Floorplanning in Modern FPGAs

Pritha Banerjee, Susmita Sur-Kolay
Advanced Computing and Microelectronics Unit
Indian Statistical Institute
203 B. T. Road, Kolkata, India
{pritha_r,ssk}@isical.ac.in

Arijit Bishnu
Dept. of Computer Science and Engg.
Indian Institute of Technology
Kharagpur-721 302, India
Arijit.Bishnu@iitkgp.ac.in

***Abstract*— State-of-the-art FPGA architectures have millions of gates in CLBs, Block RAMs, and Multiplier blocks which can host fairly large designs. While their physical design calls for floorplanning, the traditional algorithm for ASIC do not suffice. In this paper, we have proposed an algorithm for unified floorplan topology generation and sizing for recent heterogeneous FPGAs. Experimental results on a set of benchmark circuits show that our three step floorplan generation method can produce feasible solutions very fast with 45% improvement in total half perimeter wirelength compared to the very few previous approaches.**

Keywords: *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, a homogeneous resource, were arranged in rows and columns uniformly, with Primary Input and Outputs (PI and PO) at the boundaries. 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 are evenly interspersed among CLB columns. Moreover, large design with millions of gates are partitioned into smaller modules for greater demand on performance and also reduction of compilation time for place and route. 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 skipped to map designs onto the earlier sea-of-gates style FPGAs. In a typical FPGA physical design flow, after technology-mapping, a flattened CLB netlist is directly placed [1] 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 [2]. But, for modules with heterogeneous resource requirements, neither this technique nor the traditional floorplanners for ASICs adapted to FPGA, are adequate [3]. 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 is merely a handful. Emmert et al. [4] devised a macro based floorplanning methodology for earlier generation sea-of-gates style FPGAs.

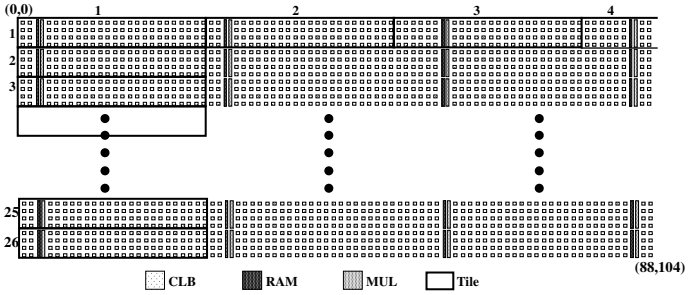

Fig. 1. Spartan-3 XC3S5000 FPGA Architecture

The method uses clustering techniques to combine macros into clusters, and then place the clusters with enhanced circuit routability and performance using Tabu search. Cheng and Wong [5] proposed the first floorplanning algorithm targeted for heterogeneous FPGAs that can produce feasible solution employing simulated annealing to optimize area, half-perimeter wirelength and the aspect ratios of modules. Yuan et al. [6] have proposed an algorithm based on Less Flexibility First (LFF) principle with worst case time complexity of $O(W^2n^5 \log n)$, where n is the number of modules and W is the width of the chip. Recently Feng and Mehta [7] presented a two step approach based on resource aware fixed outline simulated annealing starting from a given topology, followed by max-flow based constrained floorplanning to optimize wirelength.

In this paper, we propose a methodology for unified floorplan topology generation and sizing. The experimental results show that our method is fast and can produce floorplans with improved half-perimeter wirelength when compared to existing methods. The rest of the paper is organized as follows. In section II, we briefly describe the basic FPGA architecture followed by the problem definition and the objective function to be optimized. Section III presents our proposed methodology and time complexity issues. The proposed method is illustrated with an example in Section IV. Experimental results are reported in section V and concluding remarks appear in section VI.

II. Background

A. Architecture

While CLBs were arranged in rows and columns with routing wires laid out between rows and columns of CLBs earlier, modern FPGA architectures are far more heterogeneous with different types of resources to satisfy the varied

20th International Conference on VLSI Design (VLSID'07)
0-7695-2762-0/07 $20.00 © 2007 IEEE

design requirements. Fig. 1 shows a Xilinx Spartan-3 FPGA where the CLBs are arranged in columns interleaved with columns of RAM-Multiplier pair at certain intervals. Each small square represents a CLB. A pair of shaded rectangular blocks, spanning 4 CLB rows represents the RAM-MUL pair. Henceforth, we assume this architecture, although the methodology is applicable to other similar ones.

B. FPGA Floorplanning Problem

First, the basic terminology is given below.

Modules and Signal nets: Let $M = \{m_1, m_2, \cdots, m_n\}$ be a set of n distinct modules. Let $S = \{s_1, s_2, \cdots, s_q\}$ be a set of q signals. A set of distinct modules $M_{s_i} = \{m_j \mid m_j \in M\}$ is associated with each signal $s_i \in S$. s_i is called a *signal net*, and the set S is called a *netlist*. $M_{s_i} = M_{s_j}$ implies that there are two distinct signal nets connecting the same set of modules.

Resource Requirement Vector [5]: A 3-tuple vector R_m=(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 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 Fig. 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 $0 \leq y \leq H$.

FPGA Floorplanning Problem : Given a *target architecture* $(0, 0, W, H)$ with its resource locations, a set of modules M, the resource requirement vectors R_{m_i} for each $m_i \in M$, and the netlist S, assign regions $(x_{min}, y_{min}, x_{max}, y_{max})$ to each module such that
(i) $0 \leq x_{min} \leq x_{max} \leq W$ and $0 \leq y_{min} \leq y_{max} \leq H$,
(ii) no two modules overlap with each other,
(iii) R_{m_i} is satisfied for each module m_i,
(iv) a particular cost function is optimized.

A floorplan is *feasible* if it satisfies condition (i), (ii) and (iii). In this paper, the cost function used is the total half-perimeter wirelength, measured as in [7].

III. Proposed Method

Our floorplanning methodology as shown in Fig. 2, consists of: construction of a bi-partition tree, generation of floorplan topology [8][9] and realization of the topology using the resource requirements.

In the first phase, the target FPGA architecture is approximated to a $2D$ array of rectangular basic tiles. We generate a set of possible rectangular shapes (in terms of tiles) which satisfies the resource requirements and then bi-partition the netlist to obtain a binary partition tree of modules.

In the second phase, a list of candidate floorplan slicing topologies and corresponding module shapes are generated in polynomial time by appropriate sizing of the nodes in the bi-partition tree in postorder.

In the third phase, for every slicing tree obtained in the previous step, a rectangular region within $(0, 0, W, H)$ is assigned to every module which respects the cut direction and the resource requirements. Finally, the realization with the minimum wirelength among all the topologies generated is reported as the optimized floorplan.

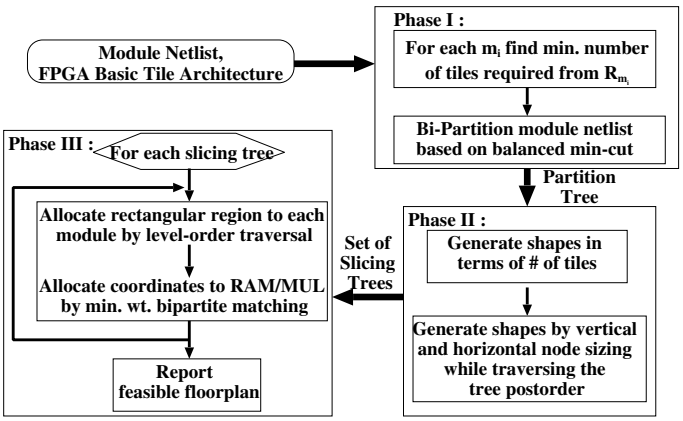

Fig. 2. Flow of our Floorplanning Method

A. Phase I: Generation of partition tree

In order to explain this step, we define the following terms.

Definition 1: A *Basic FPGA Tile* A = (a_{clb}, a_{ram}, a_{mul}) is a 3-tuple vector that represents the minimum number of CLBs, RAMs, MULs that constitute a basic tile which can be repeated horizontally and vertically to cover all the rows and columns of a given target architecture.
The given architecture is thus composed of, say, $T_w \times T_h$ *basic tiles* arranged in h rows and w columns. In Fig. 1, the basic tile A = (96, 1, 1) consists of 24×4 CLBs, 1 RAM and 1 MUL. The entire architecture (Spartan-3) shown in Fig. 1 can be covered by 26 rows and 4 columns of basic tile A.

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

$$T_{m_i} = \lceil max(\frac{m_{clb}}{a_{clb}}, \frac{m_{ram}}{a_{ram}}, \frac{m_{mul}}{a_{mul}}) \rceil \quad (1)$$

Min cut Partitioning of Module Netlist : In order to minimize the wirelength of the feasible floorplan, the module netlist is bi-partitioned recursively based on min-cut. The partitions are weight balanced across the cut edges according to the tile requirements of each module. We employ a state-of-the-art hypergraph partitioning tool *hMetis* [10] to obtain the weight balanced min-cut partitioning of the module netlist. This yields the relative ordering of the circuit modules which is the baseline of the floorplan generation.

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, $i = 1, 2 \cdots n$. An hyperedge $e = \{v_1, v_2, ...\} \in E$ corresponds to M_{s_i}. If $\{M_{s_i}, M_{s_j}, \cdots M_{s_l}\}$ are identical for signal net s_i, s_j, $\cdots$, s_l then each of $s_i, s_j, \cdots s_l$ corresponds to the same hyperedge with number of such signals as the weight of the hyperedge. The minimum number of tiles T_{m_i}, is the weight of the vertex $v \in V$. The hypergraph H thus generated, is bi-partitioned recursively to n parts, generating

a binary partition tree with its leaves corresponding to n modules.

B. Phase II: Floorplan topology generation

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.

1) Generation of Module Shapes:

Definition 3: A list D = { $(w_1, h_1), (w_2, h_2) \cdots (w_t, h_t)$ } 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 [8].

By making individual w_i and h_i distinct as in Def. 3, a shape with smaller height is chosen from two implementations with same width. Thus an inferior shape is always excluded from D. A set of possible irredundant rectangular shapes for m_i is generated by factorizing T_{m_i}. As we are considering only rectangular shapes, there may not be many choices such that $width * height = T_{m_i}$. A few more shapes, i.e., (width,height) pairs for a module are generated by factorizing all composite integers from T_{m_i} to $T_{max}(m_i)$, where $T_{max}(m_i)$ is the smallest square integer greater than or equal to T_{m_i}.

Let α be a positive integer denoting the maximum aspect ratio defined for any module. A set of (w_i,h_i) pair 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, $j = 1, 2, \cdots n$, we generate a set of t_j possible irredundant shapes D_j = {(w_1, h_1), (w_2, h_2), $\cdots$, (w_i, h_i), $\cdots$ (w_{t_j}, h_{t_j})}. The following lemma gives the time required for generating all different shapes.

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

2) Node sizing: A subtree rooted at an internal node p corresponds to a sub-floorplan. The sub-floorplan at p is generated by joining $(w_i, h_i) \in D_l$ and $(w_j, h_j) \in D_r$ vertically or horizontally, where D_l and D_r are i^{th} shape of left child (left sub-floorplan) and j^{th} shape of right child (right sub-floorplan) of p. If p is the parent of leaves, then the left and right sub-floorplans are the shapes of the modules themselves.

Vertical Cut: We use the vertical node sizing algorithm of [11] to generate a sub-floorplan with vertical cut. Let D_l = { (w_{l_1}, h_{l_1}), (w_{l_2}, h_{l_2}), $\cdots$, (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}), $\cdots$, (w_{r_t}, h_{r_t}) }, with $|D_r| = t$, be the set of possible irredundant shapes of the left subfloorplan (module) and the right subfloorplan (module) respectively, of a node in the partition tree. D_l is sorted such that,

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

D_r is also sorted as above. If (w_{l_i}, h_{l_i}) and (w_{r_j}, h_{r_j}) are merged vertically, the resultant floorplan size becomes (w_{v_k},h_{v_k}) = $(w_{l_i} + w_{r_j}, max(h_{l_i}, h_{r_j}))$. The number of resultant irredundant shapes is at most $s + t - 1$ [8].

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

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

Merging $(w_{l_i}, h_{l_i}) \in D_l$ 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 case of vertical cut, the number of resultant irredundant shapes is atmost $s + t - 1$.

3) Generation of Slicing Trees: For each internal node p of the partition tree, a vertical list V and a horizontal list H are constructed from the child sub floorplans using the algorithms described in Section III-B.2. A combined list M of irredundant shapes is constructed at p by merging V and H such that,

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

is satisfied. Here, k is the number of irredundant shapes at each internal node.

Lemma 2: If s and t are the cardinalities of the shape-list D_l and D_r of left and right subfloorplans of a node u respectively, then the number of shapes at node u is atmost $2(s + t - 1)$.

Proof: As we are merging the vertical and horizontal lists according to the condition mentioned above, the size might at most grow by a factor of 2 than in [8]. ■

The combined list M_l and M_r created at left and right child of the node p is used for sub-floorplan generation at its parent node p. Thus, the nodes of the tree are processed post-order generating a set of subfloorplans at every internal node p. We store the subfloorplan at p as a 5-tuple vector $(w_i, h_i, cut_i, l, r_r)$, where (w_i, h_i) is the i^{th} shape of node p which is generated by merging $(l_l)^{th}$ shape of left child l and $(r_r)^{th}$ shape of right child r using cut_i. cut_i is either vertical or horizontal.

Lemma 3: By horizontal or vertical node sizing atmost $nd - (\log_2 n + 1)$, i.e., $O(nd)$ shapes/slicing trees can be generated at the root of the tree where n is the number of modules and d the maximum number of shapes for a module.

Proof: At any node at level i ($i = 1, \ldots \log n$) of the slicing tree, the size of the list is $2^i d - 2i + 1$. With $i = \log_2 n$ at the root, the number of shapes is $O(dn)$. ■

During the postorder processing of nodes, we also calculate the resource requirement R_p = (p_{clb}, p_{ram}, p_{mul}) at every node p by summing up the resources R_l and R_r required by its left and right child respectively. The requirement vector is used for realization of the slicing tree in Phase III of our method.

Thus, at the root we get a set of floorplan shapes F = {(T_{w_i},T_{h_i})} where T_{w_i} and T_{h_i} are respectively the width and the height of the floorplan in terms of tiles. Each shape of F corresponds to a distinct slicing tree/ floorplan. Further, F is in increasing order of width and decreasing order of height by our method of construction. Thus, an appropriate floorplan

shape could be chosen from this list according to the given aspect ratio and/or floorplan area requirement.

Lemma 4: The time taken to generate the $O(dn)$ slicing trees is atmost $O(dn)$.

Proof: The number of slicing trees generated is $O(dn)$ (see Lemma 3). A slicing tree of depth i is produced in $(2^i d - 2i + 1)$ time. Therefore, the total time is $\sum_{i=1}^{\log_2 n}(2^i d - 2i + 1)$ $= O(dn)$. ■

Thus, to sum up, we process the tree once bottom-up generating merged lists discussed in Section III-B.3. We do not decide the cut line in this step. In the next phase (discussed below), starting from root, we proceed top-down deciding the cut line (horizontal or vertical), thus reaching the leaves.

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

For every slicing tree generated in the previous step, now we assign coordinate position to each module. This consists of two steps: Allocation of a rectangular region which satisfies the CLB requirements followed by allocation of RAM and MUL within and outside this region without any discontinuity.

1) Allocation of rectangular region to a module: Each slicing tree is traversed level-order 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. Suppose the region contains r_{clb} rows and c_{clb} columns of CLBs. If the CLB requirements at node p, its left child l and its right child r are p_{clb}, l_{clb} and r_{clb} respectively, p_{col} the number of CLB columns at p, then if p represents a vertical cut, the number of CLB columns allocated to l is

$$l_{col} = \frac{l_{clb}}{p_{clb}}.p_{col} \tag{2}$$

So $(p_{col} - l_{col})$ columns are allocated to its right child r. The number of rows required to satisfy l_{clb} and r_{clb} at l and r is simply $\frac{l_{clb}}{l_{col}}$ and $\frac{r_{clb}}{r_{col}}$ respectively. For a horizontal cut at p with p_{row} rows of CLB, the number of CLB rows allocated to l is

$$l_{row} = \frac{l_{clb}}{p_{clb}}.p_{row} \tag{3}$$

The right child node r is allocated $(p_{row} - l_{row})$ columns. The number of columns required to satisfy l_{clb} and r_{clb} at l and r is $\frac{l_{clb}}{l_{row}}$ and $\frac{r_{clb}}{r_{row}}$. The number of columns and rows required (width and height) to implement a sub-floorplan corresponding to each node of the slicing tree, are computed using this strategy.

The position of each region corresponding to a node is assigned as follows. As the root node of the slicing tree corresponding to the entire floorplan is allocated to $(0, 0, W, H)$, the left child l always inherits its top-left corner (x^l_{min}, y^l_{min}) from its parent p and the bottom-right corner (x^l_{max}, y^l_{max}) is derived from the width and height calculation described in the previous paragraph. So, for a vertical cut at parent p,

$$x^l_{max} = x^l_{min} + l_{col}; \qquad y^l_{max} = y^p_{max} \tag{4}$$

and for a horizontal cut at p,

$$x^l_{max} = x^p_{max}; \qquad y^l_{max} = y^p_{min} + l_{row} \tag{5}$$

The top-left corner (x^r_{min}, y^r_{min}) of the right child r is calculated as follows: if the cut at p is vertical,

$$x^r_{min} = x^l_{max} + 1; \qquad y^r_{min} = y^l_{min} \tag{6}$$

and if the cut at p is horizontal, then

$$x^r_{min} = x^l_{min}; \qquad y^r_{min} = y^l_{max} + 1 \tag{7}$$

(x^r_{max}, y^r_{max}) of node r is calculated analogous to Equations 4 and 5 for vertical and horizontal cuts respectively. Thus, each leaf of the tree corresponding to a module has a rectangle assigned to it and its CLB requirement is satisfied by the CLB locations within the rectangle.

2) Allocation of RAM and MUL: A rectangle assigned to a module m_i may have sufficient CLBs but not RAM/MUL positions required by it. So, RAM/MULs may have to be placed exterior to the (top and bottom) boundary of the rectangle, which falls in a rectangle assigned to a neighbouring module m_j. If the RAM/MUL requirement of m_j 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 weighted bipartite matching *(MWBM)* problem, so that a module is not realized in disconnected regions.

Let $G = \{U = U_1 \cup U_2, Z\}$ $(U_1 \cap U_2 = \phi)$ be a weighted bipartite graph, where U_1 represents the RAM units required by the modules and U_2, the candidate RAM locations. For a module m with RAM requirement m_{ram}, there are m_{ram} vertices in U_1. Suppose the rectangle R = $(x_{min}, y_{min}, x_{max}, y_{max})$ has been assigned to m. Then for each RAM column intersecting R, a RAM strip is said to include the RAM locations within R, along with m_{ram} locations above its top boundary and m_{ram} locations below its bottom boundary. There is a vertex in U_2 for every RAM location in each RAM strip with respect to rectangle R. There is an edge $(u_i, u_j) \in Z$ if u_i corresponds to a RAM unit required by a module m and u_j corresponds to a candidate RAM location with respect to R assigned to m. In order to enforce connectedness of a module, the weight of edge (u_i, u_j) is chosen as the vertical distance from the center of rectangle R to that of the RAM location for u_j.

Fig. 3 shows the candidate RAM/ MUL locations for allocating the RAM/MUL required by module m_i. Suppose m_i requires 3 RAMs, but the region allocated to m_i has only 2 RAMs. In the figure, the RAM locations within RAM strip 1 and RAM strip 2 are the RAM locations chosen for assigning the 3 RAMs required by m_i. The corresponding bi-partite graph is also shown, where from each of r_1 , r_2, r_3 there are edges to all the RAM locations $1a, \cdots 1g, 2a, \cdots 2g$.

Now, we solve *MWBM* on G to assign unassigned RAMs to available RAM locations. If there is no assignment, there is no feasible solution. MULs are also assigned to the physical locations similarly by solving a separate *MWBM*.

This process of CLB assignment followed by RAM and MUL assignment is carried out for every slicing tree generated in phase II. The half-perimeter wirelength is calculated for

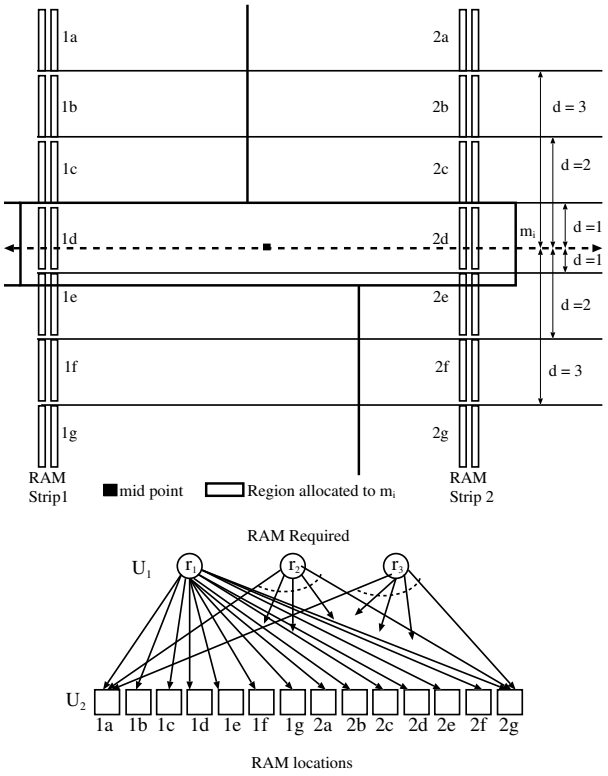

Fig. 3. Candidate RAM/ MUL location for a module

each floorplan generated. The floorplan with no discontinuity and least wirelength is chosen as the final floorplan.

D. Complexity of our approach

Theorem 1: The time complexity of our approach excluding Phase I is $O(n^2 \log n + H^{1.5} n \log n)$, where H is the height of the chip.

Proof: By Lemma 4, the time taken for generating the $O(dn)$ slicing trees is $O(dn)$. For each of the slicing trees, we traverse the tree of size $O(n)$ from root to leaves to fix the rectangular regions of the CLBs in $O(n)$ time. Then, we solve a MWBM to assign RAM/MULs in $O(|Z|\sqrt{|U|})$ time [13]. As the number of RAM/MULs are proportional to within a constant factor of the height H of the chip, the number of vertices U in the bipartite graph is $O(H)$. In the bipartite graph, the edges are assigned to RAM/MUL locations that intersect the x-span of the CLB rectangle. The CLB rectangles being non-overlapping, the number of edges $|Z|$ is again $O(H)$. Thus, MWBM takes $O(H^{1.5})$. The total time complexity is $O(dn(n + H^{1.5}))$. With $d = O(\log n)$ [11], the time complexity result follows. ■

Phase I is iterative because of the use of *hMeTis*. But, the authors of *hMetis* in [12] claims that time taken by *hMeTis* is almost linear in the number of hyperedges, i.e., netlists. Thus, our method, in terms of time complexity compares favourably against that of [6] which takes $O(W^2 n^5 \log n)$ time.

IV. An Example

The method described in Section III is explained with a small example. We considered a circuit from [5] with 20 modules and constructed an appropriate netlist for comparison purpose. Fig. 4 shows the binary partition tree obtained in Phase I of our method. The integers $0 \cdots 19$ written just below the leaves indicate module indices. 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 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 $(0, 64, 21, 83)$. Fig 5 shows the final allocation for each module on Spartan-3 (XC3S5000).

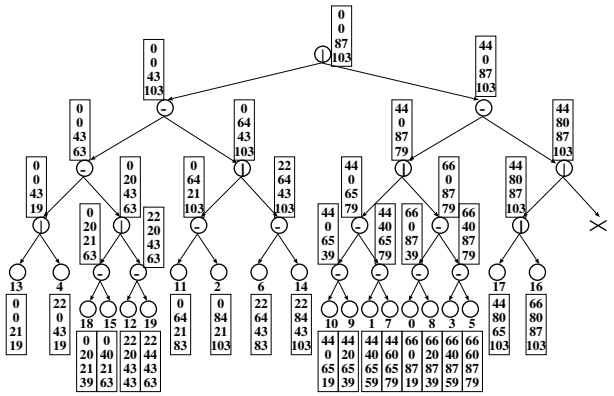

Fig. 4. An example

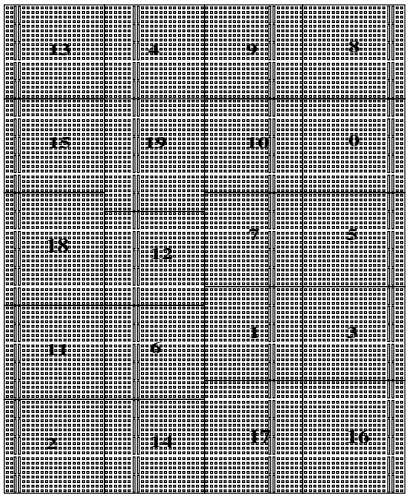

Fig. 5. Floorplan of the example circuit

V. Experimental Results

We have implemented the proposed method in C on 1.2GHz SunBlade 2000 workstation with SunOS Release 5.8. Our method is tested on Xilinx XC3S5000 (Spartan-3) FPGA with

TABLE I

Floorplan results for *20-module example* **[5]**

Idx	1	2	3	4	5
T_w,T_h	$1, 104$	$2, 52$	$3, 39$	$4, 26$	$5, 22$
WL	392	560	728	816	618
Idx	6	7	8	9	10
T_w,T_h	$6, 20$	$7, 17$	$8, 14$	$9, 13$	$10, 11$
WL	946	594	768	971	854

TABLE II

Benchmark Circuits, C: CLB, R:RAM, M:MUL, WL: wirelength

	Ckt Characteristic			Feng-Mehta [7]		Our Method		% WL Improvement	
Circuit	**#Modules**	**#Nets**	**#(C, R, M)**	**WL**	**Time(s)**	**WL**	**Time(s)**	**Case I**	**Case II**
apte	9	44	6614, 70, 70	–	–	213,540	1.22	–	–
xerox	10	183	6625, 66, 50	–	–	536,450	1.02	–	–
hp	11	44	6591, 66, 66	–	–	113,652	0.96	–	–
ami33	33	84	6289, 61, 60	89,283	2.72	51,356	1.39	42	32
ami49	49	377	6300, 63, 63	1,173,000	4.96	1,001,462	3.84	15	52
n100a	100	576	6352, 39, 38	358,338	8.87	132,682	1.16	63	28
n200a	200	1585	6342, 44, 34	700,045	58.24	291,592	2.78	58	31
n300a	300	1893	6625, 65, 54	875,602	177.67	431,855	3.47	51	34

8320 CLBs, 104 RAMs and 104 multipliers. 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 $A = (24 \times 4, 1, 1)$, i.e., 96 CLBs, 1 RAM and 1 MUL. Experimental results on 8 benchmark circuits derived from MCNC [5] and GSRC Bookshelf ASIC floorplanning benchmarks [14], are reported here. ASIC benchmarks are converted to FPGA benchmarks as in [7] by proportional CLB requirements.

The effectiveness of our method is demonstrated with the 20-module example circuit in [5], having 16 modules with 400 CLBs, 5 RAMs, 5 MULs and 4 modules with 480 CLBs, 6 RAMs, 6 MULs to cover the entire target architecture. Table I shows the result obtained by our method 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 WL shows the wirelength obtained for each slicing tree realization after phase III. The time taken to generate floorplans for all the 10 slicing trees is 2.98 seconds, which is far less than 88 seconds taken by [5] on a faster 2.4 GHz Intel(R) Xeon CPU. We observed that, our method could construct the same floorplan reported in [5] with an appropriate net partition tree. In Table I, the topology for the column $idx = 4$, is identical to that reported in [5]. Since [5] does not report the wirelength, we can not compare it with ours.

Table II has the details of the 8 benchmark circuits, namely the number of modules, signal nets, total requirements of the three types of resources, in columns 2, 3 and 4 respectively. The column 5 and 6 report the wirelength and time taken to obtain the floorplan reported in [7]. The next two columns report the same by our method. Note that the time reported in [7] is on a much faster 3.06Ghz Intel Xeon CPU. As mentioned in Section II-B, WL is the sum of the semi-perimeters of all the nets. The wirelength shown in column 7 is computed assuming the net terminals at the centre of the module whereas in [7], the location of the terminals (center/boundary) for computing the wirelength were not explicitly stated. We report the improvement in wirelength over [7] in column 9 (case I) by taking the values directly from their paper. We observed that, on the average, there is 45% improvement in wirelength over 5 circuits. The column 10 shows the improvement over [7] (case II) when we compare an appropriately scaled estimation of our wirelengths if the terminals of a module are along its boundary. There is still 35% improvement in wirelength, on the average.

The time reported in Table II (column 6 and 8) are on two different platforms. For comparison purpose, we have scaled up the time taken by our method using the scaling factors from [15] and observed that our method is about $2\times$ to $50\times$ faster than [7] depending on the size of the circuit. This shows the suitability of our method for fast FPGA floorplanning.

VI. 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. The time complexity of our approach excluding phase I of partitioning, is $O(n^2 \log n + H^{1.5} n \log n)$, where H is the height of the chip and n is the number of modules. Experimental results show a significant speed-up over existing methods. The half-perimeter wirelength of the resultant solution shows 45% improvement over the method reported in [5].

References

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

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

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

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

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

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

[7] Y. Feng and D. Mehta, "Heterogeneous Floorplanning for FPGAs" in *Proc. International Conference on VLSI Design*, Jan. 2006.

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

[9] Parthasarathi Dasgupta, Susmita Sur-Kolay, Bhargab 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.

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

[11] L. J. Stockmeyer, "Optimal Orientations of Cells in Slicing Floorplan Designs", Information and Control, 57(2/3):91-101,1983.

[12] 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, March, 1999.

[13] M. H. Alsuwaiyel, "Algorithms Design Techniques and Analysis", World Scientific, 1999.

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

[15] http://www.spec.org/cpu/results/cint2000.html

0-7695-2762-0/07 $20.00 © 2007 IEEE