HEC) techniques to coarsen the netlist, then performs an FM-like K-way partitioning procedure to partition the netlist such that the number of VIs is minimized. An integer linear programming for 3D partitioning is formulated in [11], where the objective is to reduce the number of VIs subject to area balancing constraints. Partitioning methodologies based on an initial 2D implementation solution are also proposed in previous literatures. Cong et al. [3] assign cells to tiers through folding-based transformations of an initial 2D placement solution. Based on a 2D implementation solution with scaled dimension (i.e., 0.7 x), Panth et al. [20] perform routability-driven partitioning to minimize the overall routing overflow; this can mitigate routing congestion and help minimize wirelength. Compared to these works, our work is again distinguished by being the first to inherently comprehend mix-and-match die stacking integration. In particular, unlike previous works, our partitioning methods directly maximize the design's timing slack in the mix-and-match regime.

III. Problem Formulation

We formulate the partitioning problem for mix-and-match die stacking as follows.

Given: post-synthesis netlist, Liberty files according to various process bins, vertical interconnect (VI) parasitics, timing constraints and area balancing criteria,

Perform: 3D partitioning to determine the tier index for each cell, such that the worst timing slack is maximized in the context of mix-and-match die stacking.[4]

In the next section, we describe an ILP-based partitioning methodology which is able to achieve near-optimal solutions. Section V then proposes a heuristic partitioning methodology in which we (i) perform *maximum-cut* partitioning on the subgraph of the sequential graph that is induced by timing-critical pairs of startpoints and endpoints, then (ii) apply a signoff timing-aware FM optimization for further slack improvement.

IV. ILP-Based Partitioning Methodology

We now formulate an integer linear program (ILP) to partition the netlist into two tiers such that the worst timing slack, over the corner combinations that can be formed by mix-and-match stacking, is maximized. Table I summarizes our notations.

TABLE I: Description of notations used in our work.

Term	Meaning
α_j	process condition (corner), $(1 \leq j \leq J)$
P	set of timing paths
p_k	k^{th} timing path $(p_k \in P)$
C	set of cells
c_i	i^{th} cell $(c_i \in C)$
a_i	area of cell c_i
y_i	binary indicator whether cell c_i is on Tier 0 ($y_i = 0$) or on Tier 1 ($y_i = 1$)
$\beta_{i,i'}$ $(\beta_{i',i})$	binary indicator whether a cut (VI) exists between adjacent cells c_i and $c_{i'}$, where cell c_i is on Tier 0 (Tier 1) while cell $c_{i'}$ is on Tier 1 (Tier 0).
d_i^j	stage delay of cell c_i and its fanout wire at α_j
D_k	maximum delay of path p_k over all pairs of process corners
D_{max}	maximum delay over all paths among all pairs of process corners
d_{VI}	delay impact of VI insertion
θ	area balancing criterion

$$Minimize \quad D_{max}$$

$$Subject\ to$$

$$\beta_{i,i'} \geq y_{i'} - y_i \qquad \forall \text{ adjacent cells } c_i, c_{i'} \in C \tag{1}$$

$$\beta_{i',i} \geq y_i - y_{i'} \qquad \forall \text{ adjacent cells } c_i, c_{i'} \in C \tag{2}$$

$$\beta_{i,i'} + \beta_{i',i} \leq 1 \qquad \forall \text{ adjacent cells } c_i, c_{i'} \in C \tag{3}$$

$$\sum_{c_i \in p_k} (d_i^j \cdot (1 - y_i) + d_i^{j'} \cdot y_i) + \sum_{adjacent\ c_i, c_{i'} \in p_k} (\Delta_{i'}^{j,j'} \cdot \beta_{i,i'} + \Delta_{i'}^{j',j} \cdot \beta_{i',i})$$
$$+ \sum_{adjacent\ c_i, c_{i'} \in p_k} (\beta_{i,i'} + \beta_{i',i}) \cdot d_{VI} \leq D_k \quad \forall (\alpha_j, \alpha_{j'}),\ p_k \in P \tag{4}$$

$$D_k \leq D_{max} \qquad \forall p_k \in P \tag{5}$$

$$\sum_{c_i \in C} a_i \cdot y_i - \sum_{c_i \in C} a_i \cdot (1 - y_i) \leq \theta \cdot \sum_{c_i \in C} a_i \tag{6}$$

$$\sum_{c_i \in C} a_i \cdot (1 - y_i) - \sum_{c_i \in C} a_i \cdot y_i \leq \theta \cdot \sum_{c_i \in C} a_i \tag{7}$$

Our objective is to minimize the maximum path delay D_{max} over all paths $p_k \in P$, across all relevant pairs of process corners in the context of mix-and-match die stacking. y_i is a binary indicator of cell c_i's tier assignment, with $y_i = 0$ (resp. $y_i = 1$) indicating that c_i is on Tier 0 (resp. Tier 1). For any pair of adjacent cells c_i and $c_{i'}$, we use Constraints (1) and (2) to force either $\beta_{i,i'}$ or $\beta_{i',i}$ to be one when cells c_i and $c_{i'}$ are on different tiers. In other words, $\beta_{i,i'}$ and $\beta_{i',i}$ are indicators of a cut (or VI) such that $\beta_{i',i} = 1$ (resp. $\beta_{i,i;} = 1$) when c_i is on Tier 0 (resp. Tier 1) while $c_{i'}$ is on Tier 1 (resp. Tier 0). Therefore, $\beta_{i,i'}$ and $\beta_{i',i}$ are mutually exclusive.

[4]In this paper, we only consider partitioning into two-tier 3DICs. But, our formulation generalizes easily to larger numbers of tiers.

Constraint (4) defines the maximum delay D_k for each path $p_k \in P$ among all pairs of process corners with mix-and-match stacking. The first term on the left-hand side of Constraint (4) is the sum of stage delays along path p_k. We extract stage delays at a particular corner α_j based on the timing analysis assuming all cells are at α_j. However, such an assumption can lead to an inaccurate stage delay estimation because cells of different process corners output different slews, which affect the delays of downstream cells. For example, our assumption can be pessimistic for a cell at SS when its driver is at FF. This is because to estimate the stage delay at SS, our timing analysis assumes all cells (including its driver) are at SS, which results in pessimistic input slew estimation. To compensate for such inaccuracy, we pre-calculate the delta stage delays (that is, the second term) between the case where the driver cell c_i and driven cell $c_{i'}$ are at different process corners (i.e., c_i is at α_j, and $c_{i'}$ is at $\alpha_{j'}$) versus the case where the c_i and $c_{i'}$ are at the same process corner.[5] We denote such delta stage delays as $\Delta_{i'}^{j,j'}$. Incorporating the second term, i.e., the sum of delta stage delays along path p_k, enables us to achieve a more accurate delay estimation.[6] The third term on the left-hand side of Constraint (4) accounts for VI delay impact along the path. Note that VI insertion at the output pin of a small-size cell can have quite large delay impact. However, such delay impact will be addressed with sizing/VT-swapping optimization during the P&R (placement and routing) flow. Since no sizing/VT-swapping optimization is involved during the partitioning stage, to avoid pessimism in estimation of VI delay impact, we simply use a constant value to estimate the delay impact of one VI insertion. In Constraint (5), we obtain the maximum delay D_{max} over all paths $p_k \in P$. Last, our formulation satisfies area balancing criteria which are indicated by θ in Constraints (6) and (7). We set θ as 5% in our experiments.

V. Heuristic Partitioning Methodology

Although the ILP-based methodology can achieve near-optimal partitioning solutions, its runtime can be large. Moreover, it is practically impossible to extract all timing paths for a large design.[7] We therefore propose a timing-aware FM partitioning methodology with better scalability. Our heuristic partitioning methodology contains two optimization stages – (i) the global optimization performs *maximum cut* on the timing-critical sequential graph (i.e., a partial sequential graph which contains only startpoints and endpoints of timing-critical paths) and (ii) the incremental optimization performs timing-aware multi-phase Fiduccia-Mattheyses (FM) optimization to achieve the final partitioning solution. Unlike previous works which minimize the number of cuts [16] or the number of paths passing across different partitions [15], we directly target the timing slack improvement during our partitioning optimization. Our objective is to minimize the maximum path delay (i.e., maximize the worst timing slack) for mix-and-match die stacking. Further, we show that a maximum-cut partitioning is more suitable than the traditional minimum-cut partitioning for 3DICs in the mix-and-match regime. To our knowledge, few if any previous works have applied a semidefinite program-based maximum cut optimization [8] to VLSI design.

A. *Maximum-Cut Partitioning on Timing-Critical Sequential Graph*

We first study the tradeoff between delay impact of VI insertions versus timing improvement from mix-and-match stacking. Without loss of generality, we assume a die stacking of { SS Tier 0 + FF Tier 1, FF Tier 0 + SS Tier 1 }. We denote the path delay of path p_k at SS (resp. FF) as D_k^{SS} (resp. D_k^{FF}), and the total number of stages along p_k as l_k. Approximating the path delay as a linear function of the stage number and assuming that there are l'_k stages on Tier 0, the corresponding path delay without considering delay impact of VI insertion can be estimated as

$$l'_k \cdot \frac{D_k^{SS}}{l_k} + (l_k - l'_k) \cdot \frac{D_k^{FF}}{l_k} \tag{8}$$

$$l'_k \cdot \frac{D_k^{FF}}{l_k} + (l_k - l'_k) \cdot \frac{D_k^{SS}}{l_k} \tag{9}$$

where (8) assumes the stacking of SS Tier 0 + FF Tier 1, and (9) assumes the stacking of FF Tier 0 + SS Tier 1. Maximizing the minimum value between (8) and (9) corresponds to having (8) = (9) and $l'_k = l_k/2$. We therefore estimate the timing improvement from mix-and-match stacking over the worst-case analysis (i.e., SS Tier 0 + SS Tier 1) as $(D_k^{SS} - D_k^{FF})/2$. Furthermore, we denote the worst slack of p_k among combinations of process conditions (i.e., { SS Tier 0 + FF Tier 1, FF Tier 0 + SS Tier 1 }) as s_k, and denote the delay increase due to an inserted VI as d_{VI}. Based on the above, we classify timing paths of a design into three categories:

1) Type I: Timing non-critical paths ($s_k \geq s_{th}$);
2) Type II: Timing-critical paths without tolerance of VI insertion ($s_k < s_{th}$ && $\frac{D_k^{SS}-D_k^{FF}}{2} \leq d_{VI} + s_{gb}$);
3) Type III: Timing-critical paths with tolerance of VI insertions ($s_k < s_{th}$ && $\frac{D_k^{SS}-D_k^{FF}}{2} > d_{VI} + s_{gb}$);

Here, s_{th} is the threshold of timing slack to define the timing-critical paths (i.e., s_{th} = 10% of clock period); and s_{gb} is the slack guardband to evaluate tradeoff between delay impact of VI insertions versus timing improvement from mix-and-match stacking.[8] We note that when the delay of a VI insertion is so large that most of the timing-critical paths are Type-II paths, the timing benefits from mix-and-match die stacking will be limited.

Our optimization focuses on timing-critical paths (i.e., Type-II and Type-III paths). Our optimization ensures that startpoint and endpoint of a Type-II path are assigned to the same tier. Further, our optimization maximizes the number of Type-III paths being cut, so as to improve the potential timing benefits from mix-and-match die stacking. The procedure of our optimization is described in Algorithm 1. To construct the sequential graph, each startpoint or endpoint (e.g., register, PI or PO) becomes one vertex, and a directed edge is inserted between two vertices if there exists a (combinational) timing path between the vertices when they are taken as startpoint and endpoint. Note that in this optimization we only consider the maximum-delay path between any startpoint-endpoint pair. We use the algorithm in [8] for our maximum-cut optimization, in which the maximum-cut problem is relaxed to a semidefinite program (SDP). The SDP solution is then randomly rounded to achieve a partitioning solution. We use SDPA [27] as our semidefinite programming solver.

Algorithm 1 Partitioning of the sequential graph.

```
1: Extract restricted sequential graph $G_0$ that contains only Type-II and Type-III paths.
2: Collapse vertices connected with Type-II paths (edges) into one vertex to obtain a new graph $G_1$.
3: Perform maximum cut on $G_1$.
```

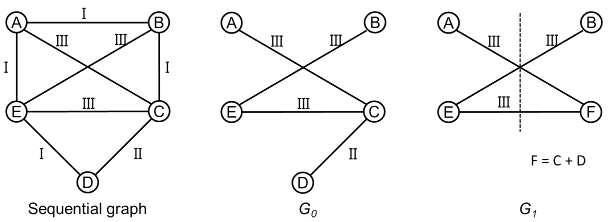


Fig. 4: Example of maximum-cut partitioning of the sequential graph. Types of paths are shown in edge labels. The dotted line indicates the final maximum-cut solution. We assume the same weight for all edges.

[5]Our separate study shows that delay impact caused by cells more than one stage upstream of the current cell is negligible (i.e., < 2ps). We therefore only consider the slew change due to current cell's direct fanins.

[6]We note that since the partitioning optimization is performed before placement and routing, the wire delay and accurate wire load information are not available, which might lead to suboptimality in the partitioning solution.

[7]Slight suboptimality of the ILP comes from the estimations of stage delay and delay impact of VI insertions, which are inputs to the ILP. The runtime to extract timing path information and solve the ILP can be even larger if there are more process bins, which makes the ILP-based methodology infeasible. The runtime of the ILP on AES (with 11K instances and 254K timing paths) is > 24 hours.

[8]The value of s_{th} needs to be empirically determined such that timing-critical paths are optimized. However, a too-large value of s_{th} can result in a large number of VI insertions and large runtime for timing analysis. Slack guardband s_{gb} is a flat timing margin, where the timing improvement from mix-and-match must exceed the VI delay impact by more than s_{gb}.

Figure 4 illustrates Algorithm 1 with an example consisting of five vertices and eight edges. The figure shows each updated graph, and the dotted line indicates the final maximum-cut solution.

B. Timing-Aware Multi-Phase FM Partitioning

Based on the maximum-cut partitioning solution of a timing-critical sequential graph, we fix the tier assignments of flip-flops and then perform *timing-aware multi-phase partitioning* to achieve the final partitioning solution. At each phase of our optimization, we perform optimizations in parallel with multiple threads. Optimization in each thread first clusters cells such that the size of the cluster is within a given range (i.e., $[N_{lb}, N_{ub}]$). Based on the clustered netlist, each thread then performs Fiduccia-Mattheyses (FM) optimization [6] to improve the partitioning solution in terms of the worst timing slack in the context of mix-and-match stacking. We vary the range of cluster sizes across different threads during our optimization. At the end of each phase, we select the partitioning solution with the maximum timing slack as the input to the next phase.

In our FM optimization, the gain function of a cluster u is defined as

$$gain(u) = \frac{\Delta slack(u)}{slack(u) - WNS} \tag{10}$$

where $slack(u)$ is the worst slack of cluster u; $\Delta slack(u)$ is the slack change when moving u across tiers; and WNS is the worst negative slack of the entire design.

Clustering cells at each phase before the FM optimization not only reduces the runtime of FM optimization but more importantly also improves the solution quality. Figure 5 shows an example in which moving one cell with negative gain can eventually lead to slack improvement after moving its neighbor cells. In the example, although moving one cell across tiers degrades the slack of the path due to VI insertions, moving its neighbor cells compensates for the delay impact of VI insertions and eventually improves the path timing for mix-and-match stacking. However, during the FM optimization, it is difficult and expensive (in terms of runtime) to "foresee" such slack benefits. In other words, to evaluate the gain function of one cell including its future impact, one must consider a large number of potential moves of its neighbor cells. The number of potential future move sequences can be large if only moving multiple stages of cells can compensate for the delay impact of VI insertions.[9] We therefore cluster cells such that timing improvement from moving a cluster can compensate for the delay impact of VI insertions. Further, since the goal of clustering and partitioning is to balance cell delays across tiers along each timing path, the desired cluster size highly depends on number of stages along the paths, fanout number at each stage, and netlist topology. Given that the number of stages along the path is limited by timing constraints, along with the maximum fanout constraint, a too-large cluster size might not help to balance delays across tiers along a timing path. We empirically set the cluster size to be no larger than 120 in our experiments.

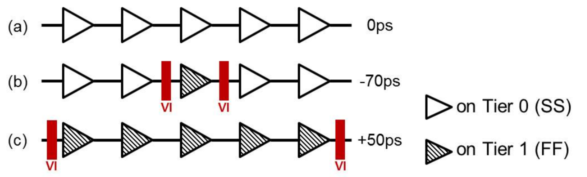


Fig. 5: Example to optimize a cell with a negative gain value. Assume that the difference between cell delays at SS and FF is 30ps, delay impact due to VI insertion is 50ps, and all cells along the path (only a segment of five cells is shown) are initially on Tier 0. Also assume that a stacking of SS Tier 0 + FF Tier 1 is applied. (a) Initial path with zero slack. (b) Moving one cell to Tier 1 degrades the slack by 70ps due to VI insertions. (c) Further optimization on the shown segment improves the slack by 50ps.

Algorithm 2 shows our clustering procedure. We first sort all cells in increasing order of their slacks (Line 1). We use topological order to break ties. We then select an unclustered cell from the ordered list as the starting point for clustering (Line 2). Based on the selected cell, we evaluate its slack changes due to moves (i.e., tier re-assignment) on its neighbor cells. If slack improves, we add the corresponding neighbor cell into the cluster (i.e., u), and further consider moves on neighbor cells of the new added cell (Lines 7-11, 15). However, when no move with slack improvement is available, we select the neighbor cell corresponding to the move with the minimum slack degradation and add it to the cluster (Lines 17-22, 27-30). The clustering procedure terminates when the cluster size meets the required range (i.e., $[N_{lb}, N_{ub}]$) or there is no unclustered neighbor cell (Lines 12-14, 24-26).

Note that each cluster contains cells originally belonging to the same tier. The slack of a cluster (i.e., $slack(u)$) is defined as the worst slack of cells within the cluster. Further, the estimation of $slack(\{c, u\})$ comprehends mix-and-match stacking (i.e., worst case over SS Tier 0 + FF Tier 1 and FF Tier 0 + SS Tier 1). Moreover, our timing analysis takes into account the delay impact of VI insertions (Figure 6 shows one example). Assuming that the incremental timing analysis is performed in constant time,[10] the runtime complexity of our clustering algorithm is $O(|C|^3)$.

Algorithm 2 Clustering.

```
1:  $cell\_list \leftarrow$ sort all cells in increasing order of their slacks
2:  for all $c \in cell\_list$ that is not clustered do
3:      $queue.push\_front(c)$; $u \leftarrow \emptyset$ // initialize cluster $u$
4:      while $|u| < N_{ub}$ do
5:          $s' \leftarrow -\infty$; $c' \leftarrow \emptyset$; $queue' \leftarrow \emptyset$
6:          while $|queue| > 0$ do
7:              $c \leftarrow queue.pop\_front()$
8:              $s_u \leftarrow slack(u)$; $s_c \leftarrow slack(c)$
9:              move $c$ to a different tier; incremental timing analysis
10:             if $|u| == 0$ || $slack(u) \geq s_u$ && $slack(c) \geq s_c$ then
11:                 $u \leftarrow u \cup \{c\}$
12:                 if $|u| \geq N_{ub}$ then
13:                     break
14:                 end if
15:                 $queue.push\_back($neighbors of $c$ that are not clustered$)$
16:             else
17:                 if $min(slack(c), slack(u)) > s'$ then
18:                     $s' \leftarrow min(slack(c), slack(u))$; $c' \leftarrow c$
19:                 end if
20:                 $queue'.push\_back(c)$
21:                 recover $c$ to its original tier; incremental timing analysis
22:             end if
23:         end while
24:         if $|u| \geq N_{ub}$ || $|queue'| == 0$ then
25:             break
26:         end if
27:         move $c'$ to a different tier; incremental timing analysis
28:         $u \leftarrow u \cup \{c'\}$
29:         $queue.push\_back($neighbors of $c'$ that is not clustered$)$
30:         $queue.push\_back(queue')$; $queue' \leftarrow \emptyset$
31:     end while
32: end for
```

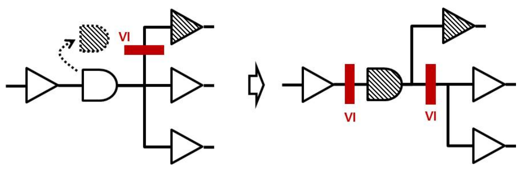


Fig. 6: Example of VI insertion/removal due to cell movement across tiers. Shaded cells are on Tier 1 and the others are on Tier 0.

In each run of FM optimization, we iteratively select the cluster with the maximum gain value and move it across tiers. We lock the clusters (cells) that have been moved. After each move, we perform incremental timing analysis and update the gain values of the neighboring clusters of which the worst slack is changed. We empirically observe that the slack improvement at the later stages of an FM run is small (e.g., shown Figure 7). Therefore, we terminate each FM iteration when 25% of clusters have been moved. Given that the initial partitioning solution is not area-balanced, in the first

[9] We are aware of "lookahead" approaches, such as gain vectors, CLIP/CDIP and LIFO gain buckets, etc. [4] [9] [17]. However, these are cut-centric and not path-aware, hence inapplicable to our current problem.

[10] In incremental timing analysis, we propagate slew and update cell delay through interpolation in Liberty lookup tables. Starting from the moved cell, we traverse the timing graph both forwards and backwards until there is no slack change. Given the maximum fanout constraints (e.g., 20) and limited number of stages to which "ripple effects" propagate (e.g., ~2-3 stages at most), in practice there is a constant bound on the number of cells updated during the incremental timing analysis.

FM iteration we terminate the optimization when the area balancing criterion is met. Figure 7 shows an example of our FM optimization on design AES. The optimization has three phases, where each phase contains two runs of FM optimization. We observe that the worst slack improves from -200ps to -14ps in this example with ∼3000 moves.

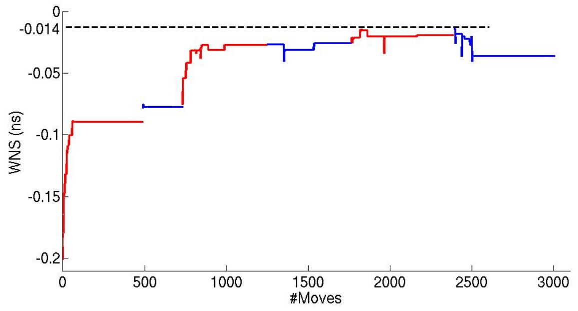


Fig. 7: An example of our multi-phase FM optimization. Design: AES. Technology: 28FDSOI. Cluster size ranges are [60, 70], [30, 40] and [15, 20]. Each phase contains two runs of FM optimization shown as red and blue curves. WNS improves from -200ps to -14ps. Runtime = 565 seconds on a 2.5GHz Intel Xeon server.

VI. Experimental Setup and Results

A. Experimental Setup

Our partitioning methodologies for mix-and-match stacking are implemented in C++. We use *CPLEX v12.5* [22] as our ILP solver and *SDPA* [27] as our semidefinite programming solver. Our SP&R (synthesis, placement and routing) flow uses *Synopsys Design Compiler H-2013.03-SP3* [23], *Cadence Encounter Digital Implementation System XL 12.0* [28], *Synopsys PrimeTime H-2013.06-SP2* [26] for logic synthesis, P&R, and timing and power analyses, respectively. Similarly to [19], we stitch SPEF files of Tier 0 and Tier 1, with annotated VI parasitics for timing and power analyses.

We use six open-source designs (DMA, USB, AES, MPEG, JPEG, VGA) [25] and an ARM Cortex M0 in our experiments. These testcases are generated with foundry 28nm FDSOI 12-track, dual-VT libraries. We use a BEOL stack of six metal layers for routing.

TABLE II: Testcases used in the experiments.

Design	#Instances	Clock period (ns)
DMA	2K	0.6
USB	4K	0.8
ARM Cortex M0	9K	1.2
AES	11K	1.1
MPEG	13K	1.2
JPEG	36K	1.4
VGA	73K	1.0

We conduct three experiments to evaluate the performance of our partitioning methodologies. (i) We validate the solution quality of our heuristic partitioning optimization by comparing its solutions with those of the ILP-based method. Due to poor scalability of the ILP-based method, we perform experiments on two small testcases (DMA and USB). (ii) We assess the benefit from our heuristic partitioning method within a brute-force 3DIC implementation flow, which we refer to as GT2012 [13]. (iii) We further assess the benefit from our partitioning method within a state-of-the-art 3DIC implementation flow (Shrunk2D) [19]. In our experiments, we perform three-phase optimization; each phase contains two FM runs. The ranges we use for cluster sizes are [100, 120], [80, 90], [60, 70], [40, 50], [20, 30], [10, 20]. Thus, our optimization uses six threads.

B. 3DIC Implementation Flows

Based on the conventional 2D implementation (P&R) flow, we study the GT2012 3DIC implementation as shown in Algorithm 3.[11] We first partition the netlist into two tiers (Line 1). After the partitioning, we place cells on Tier 0, and determine the VI locations based on that placement (Lines 2-3). With the fixed VI locations, we perform placement optimization on Tier 0 and Tier 1 separately (Line 4). We then insert a VI as the clock port on Tier 1. The clock VI location on Tier 1 is close to the clock port location on Tier 0 to minimize the cross-tier clock skew. We perform clock tree synthesis (CTS) on Tier 0 and Tier 1 separately (Lines 6-7). Last, we perform routing and routing optimization on each tier (Line 9). Note that we perform 3D timing analysis and update timing constraints for each tier after placement and CTS.

Algorithm 3 GT2012 3DIC implementation flow.

```
1: Netlist partitioning (MLPart [24] or our partitioning method);
2: Initial placement on Tier 0;
3: VI insertion based on placement of Tier 0;
4: Placement optimization on Tier 0 and Tier 1;
5: Timing constraint update;
6: VI insertion for clock port on Tier 1;
7: Clock tree synthesis (CTS) on Tier 0 and Tier 1;
8: Timing constraints update;
9: Routing and routing optimization on Tier 0 and Tier 1;
```

We also use the 3DIC implementation flow in [19] to validate our partitioning method. The flow first performs 2D implementation with scaled (i.e., 0.7 x) cell sizes and floorplan. Based on the shrunk 2D implementation, it partitions cells into two tiers. It further modifies the technology files so that BEOL stacks of two tiers (each has six layers) are connected as one (12-layer) BEOL stack and performs routing on both tiers to determine VI locations. Last, it performs routing and routing optimization on each tier separately. In the flow, all the clock cells are forced to be on Tier 0. Following [19], we refer to this flow as the Shrunk2D flow.

To be aware of mix-and-match die stacking, we extend both flows to perform a multi-view optimization after the netlist is partitioned, such that the die stacking of { SS Tier 0 + FF Tier 1, FF Tier 0 + SS Tier 1 } is captured during the P&R optimization. In addition, we assume face-to-face (F2F) die stacking in both flows.[12]

C. Experimental Results

Calibration of Heuristic Partitioning. We calibrate our heuristic partitioning method by comparing its solutions to those of the ILP-based method. We perform experiments on designs DMA and USB. We vary the VI insertion delay impact from 10ps to 50ps. We also assume different combinations of process conditions (i.e., { 3σ SS + 3σ FF, 2σ SS + 3σ FF, 3σ SS + 2σ FF }). Comparison results in Figure 8 show that except for one outlier, the timing slack resulted from our heuristic method is always within 30ps difference compared to the solution of the ILP-based method, where the ILP-based solution is considered to be very close to the optimal solution. This confirms that our heuristic method is able to comprehend asymmetric distribution of process bins and VI delay impact. The outlier occurs with the setup of large VI delay impact, where the problem becomes more challenging.

Validation of Our Method on GT2012 Flow. Table III shows the timing quality, total cell area, power, gate count, wirelength, number of VIs and post-routing utilization of implementations using the GT2012 flow and the GT2012 flow with our heuristic partitioning method. Note that the reported timing and power are the worst cases between SS Tier 0 + FF Tier 1 and FF Tier 0 + SS Tier 1. We observe that our partitioning approach leads to up to 16% timing improvement (i.e., on designs AES and VGA) compared to the GT2012 flow, which uses conventional min-cut partitioning [1] [24], while achieving similar area and power. This is a significant improvement, considering that even 20% improvement in performance per new technology generation is now quite difficult to achieve. The larger wirelength is because of additional wires routed to the increased number of VIs.

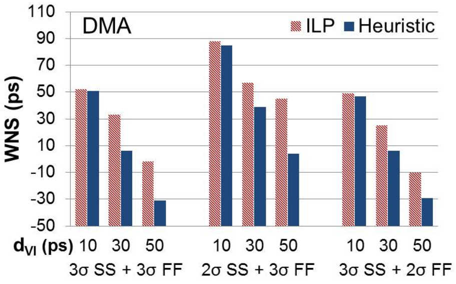
Validation of Our Method on Shrunk2D Flow. Table III shows design metrics of implementations using the original Shrunk2D flow [19] and its extension with our partitioning method. We observe that the extended flow with our partitioning approach achieves up to 7% timing improvement (i.e., on design ARM Cortex M0) with similar

[11]This 3DIC flow is similar to early flows that we have seen used, e.g., at U.S. Department of Energy laboratories.

[12]To maximize the timing benefit from mix-and-match die stacking, large number of VIs will be inserted. On the other hand, VI insertions will have area impact in a face-to-back stacking-based implementation. We therefore assume F2F stacking. We also note that F2F stacking and monolithic 3D integration are more preferable in the regime of mix-and-match die stacking due to their small VI area impact.

TABLE III: Validation of our partitioning methodology on GT2012 and Shrunk2D flows.

Design	Flow	WNS (ps)	TNS (ns)	Area (μm^2)	Power (mW)	#Instances	Wirelength (μm)	#VIs	Utilization (bottom / top)
ARM Cortex M0	GT2012 (orig)	**-178**	-56.735	8451	6.701	8816	116966	304	77% / 69%
	GT2012 (opt)	**-23**	-0.173	8448	6.210	8780	136631	2744	70% / 76%
	Shrunk2D (orig)	**-89**	-11.040	9697	6.499	9855	83462	3715	83% / 86%
	Shrunk2D (opt)	**-13**	-0.080	10106	6.985	9982	93495	4490	86% / 90%
AES	GT2012 (orig)	**-181**	-26.113	8536	10.700	10964	129896	250	74% / 70%
	GT2012 (opt)	**-8**	-0.012	8554	9.351	10947	156716	4417	65% / 79%
	Shrunk2D (orig)	**-4**	0.000	9621	10.600	11302	113209	4787	78% / 81%
	Shrunk2D (opt)	**56**	0.000	9611	10.200	11356	116816	6304	75% / 83%
MPEG	GT2012 (orig)	**-68**	-2.043	18089	13.900	13152	227734	307	69% / 73%
	GT2012 (opt)	**73**	0.000	18125	14.100	13185	321866	4674	74% / 67%
	Shrunk2D (orig)	**20**	0.000	18620	14.800	13275	158386	4741	72% / 74%
	Shrunk2D (opt)	**79**	0.000	18691	15.400	13279	174804	7727	77% / 70%
JPEG	GT2012 (orig)	**-155**	-7.094	44758	32.100	36521	703770	1159	69% / 72%
	GT2012 (opt)	**-52**	-0.462	45094	31.800	36631	1007156	12571	76% / 67%
	Shrunk2D (orig)	**-115**	-1.760	54457	42.900	52824	520123	14075	85% / 88%
	Shrunk2D (opt)	**-82**	-1.210	54637	43.000	52947	562430	20635	88% / 85%
VGA	GT2012 (orig)	**-244**	-6.213	100143	113.300	72682	2201814	1546	76% / 70%
	GT2012 (opt)	**-80**	-0.251	102683	117.200	72731	3667133	15353	70% / 80%
	Shrunk2D (orig)	**-47**	-0.270	104525	90.000	73950	904742	27780	76% / 77%
	Shrunk2D (opt)	**11**	0.000	104008	86.800	74051	929942	35908	79% / 73%



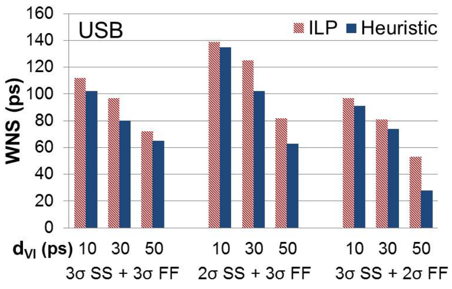


Fig. 8: Comparison of the solution qualities between the ILP-based method (which is near-optimal) and the heuristic method.

area, power and wirelength. Note that to maintain the solution of the 2D implementation in the scaled floorplan, we include additional bin-based area balancing constraints such that we uniformly divide the core area into N x N bins and set area balancing criteria for each bin during the FM optimization. We use three bin sizes in our optimizations – 20μm x 20μm, 30μm x 30μm and 50μm x 50μm – and report the result with the maximum timing slack.

VII. Conclusion

In this work, we study design-stage optimization for mix-and-match die stacking. Our motivating insight is that *a priori* knowledge of mix-and-match 3DIC integration should influence multi-die partitioning optimization and signoff. We propose an ILP-based partitioning methodology and a heuristic partitioning methodology that performs maximum cut on the timing-critical sequential graph followed by an iterative multi-phase FM optimization. We validate our partitioning optimization on two 3DIC implementation flows, each of which we have extended to be aware of mix-and-match die stacking. Our optimization leads to up to 16% timing improvement, as compared to a flow with min-cut based partitioning solution, when measured by RC extraction and signoff timing at the post-routing stage. Our study also indicates that a gate-level 3D integration has more flexibility and thus larger timing benefits in the mix-and-match regime as compared to a block-level integration. Our ongoing works include (i) integration of design-stage optimization and die- and/or wafer-level optimization for mix-and-match die stacking; (ii) clock tree synthesis for mix-and-match stacking; (iii) including BEOL variation in our optimization; (iv) a new abstraction model for slack improvement with mix-and-match stacking, for faster calculation of gain functions in FM optimization; and (v) more general formulations of die-level mix-and-match optimizations. We will also seek to develop more detailed cost modeling for multi-die integration – e.g., to understand how testability or other considerations might affect our study and/or its conclusions.

Acknowledgments

We thank authors of [19] for providing their 3DIC implementation flow scripts.

References

[1] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Improved Algorithms for Hypergraph Bipartitioning", *Proc. ASP-DAC*, 2000, pp. 661-666.
[2] T.-B. Chan, A. B. Kahng and J. Li, "Reliability-Constrained Die Stacking Order in 3DICs under Manufacturing Variability", *Proc. ISQED*, 2013, pp. 16-23.
[3] J. Cong, G. Luo, J. Wei and Y. Zhang, "Thermal-Aware 3D IC Placement Via Transformation", *Proc. ASP-DAC*, 2007, pp. 780-785.
[4] S. Dutt and W. Deng, "VLSI Circuit Partitioning by Cluster-removal Using Iterative Improvement Techniques", *Proc. ICCAD*, 1996, pp. 194-200.
[5] C. Ferri, S. Reda and R. I. Bahar, "Parametric Yield Management for 3D ICs: Models and Strategies for Improvement", *ACM JETCS* 4(4) (2008), pp. 19:1-19:22.
[6] C. M. Fiduccia and R. M. Mattheyses, "Linear Time Heuristic for Improving Network Partitions", *Proc. DAC*, 1982, pp. 175-181.
[7] S. Garg and D. Marculescu, "Mitigating the Impact of Process Variation on the Performance of 3-D Integrated Circuits", *IEEE TVLSI* 21(10) (2013), pp. 1903-1914.
[8] M. X. Goemans and D. P. Williamson, "Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming", *J. ACM* 42(6) (1995), pp. 1115-1145.
[9] L. W. Hagen, D. J.-H Huang and A. B. Kahng, "On Implementation Choices for Iterative Improvement Partitioning Algorithms", *IEEE TCAD* 16(10) (1997), pp. 1199-1205.
[10] Y. C. Hu, Y. L. Chung and M. C. Chi, "A Multilevel Multilayer Partitioning Algorithm for Three Dimensional Integrated Circuits", *Proc. ISQED*, 2010, pp. 483-487.
[11] I. H.-R. Jiang, "Generic Integer Linear Programming Formulation for 3D IC Partitioning", *Proc. IEEE ISOCC*, 2009. pp. 321-324.
[12] D.-C. Juan, S. Garg and D. Marculescu, "Statistical Peak Temperature Prediction and Thermal Yield Improvement for 3D Chip Multiprocessors", *ACM TODAES* 19(4) (2014), pp. 39:1-39:23.
[13] M. Jung, *personal communication*, 2013.
[14] A. B. Kahng, "New Game, New Goal Posts: A Recent History of Timing Closure", *Proc. DAC*, 2015, pp. 1-6.
[15] A. B. Kahng and X. Xu, "Local Unidirectional Bias for Smooth Cutsize-Delay Tradeoff in Performance-Driven Bipartitioning", *Proc. ISPD*, 2003, pp. 81-86.
[16] G. Karypis and V. Kumar, "Multilevel K-Way Hypergraph Partitioning", *Proc. DAC*, 1999, pp. 343-348.
[17] B. Krishnamurthy, "An Improved Min-Cut Algorithm for Partitioning VLSI Networks", *IEEE Trans. on Computers* 33(5) (1984), pp. 438-446.
[18] Z. Li, X. Hong, Q. Zhou, Y. Cai, J. Bian, H. H. Yang, V. Pitchumani, C.-K. Cheng, "Hierarchical 3-D Floorplanning Algorithm for Wirelength Optimization", *IEEE Trans Circuits Syst I* 53(12) (2006), pp. 2637-2646.
[19] S. Panth, K. Samadi, Y. Du and S. K. Lim, "Design and CAD Methodologies for Low Power Gate-level Monolithic 3D ICs", *Proc. ISLPED*, 2014, pp. 171-176.
[20] S. Panth, K. Samadi, Y. Du and S. K. Lim, "Placement-Driven Partitioning for Congestion Mitigation in Monolithic 3D IC Designs", *IEEE TCAD* 34(4) (2015), pp. 540-553.
[21] T. Thorolfsson, G. Luo, J. Cong and P. D. Franzon, "Logic-on-logic 3D Integration and Placement", *Proc. 3D Systems Integration Conference*, 2010, pp. 1-4.
[22] IBM ILOG CPLEX. *www.ilog.com/products/cplex/*
[23] Synopsys Design Compiler User Guide. *http://www.synopsys.com*
[24] MLPart. *http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/Partitioning/MLPart*
[25] OpenCores: Open Source IP-Cores. *http://www.opencores.org*
[26] Synopsys PrimeTime User Guide. *http://www.synopsys.com*
[27] SDPA Official Page. *http://sdpa.sourceforge.net/*
[28] Cadence SOC Encounter User Guide. *http://www.cadence.com*