Fast Floorplanning for FPGAs*

John M Emmert, Akash Randhar and *Dinesh Bhatia*

Design Automation Laboratory,
ECECS Department
University of Cincinnati,
Cincinnati, OH 45221–0030

Abstract

Floorplanning is a crucial step in the physical design flow for FPGAs. In this paper, we use min-cut based successive bipartitioning to floorplan circuits for application to FPGAs. The primary motivation of this work is reduction of execution time required to accomplish the floorplanning step of device mapping. Our method includes clustering to enhance circuit performance and terminal propagation to reduce total wire length and enhance circuit routability. The floorplanner is intended to take predefined macro based designs as input. Using the Xilinx *xc4000 series of FPGAs as the target architecture, we have demonstrated effective and fast floorplanning on a collection of designs.*

1 Introduction

Field Programmable Gate Arrays (FPGAs) have become very prevalent in every possible design scenario. Since their introduction in the mid 1980s, FPGA device architectures have undergone significant changes. Devices with an ability to map 100,000 gate equivalent designs have become common and new research and commercial development has shown the promise of one million gate equivalent devices in the near future[12]. While technological innovations are easily facilitating higher density devices, not much progress has been made towards CAD tools facilitating the implementation of large designs. Once the device density reaches one million gates and beyond, the design complexity will fall close to what we currently have for custom ASICs. These new high density FPGAs will require new and innovative methods for rapidly mapping circuits to device architectures.

In this paper, we present a min-cut successive bipartitioning based floorplanner for mapping macro based designs to high density FPGAs. Our approach makes use of precharacterized macros [2] that are used during the application design process as well as during the physical mapping process. Figure 1 illustrates the design flow. The subsequent netlist defining the interconnection of macros is used to floorplan the design for an FPGA device.

Breuer [2] suggested a min-cut based floorplanning approach for general VLSI circuits. We adapt this approach and make use of combinational *force directed clustering* along with *Fiduccia Mattheyses* [1] successive bipartitioning to relatively place macros within

* This research is partially supported by contract number F33615-96-C1912 from Wright Laboratory of the US Air Force and a grant from Lucent Technologies

[2] Components in the library include macros like `adders`, `multipliers`, `shifters`, `decoders`, and more. These macros have fixed size, expressed in terms of number of logic blocks. In addition, each block also has a shape with logic blocks optimally preassigned within the predefined shape. Such a preplacement of logic blocks within a shape boundary helps in precharacterizing the performance of the macro. Also, after floorplanning, macros can be forced to retain their shape thus avoiding the need of reassigning logic blocks within macros after floorplan completion.

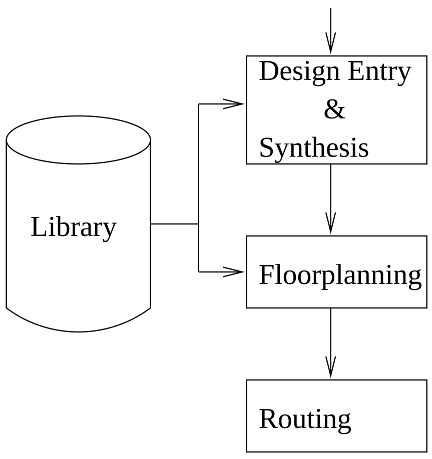

Fig. 1. *The floorplanning based design flow.*

partitions. This is followed by *legal placement* within each partition. During legal placement, flexible macros are reshaped and all macros are assigned a physical location. For flexible macros, any intra macro placement is accomplished using simulated annealing. Finally, compaction is performed on the floorplanned design to minimize the overall area.

The floorplanning problem has been studied extensively in VLSI CAD, but relatively little work has been done in the area of FPGA floorplanning for macro based designs. Shi has proposed a force directed method for macro based FPGA floorplanning [5, 6]. Mathur proposed a combination of net based and path based approaches for timing driven floorplanning [7] . Simulated annealing based approaches have been suggested [9], but they are not efficient in terms of execution time. Various other techniques suggested for general VLSI floorplanning have been adopted for FPGAs [11][2]. In this paper section 2 defines the floorplanning problem and illustrates each step in our floorplanning approach. Section 3 presents benchmarks circuits, test results, and test analysis.

2 Floorplanning

Macros are a collection of relatively placed configurable logic blocks (CLBs). They can be *hard* (of fixed area and shape) or *soft* (of fixed area but flexible shape). The goal of floorplanning is to determine a valid physical location for each macro and the dimensions and internal placement for each flexible macro. More formally stated:
Given : A set of macro blocks $M = \{m_1, m_2, ..., m_n\}$, with area $a_1, a_2, ..., a_n$ respectively.
Objective : Assign a width w_i and height h_i to each flexible block $m_i \in M$ and assign a physical location to each macro $m_i \in M$ such that the following constraints are satisfied:

1. the area of the mapped circuit is $\leq$ the area of the target FPGA,
2. the circuit delay is minimized, and
3. the circuit is 100 % routable.

2.1 Definitions

Partition Segment: A set of modules generated by bipartitioning is called a partition segment. Bipartitioning of a set generates two partition segments.

Segment Number: A unique number assigned to each *partition segment*.

Parent Partition Segment: A partition segment is *parent* to all modules it contains.

Cardinality of Partition Segment: The number of modules in the segment.

Area Slice: An area slice is a block of area on the FPGA generated by *cutlines*. Each *partition segment* is mapped to one and only one area slice, and each area slice has one and only one partition segment mapped to it.

Pseudo Module: A pseudo module is a module of size zero, introduced by the process of *terminal propagation*. A pseudo module is always contained in a *partition segment*.

Coordinates of Partition Segment: The coordinates of a *partition segment* are the coordinates of the area slice, to which it is mapped. They are represented by top left and bottom right coordinates of the area slice.

2.2 Successive Bipartitioning

Successive bipartitioning is the process of dividing the design into multiple segments, such that the cardinality of each segment is less than or equal to a constant, K. We start with the input design as the initial partition segment and assign it to the whole FPGA area. We continue to bipartition the segments, until the terminating condition (cardinality of each partition segment $\leq K$) is satisfied. With every bipartitioning of a segment, the area allocated to that segment is also divided into two area slices. Then, we allocate each new partition segment to an area slice.

Following describes the process of successive bipartitioning. A queue is maintained to keep track of partition segments which are candidates for further bipartitioning. A partition segment is a candidate for further bipartitioning if and only if the cardinality of the segment is $> K$. The queue is initially loaded with the original design. In one pass of successive bipartitioning, the head of the queue is bipartitioned. At the same time, the area of the target FPGA is sliced into two parts by a *vertical cut*, and each area slice is allocated to a segment obtained as a result of the bipartitioning. Out of these two partition segments, eligible candidates for further bipartitioning are loaded into the queue. The process stops when the queue is empty. It should be noted that vertical cuts of the target FPGA are followed by horizontal cuts and horizontal cuts are followed by vertical cuts in an alternating pattern. This process of iterative bipartitioning effectively forms a *partition tree* whose nodes are *H or V* indicating horizontal or vertical cuts respectively, and leaf cells of the tree are the partition segments with $\leq K$ modules. Each of these leaf cells has an area slice assigned to it on the FPGA. A possible order of cuts is shown in Figure 2 (A), and the corresponding partition tree is shown in Figure 2 (B). A two dimensional integer array keeps track of which segment numbers are mapped to which FPGA CLBs. Integer values at the (x, y) indices of this array, correspond to the segment numbers occupying the CLB slots on the FPGA. Since all leaf nodes of the partition tree are mapped to the array, the physical area of the FPGA is allocated by the tree nodes. Each of these nodes contains $\leq K$ macros. Hence, effectively mapping the groups of macros to localities on the FPGA.

We perform connectivity based clustering to form the initial partition for input to the FM bipartitioning algorithm. We extract two clusters out of the initial macro set, M. Each contains macros which are densely connected. Since initial clusters are further operated on by the partitioner to refine the cutset, execution speed takes precedence over quality during clustering. Therefore, a simple greedy method is used to obtain the clusters.

The FM partitioning algorithm, being iterative in nature, is highly dependent on the quality of the initial cut. Hence a good initial cut produced by clustering, remarkably improves the performance of the FM bipartitioner. Table 1 demonstrates cutset improvement for clustered input relative to random input for the FM bipartitioner.

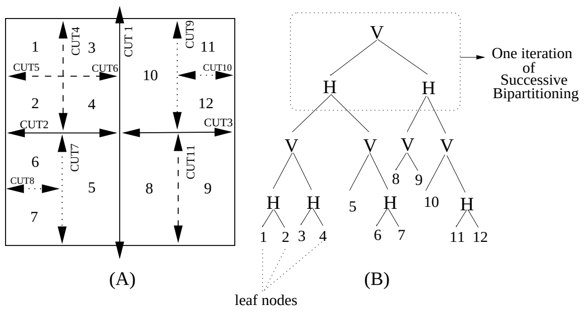

Fig. 2. *(A)-A possible order of cuts. (B)-Corresponding partition tree.*

Design	Cutset by FM Partitioner	
	Random	Clustered
CKT1	62	40
Mult16	79	45
MultRace	57	50
CLA	148	82
CPU	58	73

Table 1. *Cutset after first cut on input netlist.*

Terminal Propagation: The purpose of terminal propagation is to provide knowledge of inter-partition connections to modules in each partition [3]. This aids in reducing total interconnection length of the placed design. We begin with preprocessing of existing pseudo terminals. The pseudo terminals, which were unlocked during *clustering*, belong to one of the sub partitions generated by the *FM partitioner*. They are initially unlocked. During preprocessing, we first lock such pseudo modules. If a pseudo module belongs to a partition segment $p_i \in P$ and its previous lock position was (x, y), then given a horizontal cutline its new lock position is $(x, \frac{BRY(p_i)+TLY(p_i)}{2})$. ($BRY(p_i)$ and $TLY(p_i)$ return the bottom-right and top-left coordinates of the partition segment $p_i \in P$ respectively.) Similarly, given a vertical cutline its new lock position is $(\frac{BRX(p_i)+TLX(p_i)}{2}, y)$. After allocating the new lock positions to all unlocked pseudo modules, we lock the corresponding pseudo modules in every module *pair*, such that the relative lock position of the *pair* of modules is maintained.

After this preprocessing, we introduce a new pseudo module for each cut-net in both of the subpartitions. These new pseudo modules are located on the center of the cut-line. Hence, a new pair of pseudo modules belonging to two different partitions but locked to adjacent positions are introduced. Hereafter, when any pseudo module of this pair is moved to another location, the other pseudo module is also moved to maintain the same relative position. In this manner the pseudo modules form an intermediate connection that draws modules connected to these pseudo modules closer together (see figure 3).

In figure 3, the modules m_i and $m_j \in M$ are cut by the first vertical cut V_1. At this point, two pseudo modules (shown as circles) are introduced and locked to the the center of the cutline V_1. After the horizontal cut H_1, the module m_i is placed in the top partition. Because of this movement, the pseudo module in the partition of m_i moves to the center position on the previous vertical cut-line in the new partition of m_i. This causes the corresponding pseudo module (m_j) of the pair to be attracted to the top partition during partition H_2. In the absence of any bias from the pseudo module, the module m_j could go in the top or bottom partition made by horizontal cut H_2 But as a result of the pseudo modules, m_j has a bias to go to the top partition. This will reduce the cutset by one and m_j will remain close to m_i. For the same reason, the two modules will tend to remain in nearby partitions during subsequent cuts.

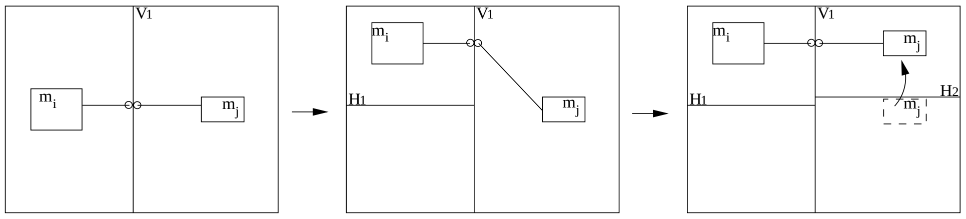

Fig. 3. *Terminal propagation : pseudo modules.*

2.3 Legal Placement

At the end of successive bipartitioning, each leaf node of the partitioning tree contains a maximum of K macros. Also, each leaf node is mapped on the target FPGA. In effect, it gives a locality on the FPGA chip corresponding to a leaf node, within which, macros contained in the leaf node should be placed. Legal placement is performed individually on each leaf node to decide the exact location and shape of macros contained in them. First, we decide relative placement of macros inside each partition. Then, we place hard macros. Finally, we process soft macros and place them. Processing of soft macros includes reshaping and deciding CLB placement inside the reshaped macro. Legal placement is also largely responsible for highly compact floorplans. This is achieved during reshaping and placement of soft macros. We place the macros maintaining actual rectilinear boundaries of the modules under consideration, figure 4.

Relative placement of Macro Blocks: To obtain relative placement of macros inside a partition, we perform an exhaustive search for the best relative placement (least total wire length). Since there are a maximum of K modules in each partition, there are $K!$ combinations to be explored in this search. Experimentally we found $K = 3$ to be a manageable value. K values beyond 3 require other methods for placement [4].

During macro block placement pseudo modules, introduced during *terminal propagation*, are still present at the boundaries of the partition. These give the direction from which each net enters or exits the partition. While deciding the relative placement of modules inside a partition, we account for wire length for connections among various macros as well as the pseudo modules in the partition. This takes care of both inter-partition and intra-partition connections. Hence an attempt to limit global minimum wire length is made.

To identify the best arrangement out of all possible permutations of macros inside a partition, we first compute the number of connections between macros within the boundary of the partition. Next, we calculate the number of connections between each module in the partition and the left boundary of the partition. Then the right boundary. Finally we exhaustively place the modules in a horizontal direction based on the number of connections.

Hard Macros: We performed *successive bipartitioning* and *relative placement* while considering module size only. No attention was paid to module dimensions. This may result in partitions that despite having enough area, are not wide enough or tall enough to accommodate a fixed $m_i \in M$. In case of such a discrepancy, we place the macro in free space of adequate dimensions nearest to the location allocated to it. Suppose the decided module location is (x, y). If this location can not accommodate the hard macro $m_i \in M$, then m_i is placed at a location $(x + \delta x, y + \delta y)$, where $|\delta x|$ and $|\delta y|$ are minimum, and location $(x + \delta x, y + \delta y)$ can accommodate the hard macro.

Soft Macros: We reshape soft macros to fit in the dimensions of the space allocated to their *leaf cell* in the partition tree. This is done by sequentially allocating available CLB slots on the FPGA to macros in the partition segment. This allocation is done in a snake like fashion (even columns start from the top row, odd from the bottom). The method of CLB assignment is such that we first fill the column of available space beginning from the first row of the column. After a column is filled the next successive column is filled starting from the last row up to the first row. The remaining CLBs of the macro start occupying the available CLB slots in the next column. This process is continued until all CLBs inside the macro are placed. If a high degree of compactness is desired, the next soft macro starts where the previous one ends. Hence the shape of the soft macros can be nonrectangular, which helps in placing the macros in the minimum possible space (figure 4). But if less compactness is desired, the next macro starts from the top row of the next column. In this case the shape of macros is always rectangular.

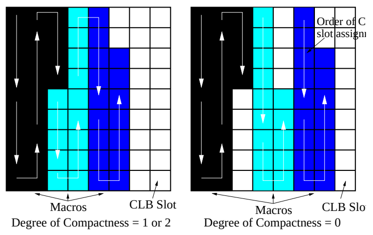

Fig. 4. *Legal Placement: Example nonrectangular shaped of macros.*

Simulated Annealing for Intra Macro Placement: CLBs inside each soft macro are placed using simulated annealing [10]. The constraints for placement are minimum total wire length and minimum longest wire. In this step both intra module connections and inter module connections are considered. Hence a global picture is in view while attempting wire length minimization.

2.4 Compaction

After legal placement, we have a valid floorplan. This floorplan has tightly placed modules inside each leaf cell of the partition tree, but the floorplan may have some unused CLBs surrounding the placed area inside leaf cells. This is illustrated in figure 5(A). During compaction, we work on the floorplan generated by the legal placement step and strive to eliminate such unused space from within the bounding box of the floorplanned layout. We are constrained by the fact that we cannot disturb the relative placement of the macros. We perform compaction by eliminating unutilized rows and columns of CLBs from the floorplan if any exist inside the boundary of the placed design. Partially empty rows and/or columns are left as they are. This ensures that relative placement obtained so far is respected. This step is carried out only if a high degree of compactness is desired. Figure 5 illustrates an example floorplan before and after compaction. After any compaction, we have our final floorplan.

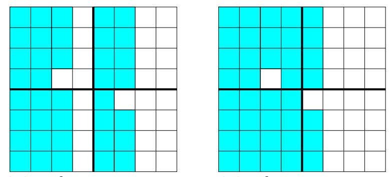

Fig. 5. *Compaction example for high degree of compactness.*

3 Results and Analysis

All results reported here were obtained on a SUN-ULTRA 2300, running solaris 2.5 (with the exception of the force directed floorplan of the CLA design, which was obtained on a SUN-SPARC 5). Table 2 describes various benchmark circuits used to test the floorplanner [8]. In this table, column 1 defines the benchmark circuit names. Columns 2 and 3 (labeled #CLB and #Macro) define the number of CLBs and number of macro blocks used for each of the bench mark circuits. Columns 4 and 5 (labeled Part and #CLB) define the target FPGA part and the total number of CLBs available on the target FPGA part. Column 6 gives the percentage of CLB utilization by the benchmark circuit on the target FPGA part $(\frac{column2}{column5})$.

Design	#CLB	#Macro	Part	#CLB	%Util
Ckt1	180	9	4005	196	91.8%
Ckt2	200	10	4006	256	78.1%
Mult16	576	16	4020	784	73.46%
Mult16	576	16	4025	1024	56.25%
MultRace	618	23	4020	784	78.82%
MultRace	618	23	4025	1024	60.35%
CLA	607	128	4020	784	77.42%
CLA	607	128	4025	1024	59.27%
CPU	674	168	4020	784	85.87%
CPU	674	168	4025	1024	65.82%

Table 2. *Benchmarks circuits used for testing.*

Table 3 gives the CPU execution time required by our floorplanner [8], the Xilinx XACT PPR floorplanner, and the force directed floorplanner [5] to floorplan the various benchmark circuits. In this table columns 3 and 4 (labeled Macro) give the execution times for the Xilinx PPR tool using macro based input circuits. In this table and subsequent tables, column 3 provides data for the default placement effort (=2) and column 4 provides data for the maximum placement effort (=5). Columns 5 and 6 (labeled Flat) give the CPU time for executing the Xilinx PPR tool on the flattened input circuits (circuits are flattened and macro hierarchy is removed). In this table and subsequent tables, column 5 provides data for the lower, default placement effort and column 6 provides data for the highest placement effort. Column 7 (labeled FD) gives the execution time for the force directed floorplanner [5], and Column 8 (labeled This Work) gives the execution time for our floorplanner.

The data in table 3 indicates our floorplanner exhibits a fast execution time relative to the other methods tested. This fast execution time by our floorplanner is possible because of the linear nature of the FM bipartitioning algorithm. It was shown that the FM bipartitioner can bipartition a hypergraph with n terminals in $O(n)$ time [1]. This will enable our floorplanner to handle very large circuits in an extremely short amount of time. Table 3 shows the time required by Xilinx increases monotonically as the size of the input design increases. On the other hand, our floorplanner is substantially faster when number of macros is larger. Hence when the average size of the macros is small compared to the design size our floorplanner is substantially faster. But when the design has fewer macros our floorplanner does not greatly outperform Xilinx PPR The force directed floorplanner takes the longest time to floorplan the larger designs. The most time consuming process in the force directed floorplanner is reshaping of the macro blocks. For circuits where the percentage of utilization of the target FPGA is low (reshaping is not performed) the force directed floorplanner is extremely fast. Our floorplanner is faster for circuits with a higher percentage utilization of the target FPGA.

Design	Part	CPU Time (sec)					
		XACT				FD	This Work
		Macro		Flat			
CKT1	4005	NA	NA	42	153	NA	8
CKT2	4006	17	27	45	217	11	12
Mult16	4020	123	174	195	1408	14	81
Mult16	4025	127	175	197	1483	14	81
MultRace	4020	NA	NA	308	1428	NA	90
MultRace	4025	143	170	302	1460	NA	90
CLA	4020	172	294	297	1458	NA	32
CLA	4025	158	273	291	1400	5hr	32
CPU	4020	247	563	412	2937	NA	36
CPU	4025	210	385	418	3035	NA	36

Table 3. *Execution times for various algorithms.*

Table 4 describes performance characteristics of the mapped circuits. It gives the maximum operating frequency for each of the benchmark circuits floorplanned by the various tools [8]. Maximum frequency (F) was calculated from the worst case delay (D) reported by the Xdelay tool from the Xilinx tool set ($F = \frac{1}{D}$). After importation into the Xilinx tool, the floorplanned designs were routed using the Xilinx PPR tool, and the routed designs were analyzed using the Xdelay timing analysis tool.

Columns 3 and 4 give the maximum operating frequency for the macro based benchmark circuits floorplanned by the Xilinx PPR tools. Columns 5 and 6 give the maximum operating frequency for the flattened benchmark circuits mapped by the Xilinx PPR tools. Column 7 gives the operating frequency for the benchmark circuits floorplanned by the force directed tool. Column 8 (labeled H) gives the operating frequency for the benchmark circuits floorplanned by our floorplanner with a high degree of compactness, and column 9 (labeled L) gives the operating frequency for the benchmark circuits floorplanned by our floorplanner with a low degree of compactness.

Performance is a critical metric for floorplanned circuits. A fast floorplanner that results in circuit maps with very low maximum operating frequency is not acceptable. Table 4 shows the operating frequency of our floorplanned designs have performance characteristics similar to those of the commercial tools (except mult16 on xc4025). The table also shows performance characteristics for our floorplanned designs are at least as good of those that were successfully floorplanned by the force directed floorplanner.

Design	Part	Frequency (MHz)						
		XACT				FD	This Work	
		Macro		Flat			H	L
CKT1	4005	NA	NA	5.8	6.0	NA	5.7	NA
CKT2	4006	4.5	4.7	5.1	5.7	4.8	4.9	4.7
M16	4020	13.5	13.7	13.7	13.7	13.6	13.8	13.4
M16	4025	13.6	13.8	13.4	12.9	13.2	13.4	13.3
MultR	4020	NA	NA	10.6	10.0	NA	10.7	10.3
MultR	4025	9.6	9.9	9.8	9.7	NA	10.1	9.5
CLA	4020	8.5	8.8	8.4	8.6	NA	8.54	8.4
CLA	4025	7.5	7.5	8.1	8.9	6.7	8.1	7.9
CPU	4020	6.6	6.7	7.8	7.8	NA	7.1	6.9
CPU	4025	5.5	6.9	7.1	7.9	NA	6.9	6.9

Table 4. *Maximum operating frequency for floorplanned benchmark circuits.*

Table 5 shows the bounding box of the placed designs for various runs of the benchmark circuits [8]. In this case the bounding box is described by the number of CLBs required to accommodate the floorplanned circuit. Obviously the flat designs require the fewest total CLBS since they are the most compact. Since inevitably macro based floorplans of circuits with fixed shaped macros cannot be 100% area efficient, the macro based floorplans require more CLB area and hence a larger bounding box area than flat designs. This is part of the cost of using a fast executing, macro based floorplanner. When high area utilization of the FPGA is required this becomes a factor and a tradeoff between execution time and area utilization may be required. Our floorplanner addresses this issue by reshaping the soft macros to reduce the overall area required by the mapped circuit. The effective reshaping and packing of macros by our floorplanner allowed it achieved feasible placement for all of the test runs.

Design	Part	Bounding Box						
		XACT				FD	This Work	
		Macro		Flat			H	L
CKT1	4005	NA	NA	182	182	NA	196	NA
CKT2	4006	240	240	200	200	240	210	240
M16	4020	675	675	506	506	576	625	672
M16	4025	702	702	506	506	576	625	672
MultR	4020	NA	NA	644	644	NA	729	784
MultR	4025	992	992	1024	1024	NA	729	864
CLA	4020	628	628	584	584	NA	625	784
CLA	4025	650	650	676	676	780	625	1024
CPU	4020	756	756	616	616	NA	702	784
CPU	4025	736	736	640	640	NA	702	870

Table 5. *Smallest bounding box area for floorplanned benchmark circuits.*

4 Conclusions

In this paper we have described the implementation of a large scale macro based floorplanner that exhibits fast execution when compared to industry standard Xilinx tools. Due to predesigned macros, the floorplanner need not address the problem of CLB level placement for all of the macros. Only soft macros, whose shapes are changed during the floorplanning process must be placed. In the majority of cases, the overall approach resulted in mapped circuits whose performance was similar to that of the circuits produced by the Xilinx tools.

The successive bipartitioning method is ideal for initial floorplanning of very large circuits. It quickly divides the circuit into sections that can be assigned to various areas on the FPGA. With the addition of clustering to improve the initial cutsets and terminal propagation to limit the total wire length the quality of the floorplan is greatly improved. In the future we expect that library based design approaches will become fairly common right from synthesis to physical mapping. Thus, floorplanning will play a significant role in both area estimation during synthesis and final mapping during late stages of the design. In our future work, we will integrate this floorplanning methodology with performance driven algorithms to enhance the performance of mapped designs [4]. We will use the successive bipartitioning method with clustering and terminal propagation in the early stages of floorplanning for very large designs. In the latter stages we incorporate performance enhancing methods to aid in the final assignment and placement of the macro blocks.

References

1. C. Fiduccia and R. Mattheyses, "A Linear time Heuristic for Improving Network Partitions", Proc. of DAC, pp.175-181, June 1982.
2. M. Breuer, "A class of min-cut placement algorithms", Proc. of DAC, pp. 284-290, 1980.
3. A. E. Dunlop and B. W. Kernighan, "A Procedure for Placement of Standard-Cell VLSI Circuits", IEEE Transactions on Computer-Aided Design, pp. 92-98, January 1985.
4. J. M. Emmert and D. K. Bhatia, "Fast Placement Using TABU Search for Total Wire Length Minimization", University of Cincinnati, ECECS Technical Report, 1998.
5. Jianzhong Shi, Akash Randhar and Dinesh Bhatia "Macro block based FPGA Floorplanning" Proc. of Intl. Conf. on VLSI Design, January 1997.
6. J. Shi and D. Bhatia, "Performance Driven Floorplanning for FPGA Based Designs" Proc. of ACM Symposium on Field Programmable Gate Arrays, February 1997.
7. A. Mathur and C.L. Liu, "Compression-Relaxation:A New Approach to Timing Driven Placement for Regular Architectures" IEEE Transactions on CAD of Integrated Circuits and Systems, pp. 597-608, June 1997.
8. A. Randhar, "Macro Based Floorplanning for FPGAs" Thesis: University of Cincinnati, December 1997.
9. C. Sechen "Chip Planning, Placement, and Global Routing of Macro/Custom Cell integrated Circuits Using Simulated Annealing" in Proc. of DAC, pp. 73-80, June 1988.
10. A. Subramaniam and D. Bhatia "Timing Driven Placement for Logic Cell Arrays" University of Cincinnati, ECECS Technical Report, 1994.
11. D.F. Wong and C.L. Liu "A new method for floorplan design" Proc. of DAC, pp. 101-107, 1986.
12. `www.xilinx.com`.

This article was processed using the LaTeX macro package with LLNCS style