Fast Unified Floorplan Topology Generation and Sizing on Heterogeneous FPGAs

Pritha Banerjee, Susmita Sur-Kolay, *SMIEEE* , and Arijit Bishnu

Abstract

Recent FPGA architectures are heterogeneous owing to the presence of millions of gates in Configurable Logic Blocks (CLBs), Block RAMs, and Multiplier blocks (MULs) which can host fairly large designs. While their physical design calls for floorplanning in contrast to normal FPGAs, the traditional algorithms for ASICs do not suffice. In this paper, we propose a three phase algorithm (HeteroFloorplan) for unified floorplan topology generation and sizing on heterogeneous FPGAs. The method consists of a recursive balanced bipartitioning followed by generation of slicing topologies, and a final allocation of CLBs, RAMs and MULs to modules by a greedy heuristic (GARR) and a min-cost max-flow formulation. Experimental results on benchmark circuits show that our floorplan generation method can produce feasible solutions within a few seconds. We observed an improvement in total half-perimeter wirelength between $\mathbf{18\%}$ to $\mathbf{52\%}$, compared to the very few previous approaches. We also compare our greedy allocation with a network flow based method to establish the effectiveness of our locally greedy heuristic.

Index Terms

Heterogeneous FPGA, floorplanning, slicing topology, sizing.

I. Introduction

The spectrum of FPGA based systems, especially embedded ones, has become very wide. Modern FPGA architectures have been aggressively taking over from ASICs in certain areas. These FPGA architectures are significantly different from those that were available in the last decade. Earlier, CLBs were a homogeneous resource and arranged in rows and columns uniformly, with Primary Input and Outputs (PI/POs) on the periphery of the chip. Recent FPGA architecture comprises not only the CLBs and PI/POs, but also Multipliers (MUL), Block RAMs, DSP and microprocessor cores. Few columns of RAM-MUL pairs and even I/O blocks are interspersed among CLB columns. Moreover, a large design with millions of gates is partitioned into a smaller number of functional modules to reduce the place-and-route time and to achieve better quality of solution. This necessitates a floorplanning step for hierarchical designs in the physical design flow of FPGAs. Though a large volume of work exists for ASIC floorplanning, it is generally not employed while mapping designs onto the earlier sea-of-gates style FPGAs.

P. Banerjee, S. Sur-Kolay and Arijit Bishnu are with the Advanced Computing and Microelectronics Unit, Indian Statistical Institute, Kolkata, 700108, India. e-mail:{pritha_r,ssk, arijit}@isical.ac.in

In a typical FPGA physical design flow, after technology-mapping, a flattened CLB netlist is directly placed [1], [2], [3], [4] and routed without any floorplanning. Of course, for hierarchical designs, modules or macros consisting of CLBs only were floorplanned/placed using various bin packing techniques [5]. But, for modules with heterogeneous resource requirements, neither this technique nor the traditional floorplanners for ASICs [23], [30], [32] adapted to FPGAs, are adequate [6]. Hence there is a pressing need for fast floorplanning techniques that consider the heterogeneous logic and routing resources of modern FPGAs.

The literature on FPGA floorplanning (both homogeneous and heterogeneous) is merely a handful. Emmert et al. [7] devised a macro based floorplanning methodology for earlier generation sea-of-gates style FPGAs. Their method uses clustering techniques to combine macros into clusters, and then places the clusters with enhanced circuit routability and performance using Tabu search. Cheng and Wong [8] proposed the first floorplanning algorithm targeted for heterogeneous FPGAs. In [9], Yuan et al. have proposed a packing-based method using Less Flexibility First (LFF) principle. Recently, Feng et. al. [10] and Singhal et. al. [12] have also proposed methods for floorplanning on FPGAs with heterogeneous resources.

The methods proposed in [8], [10], and [12] are based on *Simulated Annealing* (SA), whereas the one in [9] uses a deterministic heuristic. Our work *HeteroFloorplan*, presented in this paper, is a deterministic heuristic where we do not use simulated annealing or any other local search heuristic. Here, we propose a methodology for unified floorplan topology generation and sizing onto target FPGA architecture with preplaced heterogeneous resources. In this paper, the preliminary method proposed in [13] is further supported by various experimental results, theoretical analysis and verification of our method *HeteroFloorplan*. The experimental results indicate that *HeteroFloorplan* is fast and can produce floorplans with improved half-perimeter wirelength when compared to existing methods.

The layout of the paper is as follows. In Section II, we briefly describe the FPGA architecture, define the floorplanning problem for heterogeneous FPGAs, the objective function to be optimized and then review the prior works in floorplanning of heterogeneous FPGAs. Section III has the details of our proposed method *HeteroFloorplan* and its time complexity. This method is illustrated with an example in Section IV. Experimental results are reported in Section V. Section VI validates the greedy allocation of rectangular regions (GARR) of Phase III by comparing it with a network flow formulation for the floorplanning problem. Concluding remarks appear in Section VII.

II. Background

A. Architecture

In modern FPGAs, CLBs and routing resources are arranged in rows and columns as before, but there are also other types of resources placed in certain patterns, to satisfy a wider range of design requirements. Figure 1 shows a Xilinx Spartan-3 [14] FPGA where the CLBs are arranged in columns interleaved with columns of RAM-MUL pair at certain intervals. Each small square represents a CLB. A RAM block paired with a MUL block, and spanning a height of four rows of CLBs, is indicated by an empty and a shaded rectangle respectively. Henceforth, we assume this architecture for this paper, although the methodology is applicable to other similar ones.

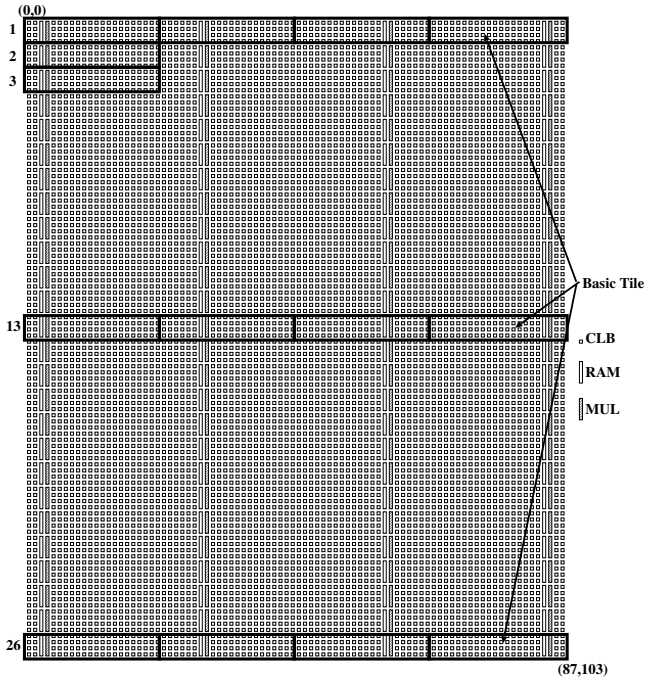


Fig. 1. Spartan-3 XC3S5000 FPGA Architecture, tesselated with a basic tile, indicated by a rectangle of 4 rows and 20 columns of CLBs and 1 pair of RAM-MUL blocks

B. FPGA Floorplanning Problem

First, the basic terminology is given below.

Modules and Signal nets: Let $M = \{m_1, m_2, \ldots, m_n\}$ be a set of n distinct modules. Let $S = \{s_1, s_2, \ldots, s_q\}$ be a set of q signal nets. Each signal net $s_i \in S$ is associated with a set of distinct modules $M_{s_i} = \{m_j \mid m_j \in M\}$, and the set S is called a *netlist*. If $M_{s_i} = M_{s_j}$, then the two distinct signal nets s_i and s_j connect the same set of modules.

Resource Requirement Vector [8]: For a module m, a 3-tuple vector $\rho = (m_{clb}, m_{ram}, m_{mul})$ represents the number of CLBs, RAMs and MULs required by module m.

Target Architecture: Let W and H be the width and height of a target FPGA architecture, where the units are the width and height of a CLB respectively. A coordinate system $(0, 0, W, H)$ with top-left corner at $(0, 0)$ and bottom-right corner at (W, H) is assumed for the given chip. In Figure 1, it is $(0, 0, 87, 103)$ Each resource on the architecture is identified by its coordinate position (x, y), where $0 \leq x \leq W$ and $0 \leq y \leq H$. Henceforth, the term target FPGA architecture and target chip will be used synonymously.

FPGA Floorplanning Problem : Given (1) a *target architecture* $(0, 0, W, H)$ with its resource locations, (2) a design $\mathcal{D}$ consisting of (a) a set of soft (flexible in shape) modules M, (b) the resource requirement vectors ρ_{m_i} for each $m_i \in M$, and (c) the netlist S,

find a floorplan by assigning a connected region $R_i = (x_i^{min}, y_i^{min}, x_i^{max}, y_i^{max})$ to each module m_i on the target

architecture such that

(i) $0 \leq x_i^{min} \leq x_i^{max} \leq W$ and $0 \leq y_i^{min} \leq y_i^{max} \leq H$,

(ii) region for no two modules overlap with each other,

(iii) for each module m_i, the resources in its region satisfies ρ_{m_i}

(iv) a certain cost function is optimized.

A floorplan is said to be *feasible* if it satisfies all three conditions (i), (ii) and (iii). The cost function to be optimized is typically the wirelength [15], [32] for which the popular metric HPWL (half-perimeter wirelength), i.e., the sum of the semi-perimeter of the bounding boxes for each net, is used. In the absence of information at this stage, the net terminals on a soft module are assumed to be at the center of the module. This bounding box cost has also been extensively used as a FPGA placement metric [1]. The problem formulation as stated above is a generalization of that given in [10], [15], [19] and as such is *NP*-hard. Like most of the prior works on FPGA heterogeneous floorplanning [10], [11], we also consider HPWL as the objective function.

C. Previous Works

Cheng and Wong [8], [11] proposed the first floorplanning algorithm targeted for heterogeneous FPGAs that can produce feasible solution employing simulated annealing to optimize area, half-perimeter wirelength (HPWL) and the aspect ratios of modules. Their method begins with a slicing floorplan topology [17] and use simulated annealing to iteratively perturb and improve it. Given an irreducible realization list (IRL), i.e., realizations with distinct (width, height) pair for each node of a slicing tree, they devise efficient means to compute IRLs as nodes are merged from leaf level up to root, thereby finding optimal realizations of a particular slicing tree. The authors mention that their method is a non-trivial extension of Stockmeyer floorplan optimization algorithm [16].

Yuan et al. [9] have proposed a packing-based method using Less Flexibility First (LFF) principle. The authors first generate all realizations of a module based on the current packing configuration. Then, based on the value of a flexibility function defined by them, the realizations are sorted. Thereafter, all realizations of different modules are collected into a list. The realization with the highest fitness value is selected. As to the worst case time complexity, the authors point out that their method takes $O(W^2 n^5 \log n)$ where n is the number of modules and W is the width of the chip. The authors claim that the $\log n$ factor in the time complexity comes from range searching with the help of a *kd-tree*. But, *kd-tree* takes $O(\sqrt{n})$ ($O(n^{1-\frac{1}{d}})$, where d is the dimension) for range searching in 2D [18]. Time complexity proportional to $\log n$ is achievable if fractional cascading [18] is used, but then space complexity would go upto $O(n \log n)$. The method focuses on packing the modules onto the target chip rather than optimizing for wirelength.

Recently Feng and Mehta [10] presented a two step approach based on resource aware fixed outline simulated annealing which optimizes the HPWL, starting from a given topology, followed by max-flow based constrained floorplanning. As FPGA is a bounded rectangle, the authors in [10] propose a fixed outline simulated annealing algorithm in contrast to the area minimization approach of [8]. The authors use a penalty term in their simulated annealing cost function so that modules are placed as close as possible to their resources. The shortcoming of each

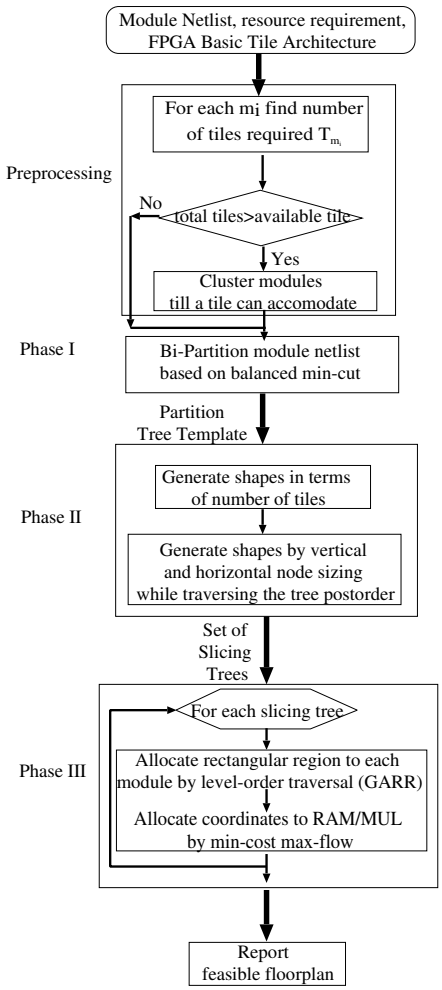


Fig. 2. Flow of our Floorplanning Method

module in meeting its resource requirement is then taken care of by a constrained floorplanning which is based on a min-cost-max-flow network formulation. This constrained floorplanning is a generalization of the method in [19] that was developed for ASICs. In [12], Singhal et. al. have proposed another SA based floorplanner with an adaptive placer algorithm. This multilayer floorplanning algorithm reports better HPWL for floorplans with statistically large variations in the heterogeneous resources. However, the runtime is $1.4\times$ than the traditional non-heterogeneous floorplanner.

III. Proposed Floorplanning Method

In our work, we begin from a point where neither the slicing topology nor the shape of the modules are given; only the netlist and the resource requirements after technology mapping are known. Our floorplanning methodology consists of three phases, namely, construction of a recursive bi-partition tree as a template for possible slicing topologies, generation of floorplan topology with sizing [23], [24] and feasible realization of the topology as shown

in Figure 2.

The target FPGA architecture is represented as a $2D$ array of rectangular *basic tiles* (defined later). If the number of tiles required by all the modules is more than the number of tiles available on the board due to over estimation of resource requirement, we employ a clustering step to pack multiple modules in tiles. Then, in the first phase, we recursively bi-partition the netlist in a balanced manner to obtain a binary partition tree with modules and clusters at the leaves.

In the second phase, for each module, we generate a set of possible rectangular shapes in terms of tiles satisfying its resource requirements. A list of candidate floorplan slicing topologies, called slicing trees [23] represented by binary trees, is constructed. Appropriate sizing of the nodes of the bi-partition tree in postorder [16] is computed in polynomial time.

In the third phase, for each slicing tree obtained in the previous step, a rectangular region within the target boundary $(0, 0, W, H)$ is assigned to each module which respects the cut direction and the resource requirements. Finally, among all the floorplans obtained, the realization with acceptable aspect ratios of modules and best HPWL is reported.

A. Basic Tile of a FPGA architecture

To explain this step, a few definitions are in order.

Definition 1: A *basic tile* of a given FPGA architecture is an indivisible unit having a fixed rectangular shape and containing a minimum number of each type of resource, such that the architecture can be represented as a 2D array of these units.

Although the number of each type of resource in any basic tile remains constant, the relative positions of these resources may vary within the rectangular boundary of the tile depending on its indices in the 2D array of basic tiles. Let the given architecture be thus composed of $(W_t * H_t)$ *basic tiles*, arranged in H_t rows and W_t columns. In the first two phases we represent the FPGA by (W_t, H_t). For the typical architectures with CLBs, RAMs and MULs, let us denote a basic tile by a 3-tuple vector A = (a_{clb}, a_{ram}, a_{mul}). In Figure 1, the basic tile A = (80, 1, 1) consists of $20 * 4$ CLBs placed in 20 columns and 4 rows, and a pair of 1 RAM and 1 MUL, placed in two adjacent columns. A *b*asic tile A is indicated by a thick-lined rectangle in the figure. The entire architecture (Spartan-3) shown in Figure 1 can be covered by 26 rows and 4 columns of the basic tile A, i.e., (W_t, H_t) = (4, 26).

Assuming the RAMs and MULs to occur in pairs, let there be r columns of such pairs of RAM/MUL in a given FPGA. Let x_i $(i = 2 \ldots r)$ be the number of CLB columns between the $(i-1)^{th}$ and i^{th} column of RAM/MUL. There may be x_1 and x_{r+1} number of CLB columns respectively to the left of the first pair and to the right of the r^{th} pair of RAM/MUL columns. Then, the width W of the FPGA can be expressed as

$$W = 2r + \sum_{i=1}^{r+1} x_i \tag{1}$$

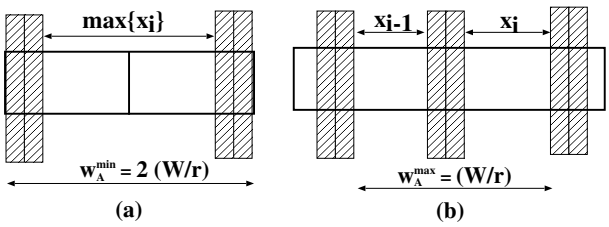


Fig. 3. Range of basic tile width w_A: (a) lower bound, (b) upper bound

Therefore, the width of a tile is w_A. The maximum number of CLB columns between two consecutive pairs of RAM/MUL columns on the target chip is $\max_{i=1}^{r+1}\{x_i\}$. This situation can occur if for two horizontally adjacent tiles, the RAM/MUL columns for the left tile is located at its leftmost and that for the right tile is located at its rightmost, as shown in Figure 3(a). So, we have

$$\begin{aligned} \max_{i=1}^{r+1}\{x_i\} &\leq 2(w_A) - 4 \\ or, \quad \frac{\max_{i=1}^{r+1}\{x_i\}}{2} + 2 &\leq w_A \end{aligned}$$

Since a basic tile has only one pair of RAM/MUL columns, the width of the basic tile w_A cannot exceed the number of columns between the $(i-1)^{th}$ and $(i+1)^{th}$ RAM/MUL pairs, for $i = 1, \ldots, r$. Refer Figure 3(b). This gives us

$$\min_{i=1}^{r+1}\{x_{i-1} + x_i + 2\} \geq w_A$$

Combining the above two inequalities, we have

$$\frac{\max_{i=1}^{r+1}\{x_i\}}{2} + 2 \leq w_A \leq \min_{i=1}^{r+1}\{x_{i-1} + x_i + 2\} \tag{2}$$

The above formulation can be generalized to the case when there are k RAM/MUL columns per tile as $w_A = \frac{W}{kr}$ and the expression is

$$\frac{\max_{i=1}^{r+1}\{x_i\}}{2} + 2k \leq w_A \leq \min_{i=1}^{r+1}\{\sum_{j=i-k}^{i} x_j + 2k\} \tag{3}$$

For any FPGA architecture satisfying the above, we can have a basic tile of width w_A. The Spartan-3 board shown in Figure 1 satisfies inequality 2. Thus, the width of the basic tile is $\frac{88}{4} = 22$. The basic tile of width 22 consists of 20 columns of CLBs, 1 RAM and 1 MUL.

If it is not possible to derive such a basic tile, a tile that covers uniformly at least the minority resources like RAM/MUL, and most of the majority resource like CLBs, can be computed. Thus, there may be a few fractional tiles, consisting of fewer CLBs than the remaining tiles on board. Our floorplanning method can still be used for architectures with such fractional tiles, which takes care of this situation in Phase III of *HeteroFloorplan*. In the target FPGA architecture used in this paper, fractional tiles however were not required.

Definition 2: For a given basic tile A, the *Tile Requirement* T_{m_i} of a module m_i with resource requirement vector ρ_{m_i}, is the minimum number of basic tiles it requires. This is given by

$$T_{m_i} = \lceil max(\frac{m_{clb}}{a_{clb}}, \frac{m_{ram}}{a_{ram}}, \frac{m_{mul}}{a_{mul}}) \rceil \quad (4)$$

Let $T_{tot} = \sum_{i=1}^{n} T_{m_i}$ be the total number of tiles required by all the modules. If $T_{tot} \leq W_t * H_t$, then the modules can be floorplanned within the target chip in the subsequent phases of *HeteroFloorplan*. However, it may happen that although the total resource requirement in terms of CLB, RAM, MUL is less than the total resources available on the chip, in terms of tiles $T_{tot} > (W_t * H_t)$. This occurs when there is a large number of small modules with resource requirement much less than a basic tile. For such circuits, we employ a greedy heuristic pre-processing step of clustering to pack as many modules as possible into a basic tile, considering their connectivity. This facilitates the third phase of *HeteroFloorplan* to generate a floorplan with a connected region for each module satisfying the respective resource requirements.

Clustering step for large number of small modules: In a given design $\mathcal{D}$, the set of small modules $M_{small} \subseteq M$, each having size less than or equal to that of a basic tile and requirement of one or zero RAM and/or MUL, are clustered by modeling M_{small} along with their signal nets as a graph $G_c = (V_c, E_c)$. Each vertex $v_i \in V_c$ corresponds to a small module $m_i \in M_{small}$. Each net $s_i \in S$ of $\mathcal{D}$ is modeled as a clique of vertices that constitute the net. There exists an edge $e = (v_i, v_j) \in E_c$ if $v_i, v_j \in V_c$ is a subset of M_{s_i} for a net $s_i \in S$.

First, the elements of M_{small} are sorted in ascending order of their CLB requirement. In each iteration, the smallest module m_c is chosen as seed from this sorted list, and a new cluster is grown around it by packing as many adjacent modules as possible, from the graph G_c in increasing order of their CLB requirement, into a basic tile for the cluster with seed m_c. After deleting all the modules in this cluster from the sorted list, a new super-module for this cluster is inserted maintaining the sorted order. This process is repeated until no more packing is possible based on net connectivity. Then we pack the rest of the modules/clusters by best-fit bin packing strategy, minimizing the number of bins. Thus we arrive at a new reduced netlist of modules/clusters which has to be floorplanned on (W_t, H_t). The time complexity of the clustering method is $O(n_s^2)$, where $n_s \leq n$ is the number of small modules.

B. Phase I: Generation of partition tree

Min cut Partitioning of Module Netlist : In order to bring connected modules closer to each other, such that the wirelength is minimized in the feasible floorplan, the module netlist is bi-partitioned recursively based on min-cut. The partitions are also weight balanced across the cut edges according to the tile requirements of each module. Since the target chip is covered with uniform tiles, balancing the modules on the basis of tile requirement can evenly distribute them across cut lines. This increases the possibility of generating a compact floorplan as there will be less white spaces generated during merging of almost equal sized modules by vertical and horizontal cutlines in the later phases of the method. We employ a state-of-the-art hypergraph partitioning tool *hMetis* [20][21] to obtain the weight balanced min-cut partitioning of the module netlist. This yields the relative ordering of the circuit modules which is the template for generating a set of slicing topologies.

The input to the *hMetis* tool is a netlist, which is a hypergraph $H = (V, E)$. Each vertex $v \in V$ corresponds to a module m_i, i = $1, 2, \ldots, n$. A hyperedge $e = \{v_1, v_2, ...\} \in E$ corresponds to M_{s_i}. If $\{M_{s_i}, M_{s_j}, \ldots M_{s_l}\}$ are identical for the signal nets s_i, s_j, ..., s_l, then each of $s_i, s_j, \ldots s_l$ corresponds to the same hyperedge with the number of such signal nets as the weight of the hyperedge. The weight of a vertex $v \in V$ is T_{m_i} for module m_i corresponding to v. The hypergraph H thus generated, is bi-partitioned recursively to n parts, generating a binary partition tree $\mathcal{B}$ with its leaves corresponding to n modules. As established by Even et al. [22] in their work on divide-and-conquer approximation algorithms for optimal linear arrangement, the left to right ordering of the leaves of $\mathcal{B}$ produced by *hMetis* is a good estimate of the *linear arrangement* of the modules in the netlist hypergraph. This can very well help in reducing the final wirelength and hence, delay.

Moreover, balanced bi-partitioning of the weighted netlist hypergraph, where the weight of each module is its proportional area requirement, helps in minimizing the whitespaces generated during node sizing and topology generation in Phase II. By employing the min-cut based balanced partitioner the modules are distributed evenly on the FPGA board on the basis of RAM/MUL.

C. Phase II: Floorplan topology generation

Next, the relative position and shapes of each module/ cluster is computed by an unified topology generation and sizing method. In this step, a set of sliceable floorplan topologies (i.e., slicing trees) is generated by appropriate horizontal and vertical node sizing of a set of possible shapes (in terms of basic tiles) for each module. The work of Otten and Stockmeyer [17], [16] finds an area optimal floorplan for a given slicing topology and a set of soft modules with a list of shapes for each module. So, it assumes a slicing topology to be given and finds the shape of each module for an area optimal floorplan. In our work, we begin from a point where neither the slicing topology nor the shape of the modules are given; only the netlist, the resource requirements after technology mapping and the partition tree template $\mathcal{B}$ are known.

1) Generation of Module Shapes:

Definition 3: A list D = $\{(w_1, h_1), (w_2, h_2), \ldots, (w_t, h_t)\}$ of *irredundant shapes* of a module m, is a list of t possible shapes of m, where (w_i, h_i) denotes the width and height of the i^{th} shape of m in terms of basic tiles. D is said to be irredundant if each individual w_i and h_i are distinct [23].

By making individual w_i and h_i distinct as in Def. 3, a shape with smaller height is chosen from two implementations with the same width. Thus, an inferior shape is always excluded from D. A set of possible irredundant rectangular shapes for m_i is created with all possible pairs of integers whose product is T_{m_i}. As we are considering only rectangular shapes, there may not be many choices such that $width * height$ = T_{m_i}. If T_{m_i} is a prime number, only two shapes are generated. so we also add the shapes corresponding to the factors of $(T_{m_i} + 1)$ to obtain a few more shape pairs, i.e., (width, height) with better aspect ratio. Since size of each cluster is only a single tile, there is only one shape, i.e.,$(w, h) = (1, 1)$, is possible.

Let $\alpha \geq 1$ be a positive integer denoting the maximum aspect ratio defined for any module. A set of (w_i,h_i) pairs is generated for each module such that,

$$\frac{w_i}{h_i} \leq \alpha, \text{ if } w_i \geq h_i \quad \text{or} \quad \frac{h_i}{w_i} \leq \alpha, \text{ if } h_i \geq w_i$$

Thus, for each module m_j, $j = 1, 2, \ldots, n$, we generate a set of t_j possible irredundant shapes $D_j = \{(w_1, h_1), (w_2, h_2), \ldots, (w_i, h_i), \ldots (w_{t_j}, h_{t_j})\}$. The following lemma gives the time required for generating all different shapes.

Lemma 1: If $\kappa = max\{t_j\}$ is the maximum number of shapes generated for any module m_j, then it would require $O(n\kappa)$ time for finding the values of all the shapes for n modules.

Proof: Obvious. ■

Let ν denote the maximum number of basic tiles that a module may require. This implies that if each of $(n-1)$ modules have only one tile, then the remaining one module can have atmost $\nu = \lceil \frac{W \times H}{c} \rceil - (n-1)$ tiles, where c is the size of a basic tile A defined for a given floorplan problem. An upper bound on the value of κ is given by $O((\log \nu)^{\log \nu})$. The details appear later in the Appendix.

2) Node sizing: A set of slicing trees/floorplan topologies are next derived from the partition tree $\mathcal{B}$ of Phase I by appropriate node sizing. So, a subtree of $\mathcal{B}$ rooted at an internal node p corresponds to a sub-floorplan. Let the list of irredundant shapes for the left and right sub-floorplans of p be denoted by $D_l = \{ (w_{l_1}, h_{l_1}), (w_{l_2}, h_{l_2}), \ldots, (w_{l_s}, h_{l_s}) \}$, with $|D_l| = s$ and $D_r = \{ (w_{r_1}, h_{r_1}), (w_{r_2}, h_{r_2}), \ldots, (w_{r_t}, h_{r_t}) \}$, with $|D_r| = t$. A sub-floorplan at node p is generated by abutting the i^{th} member (w_i, h_i) of D_l with the j^{th} member (w_j, h_j) of D_r either vertically or horizontally. If the children of p are leaves, then the left and right sub-floorplans are the modules themselves.

Vertical Cut: We use the vertical node sizing algorithm of [16] to generate a sub-floorplan with vertical cut. The list D_l is sorted such that,

$$w_{l_1} < w_{l_2} < \ldots < w_{l_s} \quad \text{and} \quad h_{l_1} > h_{l_2} > \ldots > h_{l_s}$$

D_r is also sorted similarly. If (w_{l_i}, h_{l_i}) and (w_{r_j}, h_{r_j}) are abutted vertically, the resultant floorplan size becomes $(w_{v_k}, h_{v_k}) = (w_{l_i} + w_{r_j}, max(h_{l_i}, h_{r_j}))$. Thus, the list V^p of resultant irredundant shapes for the sub-floorplan at p is generated and has at most $s + t - 1$ members [23].

Horizontal Cut: If the two subfloorplans of p are abutted using horizontal cut, we use the same irredundant lists D_l and D_r described above. But the lists are now sorted in increasing order of height and decreasing order of width, i.e.,

$$h_{l_1} < h_{l_2} < \ldots < h_{l_s} \quad \text{and} \quad w_{l_1} > w_{l_2} > \ldots > w_{l_s}$$

By abutting $(w_{l_i}, h_{l_i}) \in D_l$ and $(w_{r_j}, h_{r_j}) \in D_r$ horizontally, the resultant size of the floorplan becomes $(w_{h_k}, h_{h_k}) = (max(w_{l_i}, w_{r_j}), h_{l_i} + h_{r_j})$. As in the case of a vertical cut, the cardinality of the list H^p of resultant irredundant shapes for the sub-floorplan at p is at most $s + t - 1$.

3) Generation of Slicing Trees: For each internal node p of the partition tree $\mathcal{B}$, two lists V^p and H^p are constructed as above from the child sub-floorplans. A combined list M^p of irredundant shapes is computed by merging V^p and H^p such that,

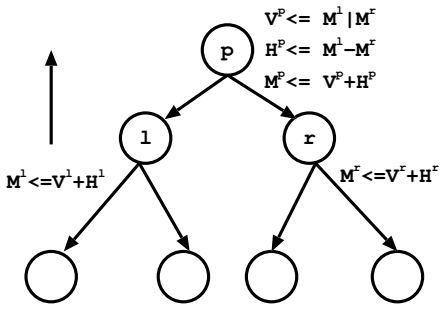


Fig. 4. Lists of shapes created at any internal node of partition tree B.

$$w_1 < w_2 < \ldots < w_k \quad \text{and} \quad h_1 > h_2 > \ldots > h_k$$

is satisfied. Here, k is the total number of irredundant shapes generated at node p. If the same shape (w, h) is generated in V^p and H^p, typically we choose the horizontal cut, as it is easier to obtain contiguity of the RAM/MULs within the region allocated to a module during Phase III of *HeteroFloorplan*. While the algorithm for node sizing [23], [16] to generate irredundant shapes at a node assumes a given slicing topology, *HeteroFloorplan* generates a set of slicing topologies as we traverse the partiton tree in post-order by considering cuts in both directions during node sizing at every node.

Lemma 2: If s and t are the respective cardinalities of the lists D_l and D_r of the left and right subfloorplans of a node p, then the number of shapes in M^p at node p is no more than $2(s + t - 1)$.

Proof: As we are merging the vertical and horizontal lists according to the condition mentioned above, the size can grow at most by a factor of 2 compared to that in [23]. ■

The combined lists M^l and M^r created at the left and right children l and r of the node p is used for sub-floorplan generation at its parent node p as shown in Figure 4. The symbols '$|$' and '$-$' in the figure represent the vertical and horizontal node sizing, whereas the symbol '$+$' denotes merging of the irredundant list of shapes obtained at left and right operand. The arrow in the figure shows the bottom-up postorder processing of nodes. Thus, the nodes of the tree $\mathcal{B}$ are processed in *post-order* to generate a set of subfloorplans at every internal node p. We store a subfloorplan at p as a 5-tuple $(w_i, h_i, cut_i, l_\sigma, r_\tau)$, where (w_i, h_i) is the i^{th} shape of node p which is generated by merging $(l_\sigma)^{th}$ shape of left child l and $(r_\tau)^{th}$ shape of right child r using cut_i. The value of cut_i is either vertical or horizontal.

Theorem 1: By horizontal or vertical node sizing atmost $O(\kappa n^2)$ shapes (slicing trees) can be generated at the root of the tree, where n is the number of modules and κ the maximum number of shapes for a module.

Proof: At any node at level i ($i = 1, \ldots, \log_2 n + 1$) (leaf is at level 1) of the slicing tree, the size of the list is $4^{i-1}(\kappa - \frac{2}{3}) + \frac{2}{3}$. Using Lemma 2, one can prove this result by induction. The number of nodes at a level i is $\frac{n}{2^{i-1}}$. So, the number of shapes at any level i is bounded by $\frac{n}{2^{i-1}}(4^{i-1}(\kappa - \frac{2}{3}) + \frac{2}{3})$. Now, as $i = O(\log_2 n)$ at the root, the number of shapes/slicing trees generated at the root is $O(\kappa n^2)$ $(n\kappa \sum_{i=1}^{\log_2 n} 2^{i-1})$. ■

During the postorder processing of nodes, we also calculate the resource requirement ρ_p = (p_{clb}, p_{ram}, p_{mul}) at every node p by summing up the resources ρ_l and ρ_r required by its left and right child respectively. The requirement vector is used for realization of the slicing tree in Phase III. Thus, at the root we get a set of floorplan shapes $\mathcal{F}$ = $\{(T_{w_i},T_{h_i})\}$ where T_{w_i} and T_{h_i} are respectively the width and the height of the i^{th} slicing tree/floorplan in terms of tiles. Each shape of $\mathcal{F}$ corresponds to a distinct slicing tree/floorplan. Further, $\mathcal{F}$ is in increasing order of width and decreasing order of height by our method of construction. We define $(T_w, T_h) \leq (W_t, H_t)$ if $T_w \leq W_t$ and $T_h \leq H_t$ and $(T_w, T_h) > (W_t, H_t)$ if any of T_w or T_h or both is/are greater than the corresponding W_t and/or H_t.

Corollary 1: The time taken to generate the $O(\kappa n^2)$ slicing trees is atmost $O(\kappa n^2)$.

Proof: The number of slicing trees generated is $O(\kappa n^2)$ (see Theorem 1). A slicing tree at level i is produced in $O(\kappa.4^{i-1})$ time. With $O(\frac{n}{2^{i-1}})$ nodes at level i, generation of slicing trees at that level requires $O(\kappa n 2^{i-1})$ time. Therefore, the total time is $\sum_{i=1}^{\log_2 n}(\kappa n 2^{i-1}) = O(\kappa n^2)$. ■

In summary, we process the tree bottom-up only once, generating a set of slicing trees $\{(T_{w_i},T_{h_i})\}$ at the root. For all i such that $(T_{w_i},T_{h_i}) \leq (W_t, H_t)$ and aspect ratio is permissible, the floorplan corresponding to the i^{th} slicing tree is feasible. There may be slicing trees with $(T_{w_i},T_{h_i}) > (W_t, H_t)$, (i.e., apparently infeasible) due to the following reasons. First, the number of basic tiles for a module is such that these are sufficient to accomodate all the CLBs, RAMs and MULs, even though some of the resources within a few tiles may be unused. Second, to obtain sufficient number of rectangular shapes with better aspect ratios, we have added shapes with one extra tile for a module with T_{m_i} prime. However, as for each type of resource, the sum over all modules of their requirements is less than the total amount available on the target chip, Phase III is necessary for reallocating different types of resources by appropriate positioning of horizontal and vertical cut lines of the slicing topology. The shape of some of the modules may become rectilinear in order to get a feasible floorplan. Finally, if the basic tile definition for a target chip calls for use of a few fractional tiles to cover it entirely, Phase III becomes essential in finding a feasible floorplan.

D. Phase III: Realization of Slicing tree on Target FPGA

For every slicing tree generated in Phase II, we position the cut lines such that total resource requirements on its either side are satisfied within the target chip. This is performed recursively to determine the coordinates of the regions assigned to the modules in the floorplan. There are two steps: (i) greedy allocation of a rectangular region (GARR) which satisfies the CLB requirements, followed by (ii) allocation of RAM and MUL blocks in and around this region.

1) Greedy Allocation of Rectangular Region (GARR) to a module or cluster: Each slicing tree is traversed top-down level by level (level order), in left to right order from root towards the leaves, and a rectangular region $(x_p^{min}, y_p^{min}, x_p^{max}, y_p^{max})$ is allocated to every node p using the cut direction and the number of CLBs required at p. The entire floorplan rectangle $(0, 0, W, H)$ is allocated to the root node of the slicing tree. Let the CLB requirements at node p, its left child l and its right child r be p_{clb}, l_{clb} and r_{clb} respectively. If there is a vertical cut at node p and the number of CLB columns in the region assigned to p is p_{col}, then the number of CLB columns allocated to

the regions for l and r, l_{col} and r_{col} are given by

$$l_{col} = \frac{l_{clb}}{p_{clb}}.p_{col}; \qquad r_{col} = p_{col} - l_{col} \tag{5}$$

Similarly, for a horizontal cut at p with p_{row} rows of CLB, the number of CLB rows allocated to l and r are

$$l_{row} = \frac{l_{clb}}{p_{clb}}.p_{row}; \qquad r_{row} = p_{row} - l_{row} \tag{6}$$

The region for the left child l of a node p always inherits the top-left corner (x_l^{min}, y_l^{min}) of p. The bottom-right corner (x_l^{max}, y_l^{max}) is derived from the width and height computed as above. So, we get

$$x_l^{max} = x_l^{min} + l_{col}; \quad y_l^{max} = y_p^{max} - \text{ vertical cut at } p \tag{7}$$

$$x_l^{max} = x_p^{max}; \; y_l^{max} = y_p^{min} + l_{row} - \text{ horizontal cut at } p \tag{8}$$

The top-left corner (x_r^{min}, y_r^{min}) for the right child r is calculated as follows:

$$x_r^{min} = x_l^{max} + 1; \quad y_r^{min} = y_l^{min} - \text{ vertical cut at } p \tag{9}$$

$$x_r^{min} = x_l^{min}; \quad y_r^{min} = y_l^{max} + 1 - \text{ horizontal cut at } p \tag{10}$$

The bottom-right corner (x_r^{max}, y_r^{max}) for node r is calculated analogous to Equations 7 and 8 for vertical and horizontal cuts respectively. At the root, p_{col} and p_{row} are the number of CLB columns and rows in the chip, e.g. 80 and 104 respectively for the Spartan-3 Board XC3S5000. If p is a leaf node, there is a module or a cluster associated with p, to be implemented in the region $(x_p^{min}, y_p^{min}, x_p^{max}, y_p^{max})$. This region may have more CLBs than required by the module/cluster. To obtain a compact shape of the module/cluster, we calculate the number of rows l_{row} or r_{row} required to satisfy the CLB requirement as $\frac{l_{clb}}{l_{col}}$ or $\frac{r_{clb}}{r_{col}}$ respectively for vertical cut at its parent. Similarly, the number of columns l_{col} or r_{col} required to satisfy the requiremnt is $\frac{l_{clb}}{l_{row}}$ or $\frac{r_{clb}}{r_{row}}$ for a horizontal cut at its parent.

2) Allocation of RAM and MUL: A rectangle assigned as above to a module m_i has adequate CLBs but not necessarily enough RAM/MUL blocks required by m_i. The deficit in RAM/MULs may have to be met by those placed exterior to the (top and bottom) boundary of the rectangle, and this may be in a rectangle assigned to a neighbouring module m_k either above or below. If the RAM/MUL requirement of m_k is also not satisfied fully within its rectangle, then there may be a conflict. Therefore, the violations in RAM/MUL requirement constraints are resolved globally by formulating it as a *minimum cost maximum flow (MCMF)* problem, such that a module is not realized in disconnected regions. Below, for ease of understanding we consider only RAMs.

We define the flow network $G = (U, Z)$ as follows. Let $U = \{s, t\} \cup U_L \cup U_R$ where s and t are source and the sink vertex respectively and $U_L \cap U_R = \phi$. If a module m_i has RAM requirements, then there is a vertex $u_i \in U_L$. If a module has no RAM requirement, then there is no vertex corresponding to it in U_L. Each vertex $v_j \in U_R$ corresponds to a candidate RAM location on the target FPGA. Let the RAM requirement for a module m_i be ram_i. Suppose the rectangle $R_i = (x_i^{min}, y_i^{min}, x_i^{max}, y_i^{max})$ has been assigned to m_i. Then, for each column of RAM units intersecting the CLB assignment inside R, a *RAM strip* of module m_i is said to include the RAM locations

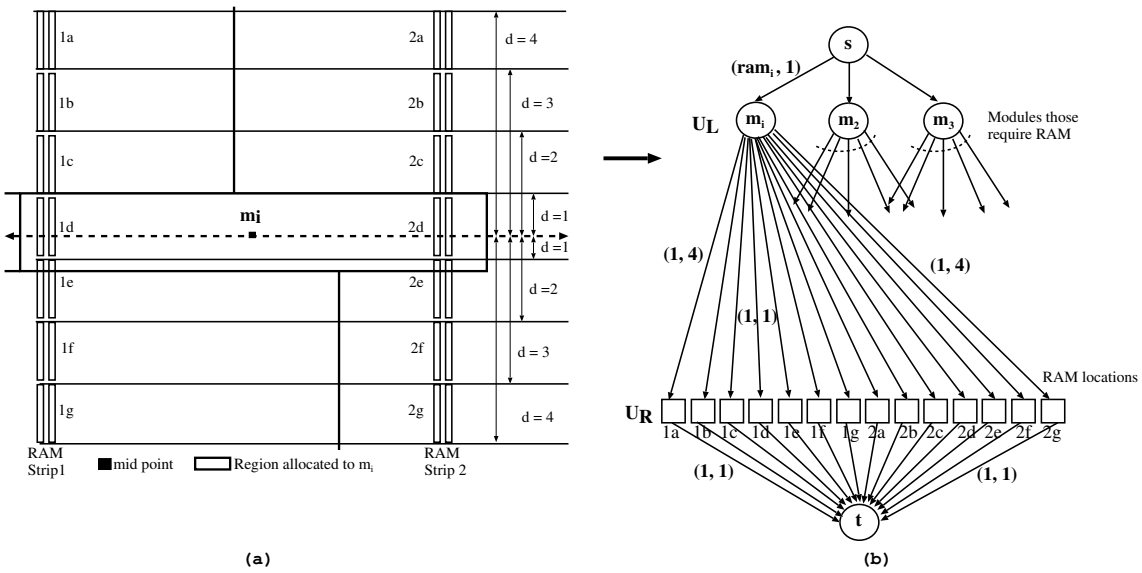


Fig. 5. (a) Candidate RAM/MUL location for a module m_i with requirement of 3 RAM units but has only 2 RAM units within its allocated region, (b) portion of the min-cost flow network corresponding to RAM/MUL allocation for module m_i. The labels (x, y) on the edges indicate capacity and cost respectively.

within R, along with ram_i locations above its top boundary and ram_i locations below its bottom boundary. There are three types of edges in Z, i.e. $Z = Z_1 \cup Z_2 \cup Z_3$.

$Z_1 = \{(s, u_i) | u_i \in U_L\}$; the capacity $c(s, u_i)$ of each edge (s, u_i) is ram_i. The cost of each edge is 1.

$Z_2 = \{(u_i, v_j) | u_i \in U_L$and $v_j \in U_R\}$ such that there is an edge between a vertex u_i corresponding to a module m_i and a vertex v_j belonging to the *RAM strip* of the module m_i. The capacity $c(u_i, v_j)$ of each edge is 1 and the cost of the edge is chosen as a rational value representing the vertical distance d from the center of rectangle R_i to that of the RAM location for v_j.

$Z_3 = \{(v_j, t) | v_j \in U_R\}$; the capacity $c(v_j, t)$ of each edge (v_j, t) is 1 and the cost is 1.

Figure 5 shows the candidate RAM/MUL locations for allocating the RAM/MUL required by module m_i. Suppose m_i requires 3 RAM units, but the region allocated to m_i has only 2 RAM units in two columns. In the figure, the seven RAM locations within each of RAM strip 1 and RAM strip 2, are the candidate RAM locations chosen for assigning the 3 RAM units required by m_i. The corresponding flow network is also shown, where from a vertex corresponding to m_i there are edges to all the fourteen RAM locations $1a, \ldots 1g, 2a, \ldots, 2g$.

We solve *MCMF* on G to assign RAMs to available RAM locations. We say the RAM assignment to be *order preserving* if two modules m_i and m_k vying for a RAM column with module m_i placed above m_k have their RAM allocations also in the same order.

Lemma 3: If the min-cost max-flow in G is equal to the total RAM requirement $\sum_{i=1}^{n} ram_i$, then the input is feasible. Also, a min-cost max-flow in G is order preserving for RAM assignment.

Proof: The first part of the lemma follows as a special case of the proof given in Observation 1 in Section VI

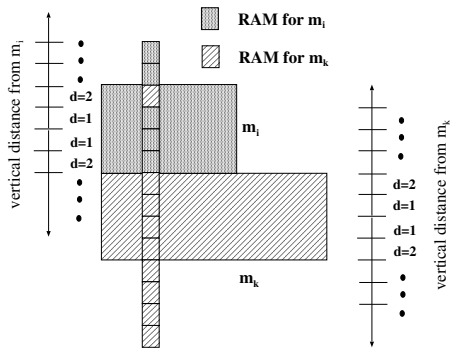


Fig. 6. Example of an allocation of RAM/MUL which is not order preserving (edge costs may be rational)

of the paper. Also, see [10].

For the second part, let module m_i lie above module m_k. Now, suppose by contradiction, there exists at least one RAM implementation of m_k (say y_i) above that of RAM implementation of m_i in the *MCMF* implementation. Refer Figure 6 for such an instance and also the costs which are same as the distance of that RAM location from the middle of the rectangle. As module m_i is above m_k, the cost of the edge into the vertex for y_i in G is more for m_k than m_i. This implies that we can swap y_i with any of its lower RAM implementations for m_k resulting in the same flow value but with a reduced cost. Hence, the flow was not of minimum cost and thus we have a contradiction. ■

Excessively large RAM requirements of adjacent modules might yield an infeasible MCMF. This implies that the chosen slicing topology is inappropriate for a feasible realization and is hence rejected. If there is no slicing tree with feasible MCMF, a different relative ordering of modules in Phase I is required. The MUL units are also assigned to the physical locations in a similar manner by solving a separate *MCMF*.

Lemma 4: The *MCMF* takes $O(H^2 \log^2 H)$ time.

Proof: Let R_i be the rectangle assigned for module m_i having l_i RAMs. So, the number of edges from vertex $u_i \in U_L$ corresponding to m_i is $l_i + 2 \times ram_i$. Therefore, the total number of edges is $\sum_{i=1}^{n}(l_i + 2 \times ram_i)$. Surely, both the total number of RAMs and the total number of RAMs enclosed within the non-overlapping rectangles has to be $O(H)$. So, $\mid Z \mid = O(H)$. Also, $\mid U \mid = O(H)$. The *MCMF* would take $O(\mid Z \mid \log \mid U \mid (\mid Z \mid + \mid U \mid \log \mid U \mid))$ time [26]. With $\mid Z \mid = \mid U \mid = O(H)$, the time complexity of *MCMF* is $O(H^2 \log^2 H)$. ■

The RAM/MUL allocation as done here by solving an *MCMF* is conceptually similar to the network flow formulation of Feng and Mehta [10] as both tries to solve the violations in RAM/MUL requirement constraints globally. But our formulation is done based on one-dimensional geometry, we ensure order preserving assignment (vide Lemma 3). Also, our formulation leads to a network of size linear in H because of the way we define and use a *RAM strip*. The size of the flow network as designed by Feng and Mehta [10] is $O(\mathcal{R})$, where $\mathcal{R}$ is the total number of CLBs, RAMs and MULs on the board, i.e., $W * H$, where W and H are the width and height of the board respectively. CLBs, which are in majority on the board, do not come into our complexity.

Allocation of regions to modules of a cluster : At the end of Phase III, we have regions assigned to each module and/or to the cluster of modules. The RAM/MUL required by each module/cluster is also assigned to RAM/MUL locations on the chip. However, the module shapes within the cluster is not yet decided. The assignment of modules of a cluster in a predefined region having CLBs, RAMs and MULs is a floorplanning problem of smaller dimension. However, in our case, this region corresponding to a cluster is very small and is atmost the size of a basic tile consisting of 80 CLBs, 1 RAM and 1 MUL. Thus we employ a greedy heuristic to implement the modules inside this region. First we place any module having RAM and/or MUL requirement next to the RAM/MUL columns in the region, then we place the rest of the modules of the cluster in the remaining available space in that region.

The process of CLB assignment followed by RAM and MUL assignment is carried out for every slicing tree generated in Phase II. The HPWL is calculated for each floorplan generated. Since RAM/MUL columns are pre-placed on FPGA, 2D compaction of the entire floorplan becomes a challenging task keeping the minimal wirelength. However compaction such as in [11] can be applied to *HeteroFloorplan* as well, though it might change the aspect ratio of modules and thus the wirelength. The floorplans with minimal wirelength and no discontinuity of modules are reported as feasible floorplans.

E. Time complexity of HeteroFloorplan

The overview of our floorplanner is given in Algorithm 1.

Theorem 2: The time complexity of *HeteroFloorplan*, excluding Phase I, the recursive balanced bipartitioning is $O(\kappa n^3 + \kappa n^2 H^2 \log^2 H)$, where H is the height of the chip, n is the number of modules and κ is the maximum number of shapes generated.

Proof: The clustering step is $O(n^2)$ with the number of modules n. By Corollary 1, the time taken for generating $O(\kappa n^2)$ slicing trees is $O(\kappa n^2)$. For each of the slicing trees, we traverse the tree of size $O(n)$ from root to leaves in order to fix the rectangular regions of the CLBs in $O(n)$ time. Then, we solve a *MCMF* to assign RAM/MULs in $O(H^2 \log^2 H)$ time (by Lemma 4). The total time complexity is thus $O(\kappa n^2(n + H^2 \log^2 H))$, i.e, $O(\kappa n^3 + \kappa n^2 H^2 \log^2 H)$. ■

The recursive bi-partitioner is iterative because of the use of *hMeTis*. But, the authors of *hMetis* in [21] claims that the time taken by *hMeTis* is almost linear in the number of hyperedges, i.e., netlists. It may be noted that the value of κ is very small, typically 6 to 10 for a floorplanning problem with hundreds of modules. The reason behind this is that as the value of n increases, the maximum value of κ decreases. Thus, *HeteroFloorplan* taking $O(\kappa n^3 + \kappa n^2 H^2 \log^2 H)$, compares favorably against that of [9] which the authors claim to take $O(W^2 n^5 \log n)$ time (our analysis shows that the $\log n$ term should be replaced by $\sqrt{n}$). It may be noted here that as the other two methods, viz. [8] and [10] use simulated annealing, it is not possible to compare the time complexity of *HeteroFloorplan* with these.

Algorithm 1: HeteroFloorplan

Input : Module netlist S, list of module resource requirements ρ_m, FPGA architecture $(0, 0, W, H)$ and its basic tile A

Output: A feasible floorplan

Preprocessing:

Compute W_t and H_t;

for *each module* m_i **do**
  find the minimum number of tiles required (T_{m_i}) from the resource requirement vector ρ_{m_i};

if T_{tot}*, total tiles required* $> (W_t * H_t)$ *yet resources are available* **then**
  cluster small modules with resource requirement less than that in basic tile A, into minimal set of clusters

Phase I: Generation of Partition Tree

Recursively bipartition the netlist of modules/clusters based on balanced min-cut;

%This partition tree is used as a template for slicing topologies.%

Phase II: Generation of Floorplan Topologies

Enumerate shapes (w, h) in terms of the number of tiles;

Determine slicing topology and shapes by vertical and horizontal node sizing by post-order traversal of the partition tree of Phase I.

%The output is a set of slicing topologies.%

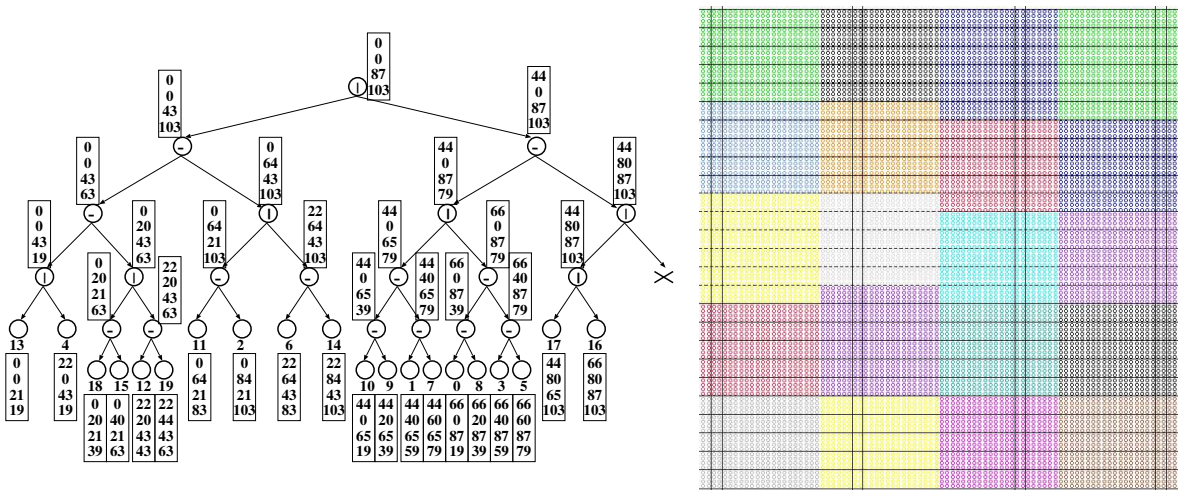
Phase III: Realization of Slicing Tree on FPGA

for *each slicing tree* **do**
  GARR: Traverse in level order and allocate rectangular region to each module based on CLB requirement;
  Allocate pre-placed RAM/MULs to modules by min-cost max-flow (MCMF);

Report feasible floorplan;

IV. An Example

Here we explain our floorplanning method with a synthetic example circuit taken from [8], [11]. Cheng and Wong [8] devised an experiment as follows. They took XC3S5000 which is the largest chip in the Xilinx [14] Spartan 3 family. They divided the XC3S5000 almost evenly into 20 blocks, each corresponding to a module. Of the 20 modules, 16 need 400 CLBs, 5 RAMs and 5 MULs each, and the rest of the 4 modules need 480 CLBs, 6 RAMs and 6 MULs each. The authors claim this to be a very tight problem. A little bit of inefficiency on the part of the floorplanning method could render the floorplan to be infeasible. The method *HeteroFloorplan* is explained with this tight problem of [8]. We considered the same circuit from [8] with 20 modules and constructed an appropriate netlist for comparison purpose. Figure 7(a) shows the binary partition tree obtained in Phase I of *HeteroFloorplan*. The integers $0, \ldots, 19$ written just below the leaves, indicate the indices of the modules. A set of slicing trees is generated in Phase II. One such slicing tree with its vertical and horizontal cut lines marked at every internal node, is shown here. Finally, the realization of the slicing tree onto the coordinates of the target architecture in terms



(a) (b)

Fig. 7. An example circuit of [8]: (a) one of its slicing trees and the box next to each node gives the four co-ordinates $(x^{min}, y^{min}, x^{max}, y^{max})$ of the region allocated to the node, (b) floorplan produced by *HeteroFloorplan*.

TABLE I

FLOORPLAN RESULTS FOR *20-module example* [8]

Index of slicing tree	1	2	3	4	5	6	7	8	9	10
(T_w,T_h)	$(1, 104)$	$(2, 52)$	$(3, 39)$	$(4, 26)$	$(5, 22)$	$(6, 20)$	$(7, 17)$	$(8, 14)$	$(9, 13)$	$(10, 11)$
Wirelength(HPWL)	392	560	X	816	X	X	X	X	X	X
Avg. aspect ratio	17.01	4.25	X	1.06	X	X	X	X	X	X

of $(x^{min}, y^{min}, x^{max}, y^{max})$ are reported in a vertical box below every node. For example, the root is realized as (0,0,87,103), i.e., the entire target architecture. Within the floorplan area, a module, say m_{11}, is realized as $(0, 64, 21, 83)$. Figure 7 (b) shows the final allocation for each module on Spartan-3 XC3S5000.

The effectiveness of *HeteroFloorplan* is amply demonstrated with the 20-module example circuit of [8] that covers the entire target architecture. Table I shows the result obtained by *HeteroFloorplan* for the example circuit. The row (T_w, T_h) is the width and height (in terms of basic tiles) of each floorplan topology generated after phase II. The row marked wirelength (HPWL) shows the wirelength obtained for each slicing tree realization after phase III. A X mark in WL indicates that the corresponding slicing tree is infeasible. Note that the 4^{th} slicing tree has $(T_w, T_h) = (4, 26) = (W_t, H_t)$. Hence, for this topology, we need not execute Phase III. However, we execute Phase III on all the slicing topologies generated even if their $(T_w, T_h) > (W_t, H_t)$. As mentioned in last part of Section III-C3, the tiles remain under utilized due to the mismatch in three different resource requirements of a module. It may be possible that by modifying the cutlines, the topology could be made feasible. Thus, we execute phase III on all the topologies to get a set of feasible floorplans. This is shown in Table I, where topology configuration 1 and 2 with $(T_w, T_h) > (W_t, H_t)$ has become feasible after execution of Phase III. However, topologies with $T_h < H_t$

TABLE II
BENCHMARK CIRCUITS, C: CLB, R:RAM, M:MUL

	Ckt Characteristic		
Circuit	**#Modules**	**#Nets**	**#(C, R, M)**
ideal	20	18	$(8320, 104, 104)$
apte	9	44	$(6614, 70, 70)$
xerox	10	183	$(6625, 66, 50)$
hp	11	44	$(6591, 66, 66)$
ami33	33	84	$(6289, 61, 60)$
ami49	49	377	$(6300, 63, 63)$
n100a	100	576	$(6352, 39, 38)$
n200a	200	1585	$(6342, 44, 34)$
n300a	300	1893	$(6399, 65, 54)$

and $T_w > W_t$ usually lead to infeasible solutions. The time taken by *HeteroFloorplan* to generate the floorplans for all the 10 slicing trees is 2.98 seconds on a slower 1.2GHz SunBlade 2000, which is far less than 88 seconds taken by [8] on a faster 2.4GHz Intel (R) Xeon CPU. We observed that *HeteroFloorplan* could construct the same floorplan reported in [8] with an appropriate partition tree. In Table I, the topology in column 5 (slicing tree index 4), is identical to that reported in [8]. Since [8] does not report the wirelength for this 20 module example, we can not compare it with ours. Further, many of the slicing trees remained infeasible as the resource requirement is very tight. The average aspect ratio of module dimensions is shown in the last row of Table I. Although the first two floorplan realizations have smaller wirelength, the average aspect ratio of the modules are far off from 1, whereas the realization $(4, 26)$ has the average aspect ratio very close to 1 and we report this as the final feasible floorplan.

V. EXPERIMENTAL RESULTS ON BENCHMARKS

We have implemented the proposed method in C using LEDA library[25] on 1.2GHz SunBlade 2000 workstation with SunOS Release 5.8. *HeteroFloorplan* is tested on Xilinx XC3S5000 (Spartan-3) FPGA with 8320 CLBs, 104 RAMs and 104 MULs. These are arranged in 88 columns (including 4 RAM-MUL column pairs) and 104 rows of CLBs. As mentioned earlier, the basic tile size is chosen as A = $(20 \times 4, 1, 1)$, i.e., 80 CLBs, 1 RAM and 1 MUL. Experimental results on 9 benchmark circuits derived from MCNC [8] and GSRC Bookshelf ASIC floorplanning benchmarks [27] are reported here. ASIC benchmarks are converted to FPGA benchmarks as in [10] by proportional CLB requirements.

Table II has the details of the 9 benchmark circuits, namely, the number of modules, the signal nets, the total requirements of the three types of resources, in columns 2, 3 and 4 respectively.

Table III shows the comparative results of wirelength (HPWL) obtained by *HeteroFloorplan* and by that in [10]. The second column reports that (T_w, T_h) in Phase II for which we obtained a feasible solution with aspect ratio closest to 1. Note that $(T_w, T_h) > (W_t, H_t)$ in many of the cases. As explained in the previous section, this is due

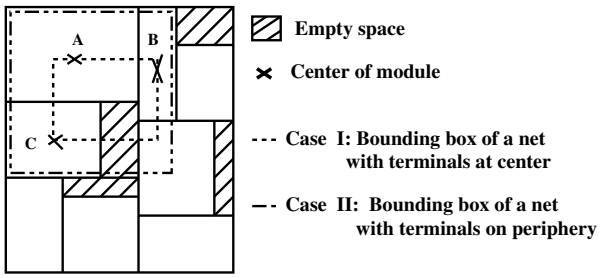


Fig. 8. Computation of bounding box of a net with terminals on modules marked A, B and C

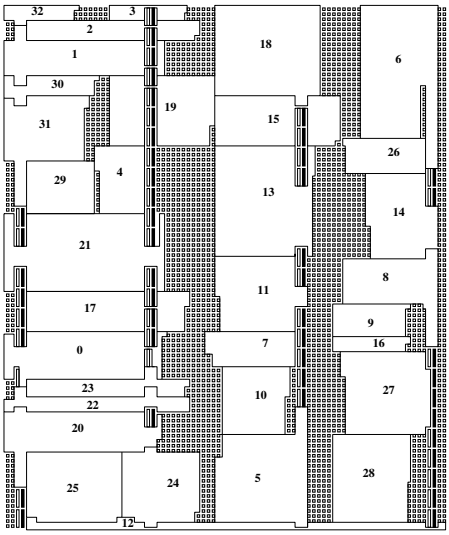


Fig. 9. Floorplan of ami33 after phase III for $(T_w, T_h) = (4, 30)$

to mismatch between the resources available in tile and variations in type of resource requirements of modules. By executing Phase III, the floorplan becomes feasible. One such floorplan of the circuit *ami33* after Phase III is shown in figure 9. The wirelength HPWL can be computed with the assumptions that the terminals are either (Case I) at the center of the modules, or (Case II) on the periphery of the modules, as shown in Figure 8. For the FPGA benchmarks available, the terminal locations on the modules were not known to us. Thus we have computed center-to-center HPWL as in case I for all the benchmark circuits and reported them in column 3 of Table III. Since the method in [10] employs SA based PARQUET [30], [31], we assume that the wirelength reported there is computed as in case II (terminals at periphery). For comparison of *HeteroFloorplan* with theirs, we take the worst case scenario by considering the terminals to be at the extreme corners, i.e., the top-left and the bottom-right corner, of the enclosing bounding box of the net. As expected, this is much larger than center-to-center HPWL. This worst case HPWL as in Case II, is reported in column 4 of Table III. As [10] computes the wirlength on the ASIC dimension, we have also scaled the wirelength from FPGA board to ASIC dimension. Columns 5 reports the wirelength (HPWL) directly from [10]. We compare the percentage gain in wirelength both w.r.t Case I and

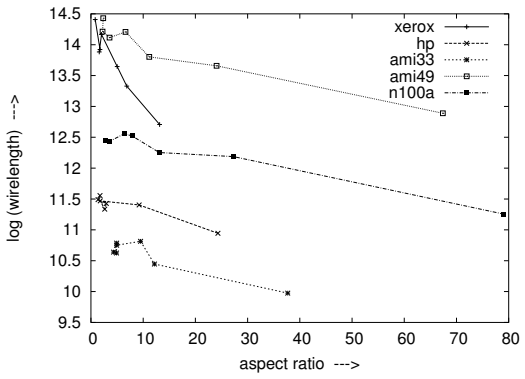


Fig. 10. Variation of HPWL (Case I) with average aspect ratio of a module

TABLE III

COMPARISON OF WIRELENGTH (HPWL): *HeteroFloorplan* VS. [10] CASE I: TERMINALS AT CENTER; CASE II: TERMINALS ON PERIPHERY

	HeteroFloorplan			Method in [10]		
		HPWL		HPWL	% Gain	
Ckt	(T_w, T_h)	I	II	II	I	II
ideal	$(4, 26)$	816	1605	–	–	–
apte	$(4, 22)$	189720	441801	–	–	–
xerox	$(4, 28)$	919485	1636096	–	–	–
hp	$(4, 26)$	106170	194597	–	–	–
ami33	$(4, 30)$	42623	87592	89283	52	20
ami49	$(3, 45)$	950322	1242337	1173000	18	-5
n100a	$(4, 39)$	209456	371339	358338	41	-3
n200a	$(4, 38)$	439154	594254	700045	37	15
n300a	$(4, 42)$	690391	886015	875602	21	-1

Case II in columns 6 and 7 respectively. We observed that the gain in wirelength ranges between 18% to 52% in Case I (comparing columns 3 and 5) and -5% to 20% in Case II (comparing column 4 and 5). On the average, the improvement for Case I is 34% and for Case II is 5% with our calculation based on worst-case scenario.

We have also compared our HPWL with that of [11] in Table IV, where center-to-center HPWL is considered on the 88×104 XC3S5000 board itself. Column 2 shows the same (T_w, T_h) as in Table III. The center-to-center HPWL on the same FPGA board obtained by *HeteroFloorplan* is shown in column 3. The wirelength obtained by [11] is reported in column 4. We observed from column 5 that the percentage gain in HPWL ranges from 3% to 14%. Here, we have reported the wirelengths of such (T_w, T_h) where the average aspect ratio over all modules ranges from 1 to 13. Further, by *HeteroFloorplan* the user has the option of selecting a floorplan from a set of slicing trees with a trade-off between aspect ratio and wirelength, as desired. Figure 10 shows how the wirelength increases as the aspect ratios aproach 1, for a subset of circuits.

TABLE IV
COMPARISON OF WIRELENGTHS(HPWL): *HeteroFloorplan* VS. [11]

	HeteroFloorplan		Method in [11]	
Ckt	(T_w, T_h)	HPWL	HPWL	% Gain
ideal	$(4, 26)$	758	–	–
apte	$(4, 22)$	2599	2704	3
xerox	$(4, 28)$	9187	10476	12
hp	$(4, 26)$	2732	3123	12
ami33	$(4, 30)$	3644	4114	11
ami49	$(3, 45)$	13336	15311	12
n100a	$(4, 39)$	25896	30295	14
n200a	$(4, 38)$	58586	–	–
n300a	$(4, 42)$	72820	–	–

Table V shows the CPU time (in secs.) taken to produce the floorplans. In column 5, we report the time taken by *HeteroFloorplan* on 1.2GHz SunBlade 2000 workstation, which is much slower than the platform reported in [10], [11]. For a fair comparison with their work, we adopt the following steps. Since both methods employ simulated annealing based fixed outline ASIC floorplanner PARQUET [30], [31], we execute the default PARQUET (without wirelength minimization) on our platform and report the runtime in column 2. We observed that PARQUET with wirelength minimization takes longer execution time. Feng et al. employs another step called Constrained Floorplanning (CF) based on min-cost max-flow formulation. We have implemented the min-cost max-flow on our platform using LEDA library[25]. The time taken to execute this step is reported in column 3. The time taken by *HeteroFloorplan* is given in column 5 along with the corresponding number of slicing trees generated in Phase II, in column 4. From the experiments, we conclude that the combined step of PARQUET and CF is $2\times$ to $373\times$ slower depending on the size of the circuit. As [11] is also based on simulated annealing and a compaction step followed by it, the time taken must be more than the time taken by PARQUET. Hence *HeteroFloorplan* must be faster than [11] in the same order of magnitude as [10]. We observed that not all these slicing trees generated by *HeteroFloorplan* are really necessary, hence we can prune the slicing trees, thereby reducing the time even further. This shows the suitability of *HeteroFloorplan* for fast FPGA floorplanning.

VI. HOW GOOD IS OUR GARR?

In this section, we evaluate our greedy heuristic of positioning the cut line in the slicing topology based on the resource requirements on either side of that cut line, against a network flow based formulation. Our greedy strategy is locally optimal as it determines the location of a particular cut line at node p from the ratio of the CLB requirements of the left and right children of p.

TABLE V
COMPARISON OF CPU TIME

	Time(s) [10]		*HeteroFloorplan*	
Circuit	**Parquet**	**CF**	**#slicing trees**	**Time(s)**
ideal	–	–	20	2.98
apte	1.78	0.61	16	1.22
xerox	2.1	0.61	20	1.02
hp	2.57	0.61	18	0.96
ami33	19.9	0.55	23	1.39
ami49	43.12	0.61	23	3.84
n100a	92.31	0.95	24	1.16
n200a	695.92	0.92.	32	2.6
n300a	1605.07	1.02	39	4.3

A. Max-flow Formulation for CLB Allocation

For every slicing tree generated in Phase II of *HeteroFloorplan*, the problem is to allocate a rectangular region to each module satisfying its CLB requirements and without causing any discontinuity of the modules. Had there been no bounds on W and H of the target FPGA architecture, the slicing tree could have been easily realized on a (W_t, H_t) floorplan as stated earlier at the end of section III-C3. But given the fixed size constraint, we start with a (W_t, H_t) realization, and linearly scale it down to the given target $(0, 0, W, H)$. Let us denote such a realization by $\mathcal{S}$.

Two salient issues are to be raised here: (i) whitespaces are created in $\mathcal{S}$ during node sizing in floorplan topology generation; (ii) linear scaling down may not guarantee satisfaction of resource requirements of each module within the rectangle assigned to it. Reallocation of resources can be modeled as a network flow problem as discussed next. Feng et al. [19] used a similar kind of network flow formulation in their work on constrained floorplanning in ASICs. As shown in Figure 11 (a), in $\mathcal{S}$, the $(0, 0, W, H)$ target chip is dissected into rectangles, n of which are earmarked for the modules and the remaining are whitespaces. For a rectangle r_j earmarked for a module m_i, let the available resource in r_j be ra_j and the resource requirement of m_i be rr_i. For an empty rectangle, only ra_j is associated with it.

We define the flow network $G = (V, E)$ as follows. Let $V = \{s, t\} \cup V_L \cup V_R$ where s and t are the source and the sink vertices respectively. Each vertex $v_i \in V_L$ corresponds to the i^{th} module to be floorplanned. Each vertex $v_j \in V_R$ corresponds to the j^{th} rectangle as shown in Figure 11(b). Certain rectangles are earmarked for modules; the rest are whitespaces that are to be utilized for reallocation of CLBs to satisfy resource requirements. There are three types of edges in E, i.e. $E = E_1 \cup E_2 \cup E_3$.

E_1: $\{(s, v_i)|v_i \in V_L\}$; capacity $c(s, v_i)$ of edge (s, v_i) is rr_i.

E_2: $\{(v_i, v_j)|v_i \in V_L$and $v_j \in V_R\}$ such that $i = j$, or r_i and r_j are adjacent horizontally, vertically or diagonally; capacity $c(v_i, v_j)$ of edge (v_i, v_j) is ra_j.

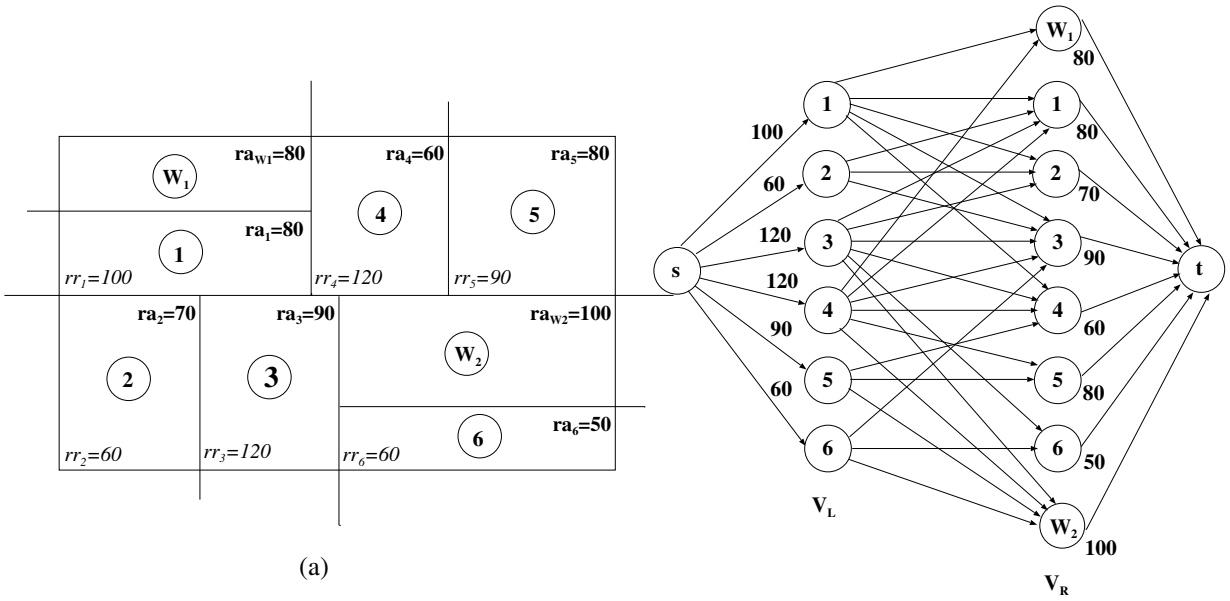


Fig. 11. (a) A linearly scaled down realization of a node-sized slicing tree to fit an $(0, 0, W, H)$ architecture. Rectangles marked $1, \cdots, 6$ denote the module realizations; W_1 and W_2 denote the whitespaces. ra_i (rr_i) denotes resource available (requirement). (b) Flow network corresponding to the realization shown in (a). While the capacity for each edge from s to V_L is shown next to the edge, that for any edge incident on a vertex in V_R is marked beside the vertex.

E_3: $\{(v_j, t)|v_j \in V_R\}$; capacity $c(v_j, t)$ of edge (v_j, t) is ra_j.

Figure 11(b) shows the flow graph for the realization in Figure 11(a). Note that, the capacity $c(v_i, v_j)$ of edge $(v_i, v_j) \in E_2$ is assigned in such a way that the flow $f(v_i, v_j)$ is equal to the whole or part of the CLB requirements of module i met by rectangle j.

Definition 4: A realization $\mathcal{S}$ of b rectangles and n modules is said to be *a feasible realization* of a given floorplan problem if $\sum_{i=1}^{n} rr_i \leq \sum_{j=1}^{b} ra_j$ and the module requirements are met by the resource available with the rectangles.

Along the lines of [19], we have the following observation.

Observation 1: Let f_{max} be the maximum flow in the network G corresponding to the realization $\mathcal{S}$. If $f_{max} = \sum_{i=1}^{n} rr_i$, then $\mathcal{S}$ is feasible.

Proof: Consider a cut $C = (\{s\}, \{V_L \cup V_R \cup t\})$. The flow f across C signifies the part or whole of the resource requirements of all modules that have been satisfied by the resources available in all the rectangles. Thus,

$$f \leq \sum_{v_i \in V_L} c(s, v_i) = \sum_{i=1}^{n} rr_i \tag{11}$$

Let $F(V_L, v_j)$ be the sum of flows into $v_j \in V_2$ from all $v_i \in V_L$ which signifies the allocation of resources in rectangle j. Also,

$$f(v_j, t) \leq c(v_j, t) \quad v_j \in V_R \tag{12}$$

Due to conservation of flow at v_j,

$$F(V_L, v_j) = f(v_j, t) \tag{13}$$

From Equations 12 and 13 and that $c(v_j, t) = ra_j$,

$$F(V_L, v_j) \leq c(v_j, t) = ra_j \tag{14}$$

Equation 14 signifies that the resource allocated for any rectangle j is at most ra_j, which is the resource available in rectangle j. Now, if $f_{max} = \sum_{i=1}^{n} rr_i$, then we can claim that the resource requirements of all modules have been met and by Equation 14, we know that the resource allocation to each rectangle does not exceed its available resources. Thus, if $f_{max} = \sum_{i=1}^{n} rr_i$, $\mathcal{S}$ is feasible because the total resource requirement $(\sum_{i=1}^{n} rr_i)$ has been met. ■

By flow computation, if we obtain a feasible $\mathcal{S}$, then from the non-zero flow values at each vertex $v_j \in V_R$ we get the subset of modules which make use of the available resource (ra_j) of rectangle j. Suppose, these non-zero flows into v_j originate from a set of vertices $V_L^s \subseteq V_L$. Then, the available resource ra_j of rectangle j is to be allocated to each module m_i corresponding to a vertex in V_L^s. In this manner, a module m_i can be assigned to one or more neighboring rectangles (*aka* edge set E_2 of network G), with a condition that it does not become discontinuous. It may be observed that each rectangle j has to be partitioned among the modules (part or whole) corresponding to V_L^s, which itself is a floorplanning problem of a smaller size. This is the reason for which our proposed method has not adopted network flow approach. However, this network flow based formulation gives us a global picture of the realization of a slicing tree, against which we can validate our greedy heuristic.

B. Comparison

Let $M = \{m_1, m_2, \ldots, m_n\}$ be a set of n distinct modules with CLB requirements being $\{c_1, c_2, \ldots, c_n\}$ respectively. Let c_{ij}^o and c_{ij}^{nf} be the number of CLBs of module m_i that are contained in rectangle j by *HeteroFloorplan* and the network flow based method respectively. As both c_{ij}^o and c_{ij}^{nf} may be 0, we introduce two new boolean variables x_{ij}^o and x_{ij}^{nf}; $x_{ij}^o = 0$ ($x_{ij}^{nf} = 0$) means module m_i is not implemented within rectangle j by *HeteroFloorplan* (network flow based method); similarly, $x_{ij}^o = 1$ ($x_{ij}^{nf} = 1$) means module m_i or a part of it has been implemented in rectangle j by *HeteroFloorplan* (network flow based method). In order to measure the similarity between the CLB distribution of our floorplan and the network flow based one, we define a metric χ for percentage of match as follows:

$$\chi = (1 - \psi).100 \tag{15}$$

where,

$$\psi = \frac{1}{b} \sum_{j=1}^{b} \frac{1}{\sum_{i=1}^{n} (x_{ij}^o \vee x_{ij}^{nf})} \sum_{i=1}^{n} \frac{|c_{ij}^o - c_{ij}^{nf}|}{c_i} \tag{16}$$

The parameter ψ takes real value in $[0, 1]$. $\psi = 0$ indicates an absolute match between the CLB distribution of our floorplan and the network flow based one and $\psi = 1$ indicates an absolute mismatch. Thus χ represents the matching percentage of CLB distribution between *HeteroFloorplan* and the network flow method. Table VI

TABLE VI

SIMILARITY OF CLB ALLOCATION BY *HeteroFloorplan* WITH NETWORK FLOW BASED METHOD

Circuit	**#rectangles**	χ **%**
ideal	20	96
apte	10.5	83
xerox	11.9	81
hp	12.8	76
ami33	41.4	75
ami49	57.8	63
n100a	110.0	65
n200a	219.5	63
n300a	219.1	62

presents the value of χ for the different circuits under consideration. The second column lists the average number of rectangles across all floorplan topologies and the third column shows χ. As can be observed from the values of χ ranging from 62% to 96%, *HeteroFloorplan* follows to a great extent, resource allocation in the network flow based method. Furthermore, due to inclusion of the diagonal neighbours in the edge set E_2, the CLBs of the modules may be distributed among all its neighbors. But, the realization by Phase III of our proposed method in Section III is by horizontal and vertical cut lines; thus retaining mainly rectangular shapes. This explains the reason for not having a very high value of χ in all cases.

VII. CONCLUDING REMARKS

In this paper, we have developed a fast floorplanning methodology for FPGAs with heterogeneous resources consisting of CLBs, RAMs and Multipliers as in Spartan-3 FPGA architectures. We propose a deterministic three phase method *HeteroFloorplan* for unified floorplan topology generation and sizing for such heterogeneous FPGAs. The time complexity of our approach excluding the recursive balanced bi-partitioning by $hMetis$ in Phase I, is $O(\kappa n^3 + \kappa n^2 H^2 \log^2 H)$, where H is the height of the chip, n the number of modules and κ the maximum number of shapes generated. This is an improvement over the other deterministic heuristic presented in [9]. Experimental results demonstrate a speed-up in the range of $2\times$ to $373\times$, depending on the size of circuit, over the existing methods, and an improvement on the average, of 34% in wirelength. Further, we evaluated our greedy resource allocation $GARR$ against a network flow based formulation to establish that $GARR$ measures up to the global allocation by max-flow method. A two-dimensional compaction step in the presence of pre-placed resources without sacrificing the wirelength is a challenging task and needs deeper investigation. Our method can be used for other architectures by selecting appropriate tile structure. Floorplanning on more irregular architectures with two or atmost three different basic tiles is being studied presently.

APPENDIX

The time complexity deduced in Theorem 2 is $O(\kappa n(n + H^{1.5}))$ where $\kappa = max\{t_j\}$ is the maximum number of shapes generated for any module m_j in terms of basic tiles. Let the size (number of CLBs) of the basic tile be c. So, the number of basic tiles being targeted onto the $W * H$ chip is $\lceil \frac{W*H}{c} \rceil$. Thus, ν the maximum number of tiles a rectangular module can take is $\lceil \frac{W \times H}{c} \rceil - (n-1)$, where n is the number of modules in the floorplan. The maximum number of rectangular shapes κ that can be generated for such a module is equal to the number of distinct factor pairs of ν.

From basic number theory [28], [29], any natural number ν has a unique prime factorization given by

$$\nu = p_1^{r_1}.p_2^{r_2}.\ldots.p_k^{r_k} \tag{17}$$

where each p_i is a prime with $p_1 < p_2 < \ldots < p_k$. Then, γ the number of all integer factors of ν is given by

$$\gamma = (r_1 + 1).(r_2 + 1).\ldots.(r_k + 1). \tag{18}$$

By taking logarithm of both sides of Equation 17, we get

$$\log \nu = \sum_{i=1}^{k} r_i \log p_i \tag{19}$$

As each r_i and p_i are positive, for all i, we have $1 \leq r_i \leq O(\log \nu)$, i.e, $r_i \leq O(\log \nu)$. So, Equation 18 becomes $\gamma = O((\log \nu)^k)$. Moreover, both sides of Equation 19 tally only if the number of terms on the right hand side, i.e., the number of primes $k \leq O(\log \nu)$. Hence, the upper bound on the number of factor pairs of ν, and therefore κ is given by $O((\log \nu)^{\log \nu})$.

ACKNOWLEDGEMENT

We thank Lei Cheng for the FPGA benchmark circuits, Jarrod Roy for providing the PARQUET software, and the anonymous reviewers for their incisive and critical comments.

REFERENCES

[1] V. Betz and J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research", in 7^{th} *International Workshop on Field Programmable Logic and Applications*, pp. 213-222, 1997.

[2] J. M. Emmert, and D. K. Bhatia, "Tabu Search: Ultra-Fast Placement for FPGAs", in *Proc. 9th Intl. Workshop on Field Programmable Logic*, pp. 81-90, 1999.

[3] P. Maidee, C. Ababei, K. Bazargan, "Timing-driven partitioning-based placement for island style FPGAs" ,*IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 24, no. 3, pp. 395-406, 2005.

[4] P. Banerjee, S. Bhattacharjee, S. Sur-Kolay, S. Das, S. C. Nandy, "Fast FPGA Placement using Space-filling Curve", in *Proc. FPL 2005*, pp. 415-420, 2005.

[5] R. Tessier, "Fast Placement Approaches for FPGAs", *ACM Transactions on Design Automation of Electronic Systems*, vol. 7, no. 2, pp. 284-305, 2002.

[6] M. Wang, A. Ranjan, and S. Raje, "Multi-Million Gate FPGA Physical Design Challenges", in *Proc. ICCAD*, pp. 891-898, 2003.

[7] J. M Emmert, A. Randhar, D. Bhatia, "Fast Floorplanning for FPGAs" in *Proc. FPL* , pp. 129-138, 1998.

[8] L. Cheng and Martin D. F. Wong, "Floorplan Design for Multi-Million Gate FPGAs", in *Proc. ACM ICCAD*, pp. 292-299, 2004.

[9] J. Yuan, S.Q. Dong, X.L. Hong, and Y.L. Wu, "LFF Algorithm for Heterogeneous FPGA Floorplanning," in *Proc. ASP-DAC*, pp. 1123-1126, 2005.

[10] Y. Feng and D. P. Mehta, "Heterogeneous Floorplanning for FPGAs" in *Proc. International Conference on VLSI Design*,pp. 257-262, 2006.

[11] L. Cheng and Martin D. F. Wong, "Floorplan Design for Multimillion Gate FPGAs", in *I*EEE Trans. on Computer Aided Design of Integrated Circuits and Systems, vol. 25, No. 12, pp. 2795-2805, Dec. 2006.

[12] L. Singhal and E. Bozorgzadeh, "Novel multi-layer floorplanning for Heterogeneous FPGAs", in *IEEE International Conference on Field Programmable Logic and Applications (FPL)*, pp. 613-616, August 2007.

[13] P. Banerjee, S. Sur-Kolay, and A. Bishnu, "Floorplanning in Modern FPGAs" in *Proc. International Conference on VLSI Design*, pp. 893-898, 2007.

[14] http://www.xilinx.com

[15] A. B. Kahng, "Classical Floorplanning Harmful?", *ISPD 2000*, pp. 207-213.

[16] L. J. Stockmeyer, "Optimal Orientations of Cells in Slicing Floorplan Designs", Information and Control, vol. 57, no. 2/3, pp. 91-101, 1983.

[17] R. H. J. M. Otten, "Automatic Floorplan Design", in *Proc. DAC*, pp. 261-267, 1982.

[18] M. de Berg, M. van Kreveld, M. Overmars and O. Schwarzkopf, "Computational Geometry: Algorithms and Application", Springer, 2000.

[19] Y. Feng, D. P. Mehta and H. Yang, "Constrained Floorplanning Using Network Flows", *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 4, pp. 572-580, 2004.

[20] http://www-users.cs.umn.edu/karypis/metis/hmetis

[21] G. Karypis, R. Aggarwal, V. Kumar and S. Shekhar, "Multilevel Hypergraph Partitioning: Applications in VLSI domain", *IEEE Trans. on VLSI*, vol. 7, no. 1, pp. 69-79, 1999.

[22] G. Even, Joseph S. Naor, S. Rao and B. Schieber, "Divide-and-conquer approximation algorithms via spreading metrics", *Journal of the ACM (JACM)*, vol. 47, no. 4, pp. 585-616, July 2000.

[23] M. Sarrafzadeh, C.K. Wong, "An Introduction to VLSI Physical Design", Mcgraw Hill, 1996.

[24] P. Dasgupta, S. Sur-Kolay and B. B. Bhattacharya, "A Unified Approach to Topology Generation and Optimal Sizing of Floorplans", in *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 17, no. 2, pp. 126-135, 1998.

[25] http://www.algorithmic-solutions.com/leda/index.htm

[26] R. Ahuja, T. Magnanti, and J. Orlin, *Network Flows: Theory, Algorithms and Applications*, Englewood Cliffs, NJ: Prentice Hall, 1993.

[27] http://www.cse.ucsc.edu/research/surf/GSRC/progress.html

[28] G. H. Hardy and E. M. Wright, "An Introduction to the Theory of Numbers", 5^{th} edition, Oxford, England, 1979.

[29] Ivan Niven, H. S. Zuckerman, H. L. Montgomery, "An Introduction to the Theory of Numbers", 5^{th} edition, Wiley, Singapore, 1991.

[30] S. N. Adya and I. L. Markov, "Fixed Outline Floorplanning: Enabling Hierarchical Design," *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, vol. 11, pp. 1120-1135, Dec. 2003.

[31] http://vlsicad.eecs.umich.edu/BK/parquet/

[32] J. A. Roy, S. N. Adya, David A. Papa, and I. L. Markov, "Min-Cut Floorplacement," *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, vol. 25, No. 7, pp. 1313-1326, Jul. 2006.