FRONTIER: A FAST PLACEMENT SYSTEM FOR FPGAS

Russell Tessier

Department of Electrical and Computer Engineering
University of Massachusetts, Amherst, Ma. 01003.

tessier@ecs.umass.edu

Abstract

In this paper we describe Frontier, an FPGA placement system that uses design macro-blocks in conjuction with a series of placement algorithms to achieve highly-routable and high-performance layouts quickly. In the first stage of design placement, a macro-based floorplanner is used to quickly identify an initial layout based on inter-macro connectivity. Next, an FPGA routability metric, previously described in [14], is used to evaluate the quality of the initial placement. Finally, if the floorplan is determined to be unroutable, a feedback-driven placement perturbation step is employed to achieve a lower cost placement. For a collection of large reconfigurable computing benchmark circuits our placement system exhibits a 4× speedup in combined place *and* route time versus commercial FPGA CAD software with improved design performance for most designs. It is shown that floorplanning, routability evaluation, and back-end optimization are all necessary to achieve efficient placement solutions.

Keywords: FPGA, placement, floorplan

1 INTRODUCTION

Over the past decade field-programmable gate arrays (FPGAs) have revolutionized the way digital systems are designed and built. With architectures capable of holding millions of logic gates on the horizon and planned integration of reconfigurable logic into system-on-a-chip platforms, the versatility of programmable devices is expected to increase dramatically.

When programmable logic first became available a decade ago the task of converting a high-level design into a high-performance physical implementation was frequently a time-consuming, manually-driven process requiring many days or weeks. While sizable development times are still tolerable for some applications of FPGA devices today, many uses of FPGA technology, such as reconfigurable computing and ASIC prototyping, require compilation times on the order of minutes to allow for rapid design turnaround from high-level design to physical implementation. Currently, a majority of FPGA compilation time is spent in device layout due primarily to the assumption that each collection of new design elements must be placed and routed from scratch. Given the exponential growth of FPGA logic capacity expected in the next few years, place and route times using algorithms currently employed in FPGA software systems can only be expected to get worse.

While early FPGA designers used low-level schematics to create new designs, most FPGA implementations today start as RTL or procedural algorithm descriptions. Typically these high-level designs are synthesized to circuit structures with the aid of pre-compiled macro-blocks that have predictable area and timing characteristics. In general, these elements, such as adders, multipliers, and multiplexers, are much larger than the primitive logical elements of the FPGA device and are used in multiple locations in

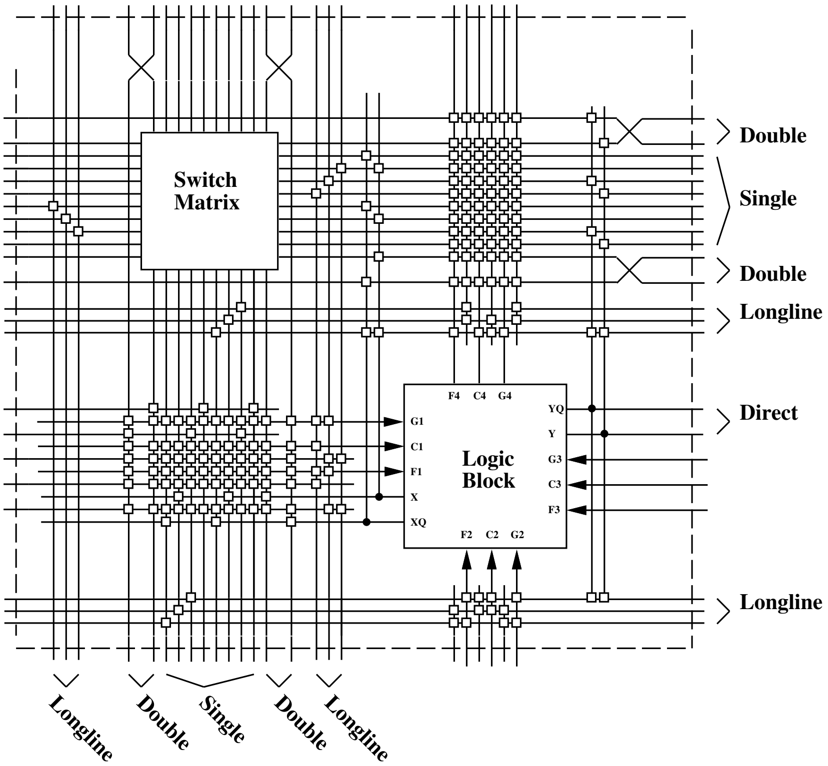

Figure 1 Xilinx XC4000 Logic and Routing Cell

a given design. While macro-blocks have been leveraged successfully for FPGA synthesis for some time, little work has been done in integrating macro techniques into automated FPGA layout.

In this paper Frontier, an integrated placement system that aggressively uses macro-blocks and floorplanning to quickly converge to a high-quality placement solution, is detailed. This system can be used in place of existing placement approaches for macro-based designs targetted to devices with architectures similar to the Xilinx XC4000 [2] and Lucent Orca [1] families. Rather than using a single algorithm, the new Frontier tool set relies on a sequence of interrelated placement steps. First, in a floorplanning step, hard and soft macros are combined together into localized clusters of fixed size and shape and assigned to device regions to minimize placement cost. Following initial floorplanning, a routability evaluator, based on wire length, is used to determine if subsequent routing for a given target device is likely to complete successfully. If this evalution is pessimistic, low-temperature simulated annealing is performed on the contents of all soft macros in the design to allow for additional placement cost reduction and enhanced design routability.

The organization of this paper is as follows. In Section 2 a description of the issues involved in FPGA placement are presented. Section 3 describes previous related work in FPGA placement and floorplanning. In Section 4 our placement system is discussed in detail. Experimental results obtained by applying Frontier to a collection of benchmark circuits is presented in Section 5. Finally, Section 6 summarizes our research and outlines directions for future work.

2 PROBLEM STATEMENT

The target FPGA architecture used for this research is the *island-style* architecture commonly found in commercial FPGA devices such as the Xilinx XC4000 family [2] and the Lucent Orca family [1]. These architectures are characterized by a regular two-dimensional array of logic and routing *cells*, such as the example from the Xilinx XC4000 family shown in Figure 1. Each identical cell contains a logic block consisting of a small number of programmable lookup tables and flip flops and associated routing wires of differing segmentation lengths. Connections between logic blocks and routing resources are made through programmable switches represented as small squares in the figure.

An FPGA design under placement consideration consists of N_{blocks} logic blocks grouped into M instantiated macro-blocks. Each macro-block contains an RTL component such as a datapath function or finite state machine and has a distinct logic block capacity N_{M_i}. Hard macro-blocks are assigned fixed height h_i and width w_i while soft macro-blocks have flexible shape.

The goal of our placement approach is to create a placement for N_{blocks} design logic blocks encompassed by a set of M macro-blocks onto N_{cells} array logic blocks such that subsequent routing may complete successfully. A set of $Nets_M$ inter-macro wires interconnect all macro-blocks and $Nets_{blocks}$ wires interconnect all logic blocks inclusive of $Nets_M$. In general, placement progresses subject to the following constraints:

1. Each hard macro-block is assigned a distinct placement rectangle R_i of dimensions h_i and w_i so that no two macros overlap ($R_i \bigcap R_j = \phi$).
2. Placement is performed to maximize overall routability by minimizing overall routability-based placement cost. Initially, floorplanning considers, among other criteria, minimizing the length of all inter-macro nets $Nets_M$. Subsequently, during placement refinement, the length of all design wiring, $Nets_{blocks}$, is considered.

In addition to wire length, several supplemental cost criteria are considered in performing placement. As shown in Figure 1, commercial FPGA devices, like the XC4000 family have *direct* connections between logic blocks that enhance routability and long-lines that span the extent of the entire device. By building hard macros that are constructed to take advantage of these features, additional design routability can be achieved.

The need for placement optimization is directly dependent on the amount of wiring resources available in an FPGA device and the specific design under consideration. In subsequent sections it is shown that while in many cases floorplanning achieves a routable placement quickly, some initial floorplans may be unroutable for a target device due to the nature of a specific design's interconnection and the shape and extent of its macro-blocks. When, through the use of an accurate routability metric, it is determined that routing will not succeed for an initial floorplan, additional placement optimization that operates on soft macros is performed. The quality of placement achieved with the placement system documented here is comparable to that achieved with existing commercial tools, but placement completes on average more than 60× faster.

3 BACKGROUND

3.1 RELATED WORK

Compile time has recently been recognized as an important issue for FPGAs. Most island-style FPGA placement algorithms used in commercial software packages assume that a user design contains little or no hierarchy and can be considered as a collection of logic components whose grain size matches the logic block of the target device. Since the primary measure of routability for array layout is typically wire length, a flattened design provides maximum flexibility in searching the placement space for reduced overall cost.

Most commercial FPGA placement packages use simulated annealing [11] to evaluate a series of logic block swaps based on a predefined cost function. Annealing, started from an initial placement, typically achieves good placement quality at the cost of long execution times that are exponentially bounded by the number of design logic blocks [15]. In [10], recursive clustering was used to identify circuit locality prior to annealing to reduce subsequent annealing execution time. While this approach yielded a placement time speedup of four in obtaining minimized FPGA placement cost, no mechanism for supporting pre-placed macro-blocks was included. In performing hierarchical clustering, substantial placement time is spent recreating locality information previously encompassed by RTL components. Additionally, this approach does not deal with costs related to long-line alignment and near-neighbor logic block direct connects through the use of hard macros.

A large amount of work in macro-based floorplanning has been applied to full and semi-custom VLSI design styles including approaches based on mincut slicing, simulated annealing, and force directed placement, among others [12] [13]. In general, the floorplanning problem for island-style FPGAs is much harder than for non-programmable technologies since for FPGAs the amount of available routing resources is fixed in preassigned channels that run through placed macro-block regions and additional resources cannot be redistributed around macro borders. Several floorplanning efforts for island-style FPGAs have relied on specific user design implementation styles to quickly achieve a highly-routable placement. These systems [5] [9] [8] restrict target circuits to datapaths oriented in a left-to-right linear communication pattern. Design regularity facilitates vertical bitwise abutment of macro-blocks and allows for a rapid traversal of the one-dimensional topological search space. In general, one-dimensional approaches cannot be easily modified for circuits with more irregular communication patterns and larger Rent parameters.

Several floorplanning approaches for island-style FPGAs based on mincut slicing have recently been developed and tested. In [15] it was shown that while slicing floorplanning with hard macros achieves a placement solution quickly, routability and performance may suffer due to increased wire length. In [7], slicing with terminal propagation in conjunction with the reshaping of soft macros, was shown to quickly generate high-utilization placements for Xilinx XC4000 series devices. While a factor of two speedup was achieved for placement versus Xilinx PPR software, no routing execution times were reported so it is impossible to determine overall place and route speedup.

A floorplanning methodology based on hierarchical placement was recently described in [6]. This floorplanner clusters macros together into fixed sized bins and then optimizes bin placement using a two-step tabu search. Several of the large benchmarks targetted by the system showed considerable speedup in placement time but much more than a 100% increase in routing time. In this paper we show that this routing time increase was likely caused by the lack of a globally optimizing placement smoothing step following floorplanning to minimize localized wire length inefficiencies.

3.2 IMPLEMENTATION TRADEOFFS

The use of macro-blocks to accelerate placement for island-style FPGAs requires accommodation of the following two competing placement goals:

- Locality information stored in pre-placed macro-block libraries should be used to avoid the need to reconstruct *local* design structure from scratch and to better take advantage of device features such as near-neighbor direct connection and long-line alignment.
- The placement system should have the flexibility to minimize *global* wire length by swapping individual logic blocks across the entire design. An approach that is insufficiently flexible will lead to high wire length placements that are likely to take additional time to route, eliminating the benefit of placement time speedup.

The placement system described in this paper is the first integrated approach that addresses both of these competing concerns in one package. First, a macro-based floorplanner based on clustering and shaping is used to quickly identify a feasible floorplan that incorporates hard and soft macros and achieves high device utilization. Once initial placement is complete, a routability estimator is applied to determine if the placement is routable. If it is not routable, a low-temperature annealing step is performed on the entire design to smooth out localized wire length maxima while maintaining the basic structure of the floorplan. The diversity and flexibility of this system makes it applicable not only to user designs

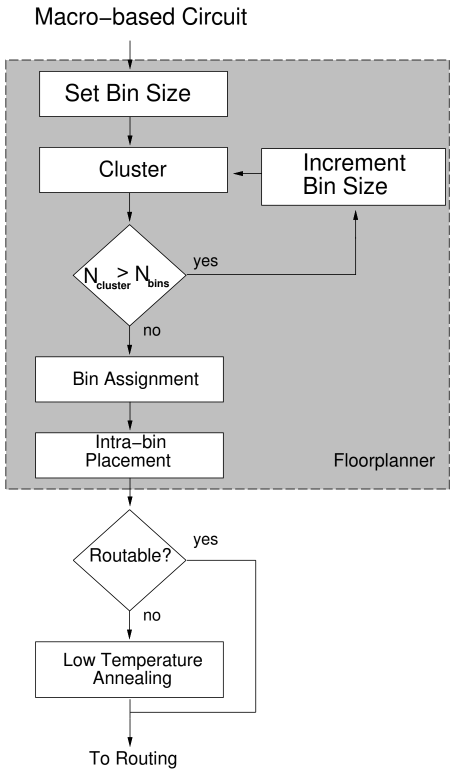

Figure 2 Frontier Placement Flowchart

which communicate as linear arrays and two dimensional meshes, but also to circuits exhibiting irregular communication patterns.

4 FRONTIER IMPLEMENTATION

4.1 SYSTEM OVERVIEW

Our placement system progresses in a series of algorithmic steps by supplementing new layout techniques with recent advances in FPGA routability analysis. As illustrated in Figure 2, the layout process starts with a macro-based netlist of soft and hard macros targetted to an FPGA device containing N_{cells} logic blocks. Initially, to enhance locality, the FPGA device is decomposed into an array of placement *bins*, each of the same physical dimension, as shown in Figure 3. To determine bin contents, macros are grouped together into clusters, each of which will accommodate the volume of macro logic blocks and the physical dimensions of hard macros inside a bin. If following clustering an insufficient number of bins are available to place all clusters, bin sizes are increased and clustering is restarted. After clustering, each cluster is assigned to a physical bin location on the target device and entire bin clusters are subsequently swapped between physical bins to minimize inter-bin placement cost including connectivity to device pins. Since the number of bins allocated to a device is frequently much smaller than the number of device logic blocks, this process proceeds rapidly. The annealing formulation used in inter-bin swapping follows directly from logic block-level annealing used for flattened designs and is easily incorporated into the software flow. Following bin placement, hard and soft macro-blocks are placed within each bin in a space-filling fashion. All intra-bin placement is based on inter and intra-bin connectivity. Soft macros are resized at this point to meet bin shape constraints.

In Section 5, it is shown that while floorplanning alone is sufficient to provide effective placements for many designs targetted to contemporary FPGA devices, in some cases additional placement perturbation is required. In Frontier, following floorplanning, a detailed estimate of the placement wire length is

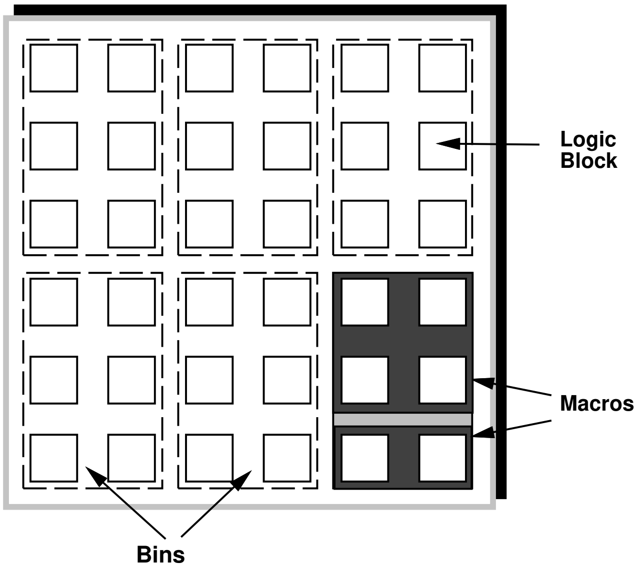

Figure 3 Bin-based Cluster Assignment

determined, taking into account the special features of the FPGA device. As described previously in [14], this wire length estimate can be used to evaluate whether subsequent device routing will complete quickly, require a long period of time, or fail to route at all. For floorplans that are impossible or difficult to route, low-temperature simulated annealing is performed on soft macros to smooth wire length inefficiencies. Through a series of design examples a set of annealing parameters that lead to the best time versus performance tradeoff are determined.

4.2 PLACEMENT STEPS

Bottom-Up Clustering. In the first step of placement, macros are clustered together into placement bins of identical dimensions and CLB volume to identify inter-macro design locality. While bins must be sized to support a range of macro-block dimensions, needlessly large bins limit the number of bins available for subsequent inter-bin swapping and may have a negative impact on final floorplan quality. As previously suggested in [6], when floorplanning is started, bins are initially set to the X and Y dimensions of the largest hard macro-block or, if no hard macros exist, to the square root of the logic block volume of the largest soft macro.

Given the fixed dimension of each bin, clustering must not only take into account connectivity, but also size feasibility of the cluster under consideration. To smooth macro-block size disparities, smaller macro-blocks should be clustered first with other like-sized blocks so that the total number of created clusters is minimized. For Frontier this is accomplished through the use of a cost function described in [16] and [6] that is weighted to take logic block counts and interconnectivity into account:

$$Cost_{ij} = feas(i,j) \times \frac{N_{blocks}}{N_{M_i} + N_{M_j}} \times \frac{min(N_{M_i}, N_{M_j})}{max(N_{M_i}, N_{M_j})} \times \sum Nets_{ij} \tag{1.1}$$

where N_{blocks} is the total number of logic blocks in the circuit, N_{M_i} and N_{M_j} are the number of logic blocks in the macro-blocks M_i and M_j under consideration and $Nets_{ij}$ are the nets connecting M_i and M_j. The first term in Equation 1.1 determines if a candidate cluster can be feasibly shaped during intra-bin placement, using criteria described later in this section, to fit the physical dimensions of a target bin. Its value is set to 1 if a shape is feasible and 0 if it is not. The second term in the cost function prevents a specific cluster from becoming too large in relation to the rest of the circuit. The third term prevents

M: Initial set of design macro-blocks.
C: Set of macros or macro clusters to be combined.
SizeC: Number of elements of C.
Add elements of M to C.
While SizeC can be reduced
    Loop over all $SizeC$ elements.
        Select *feasible* combination C_i, C_j that maximizes $Cost_{ij}$.
    If feasible C_i, C_j found.
        Remove C_i, C_j from C.
        Add macro cluster $C_i \bigcap C_j$ to C.
        Update connectivity.
EndWhile

Figure 4 Weighted Clustering Algorithm

two macros with vastly different numbers of blocks from being connected together thereby creating area inefficiencies, and the last term measures connectivity. A detailed description of the $O(M)$ clustering algorithm appears in Figure 4.

If, following clustering, more clusters C than bins B exist, bin dimensions are modified by increasing bin horizontal and vertical dimensions by 1 logic block and clustering is started again from scratch.

Bin Assignment. Following clustering, all macros are bound to a cluster and the number of clusters is less than or equal to the number of available device bins. The next step is to determine an assignment of clusters to specific device bins. After initial random assignment of clusters to bins, simulated annealing is used to evaluate cluster swaps based on both inter-bin and bin-to-pad wire lengths. The dynamic annealing schedule described in [4] is used to reach a good quality placement quickly. Given the small number of bins (typically less than 20) annealed swapping can typically be completed in a few seconds.

Internal Bin Placement. Once each cluster of macro-blocks is assigned to a specific bin, intra-bin placement is performed to assign macro logic blocks to specific device logic block locations. As a first step for each bin, all N_{hard} hard macro-blocks and N_{soft} soft macro-blocks in the assigned cluster are linearly ordered in the horizontal dimension using a topological sort based on intra and inter-bin connectivity. For soft macro-blocks, previously-determined library placements are used to approximate final soft macro logic block positions and wire lengths.

Following ordering, exact X, Y logic block positions in each bin are determined for *hard* macros by resolving inter-macro spacing. If the width of a bin is w_{bin} and the combined horizontal width of all hard macros in a bin is $\sum_i w_{M_i}$, the space between hard macros occupied by soft macros is determined to be $X_{soft} = \lfloor \frac{w_{bin} - \sum_i w_{M_i}}{N_{soft}} \rfloor$. This equation leads to an X_{soft} value of 1 for the bin shown in Figure 5. Following X_{soft} determination, hard macros are assigned X coordinates inside each bin in a left-to-right order with X_{soft} spacing inserted for each soft macro.

Subsequent to the positioning of hard macros, intra-bin X and Y locations for *soft* macro logic blocks are determined. These locations are determined by allocating bin space remaining after hard macro placement in a snake-like fashion starting in the upper left-hand corner of the bin. Individual soft macro logic blocks are assigned to specific locations within this shape by sequentially selecting logic blocks that minimize overall wire length. By following this methodology, up to 100% logic block utilization can be achieved in each bin.

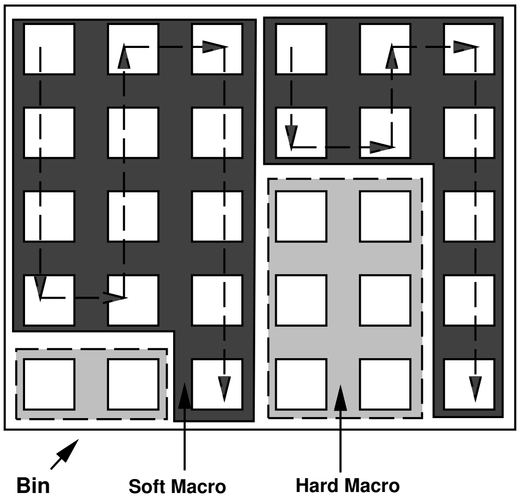

Figure 5 Internal Bin Placement

Routability Prediction. Recently, a direct correlation has been formulated between the number of routing tracks in an FPGA device, the wire length of a design placement, and the amount of time needed to route a design [14]. Due to macro-block shape considerations and specific interconnection patterns of individual designs, a successful floorplanning step provides no guarantee that a placement possessing close to the global minimum cost has been achieved or that routing will subsequently succeed for a given target device. To evaluate placement fitness, a wire length-based routability metric [14] has been directly built into Frontier.

For a given placement, W_{min} may be defined as the minimum track count per FPGA routing channel required to successfully route a design. If the device track count available in a target FPGA, W_{FPGA}, exceeds $1.1 \times W_{min}$ the routing problem is defined to be *low-stress* and can be expected to complete quickly (e.g. within several minutes). If $W_{min} < W_{FPGA} < 1.1W_{min}$ the routing problem is defined as *difficult* and will likely require many minutes to complete. Generally, if $W_{FPGA} < W_{min}$ it is unlikely routing will complete successfully even after substantial routing time.

Swartz [14] noted that since a placement-only estimate of routability is required, it is necessary to use an estimated rather than an exact W_{min} value to determine routability for a design. Through experimentation it was determined that W_{min} can be estimated from placement wire length as:

$$W_{min-est} = \lceil wirelength/(2 \times N_{cells} \times U) \rceil \tag{1.2}$$

where $wirelength$ is the total estimated wire length determined from placement, N_{cells} are the number of logic blocks in the device and U is a utilization factor determined to be architecture-specific. By using this equation for estimation it was possible to determine needed device routing resources following floorplanning for specific Xilinx XC4000XL devices exhibiting W_{FPGA} of 32 tracks per channel and U of 0.6.

Low-Temperature Annealing. As will be shown in Section 5, simply performing floorplanning is generally sufficient to create a placement in the low-stress routing range for many designs. In some cases, however, routability evaluation may reveal that the current placement is difficult or impossible to route given available target device routing resources. For these cases, additional placement perturbation is needed to ensure subsequent fast routing.

To overcome placement inefficiency, Frontier employs low-temperature simulated annealing of individual logic blocks to allow for smoothing of wire length across soft macros and bins without destroying the

Design	Device	CLBs	Macros	Device Utilization
bheap5	4085XL	2715	30	87%
bubble16	4044XL	1280	31	80%
bubble32	4085XL	2608	63	83%
fft16	4085XL	3032	48	97%
spm4	4036XL	1064	11	82%
spm8	4085XL	2425	23	77%
ssp16	4052XL	1491	46	77%
ssp32	4085XL	2370	79	76%

Table 1 Macro-based Design Statistics

Execution times (s)							
	Xilinx PAR-M1.4			Floorplan Only			
Design	Place	Route	Total	Fplan	Route	Total	$W_{min-est}$
bheap5	903	**343**	1246	8	**647**	655	35
bubble16	199	130	329	3	101	104	28
bubble32	1793	294	2087	15	316	331	29
fft16	710	360	1070	14	377	391	22
spm4	143	103	246	8	101	109	26
spm8	530	266	796	4	240	244	26
ssp16	180	81	261	16	72	88	26
ssp32	855	167	1022	17	136	153	22
average	664	218	882	10.6	248	259	27.5

Table 2 Design Layout Statistics - Xilinx 4000XL devices

high-level placement structure achieved by the floorplanner. While detailed discussions of the simulated annealing algorithm for placement [11] and associated controlling parameters [4] are available elsewhere, a brief description of several important parameters needed for floorplan refinement may be summarized as follows:

- Starting annealing temperature, T_{init} - Starting temperature must be set high enough so that the placement may be perturbed to a lower overall minimum, but low enough to avoid destroying the basic hierarchy determined through floorplanning. For our system, a number of experiments were performed to determine T_{init} values that lead to an effective quality versus time tradeoff.
- Inner number, β - This value varies the number of swaps made at each annealing temperature [10]. In the annealing formulation used to perturb logic block placement, the number of moves at each temperature is set to $\beta \times N_{blocks}^{4/3}$ [4]. In Section 5, quality-time tradeoffs for a range of β values are considered and a β value of 1 is shown to exhibit the most favorable quality-time characteristics.

5 RESULTS

The placement system outlined previously was applied to eight macro-based reconfigurable computing benchmarks from the RAW Benchmark Suite [3]. Prior to experimentation, all hard and soft macros were mapped to Xilinx XC4000 logic blocks and resulting XNF netlist files were annotated with *RLOC* placement information. Macro-based netlists in XNF format were then used as input to both the Frontier system and to Xilinx PAR software, version M1.4 [2]. Design statistics for the benchmarks appear in Table 1. All run time results for both Frontier and PAR were obtained using a 140 MHz UltraSparc I

Execution times (s)						Performance
Flow	PAR	Fplan	Low Temp Anneal	Route	Total	(MHz)
PAR-only	903	0	0	343	1246	5.08
Fplan Only	0	8	0	647	655	8.17
Fplan/Anneal	0	8	34	272	314	12.71

Table 3 Layout Execution Time/Performance Comparison - Design bheap5

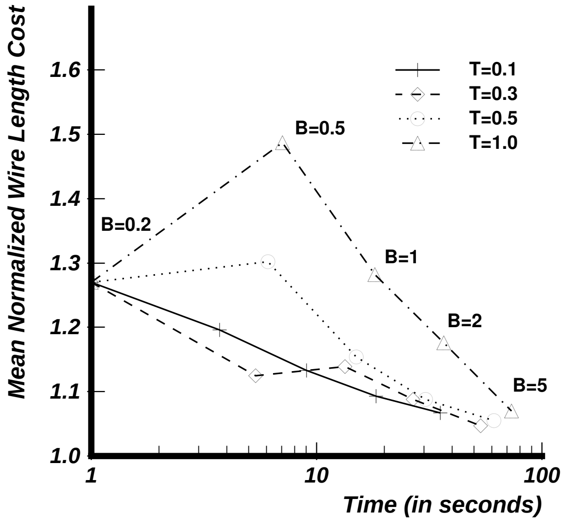

Figure 6 Simulated Annealing Parameter Variation

with 288 Mb of memory. Routing for all designs was performed using Xilinx PAR-M1.4 software with default parameter and routing effort settings.

Execution times for PAR placement, Frontier floorplanning (without low-temperature refinement), and PAR routing appear in Table 2. For all designs except one (*bheap*5) routing times for the floorplanned designs were comparable to those found by the PAR-M1.4 placer that required 60× longer. This is not suprising since for all designs except *bheap*5 the estimated minimum track count per channel needed to route the circuit was less than the 32 tracks per channel available in Xilinx XC4000XL devices. $W_{min-est}$ values were determined by measuring the post-floorplan wire lengths of designs and then directly correlating them to required track counts via Equation 1.2. From Table 2 it can be seen that the minimum track count needed to route the floorplanned version of *bheap*5 is significantly greater than the track count available inside the XC4000XL device and route times (indicated in boldface) reflect the disparity. Following floorplanning and routability determination, it is apparent that placement refinement is needed.

To determine appropriate values for β, the annealing moves-per-iteration variable, and T_{init}, the annealing start temperature for low-temperature annealing, a series of time-quality tradeoffs were evaluated. Starting from a floorplanned placement, each design underwent low-temperature annealing with parameters indicated in Figure 6 and the resulting placement cost was determined relative to the best cost that could be achieved by performing simulated annealing from a random placement for many minutes. Each curve in Figure 6 represents the geometric average of all eight designs over a collection of parameter values. It was found that constraining soft macro logic blocks within the bounds of the soft macro determined during intra-macro placement or within bin boundaries resulted in worse results than allowing soft macro logic blocks to pass between bins. All results shown in the figure were collected without block movement

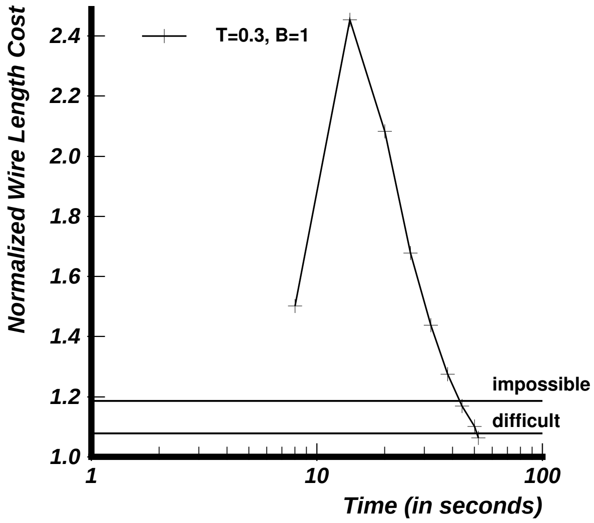

Figure 7 Placement Versus Design Quality - bheap5

Design	Xilinx PAR-M1.4	Floorplan
	MHz	MHz
bheap5	5.08	12.71
bubble16	6.93	14.79
bubble32	10.04	12.48
fft16	21.94	22.47
spm4	8.10	9.20
spm8	6.06	2.07
ssp16	7.40	13.78
ssp32	14.77	13.12

Table 4 Placement Performance Comparison (post-route)

constraints. From the data collected it was determined that for our system the best time-quality tradeoff was achieved for $\beta = 1$ and $T_{init} = 0.3$.

These parameter values were used to refine the initial floorplan for *bheap*5 to a lower cost placement. A graph of relative placement cost versus time at various points during execution is shown in Figure 7 for $T_{init} = 0.3$ and $\beta = 1$. Cost points associated with impossible, difficult and low-stress routing, as defined in Section 5, are labelled. It can be seen that as low-temperature annealing is performed, placement cost is moved from the impossible-to-route range, through difficult, and into the low-stress region.

The effect of this modified placement is clear from the results shown in Table 3. Even though placement time has been extended by 34 seconds, routing time has now been significantly reduced due to the refined placement. Table 4 shows that following routing by Xilinx PAR-M1.4 software, floorplanned (and low-temperature annealed in the case of *bheap*5) circuits exhibit favorable performance characteristics compared to circuit placements created by the PAR placer.

6 CONCLUSION AND FUTURE WORK

In this paper a novel FPGA placement tool has been described that quickly achieves high-quality placement by leveraging design regularity in the form of pre-compiled macro-blocks. While placement achieved through initial macro-based floorplanning steps are shown to be highly routable in most cases, for some designs additional placement refinement may be necessary to achieve a routable placement. The system that has been introduced exhibits this capability by first identifying if a design is routable and then perturbing an initial floorplan with low-temperature simulated annealing.

In this work, algorithms were developed to address placement on existing FPGA architectures. An alternate approach would be to consider modifying island-style FPGA devices to include additional levels of routing hierarchy, effectively isolating intra-macro routing from inter-macro routing. Placement could then be more effectively partitioned into local and global placement steps, much like contemporary multi-FPGA systems.

References

[1] *Field-Programmable Gate Arrays Data Book.* Lucent Technologies, 1996.

[2] *The Programmable Logic Data Book.* Xilinx Corporation, 1996.

[3] J. Babb, M. Frank, V. Lee, E. Waingold, and R. Barua. The RAW Benchmark Suite: Computation Structures for General Purpose Computing. In *Proceedings, IEEE Workshop on FPGA-based Custom Computing Machines*, Napa, Ca, Apr. 1997.

[4] V. Betz and J. Rose. VPR: A New Packing, Placement, and Routing Tool for FPGA Research. In *Proceedings, Field Programmable Logic, Seventh International Workshop*, Oxford, UK, Sept. 1997.

[5] T. Callahan, P. Chong, A. Dehon, and J. Wawrzynek. Fast Module Mapping and Placement for Datapaths in FPGAs. In *International Symposium on Field Programmable Gate Arrays*, Monterey, Ca., Feb. 1997.

[6] J. Emmert and D. Bhatia. A Methodology for Fast FPGA Floorplanning. In *International Symposium on Field Programmable Gate Arrays*, Monterey, Ca., Feb. 1999.

[7] J. Emmert, A. Randhar, and D. Bhatia. Fast Floorplanning for FPGAs. In *Field-Programmable Logic and Applications (FPL'98)*, Tallinn, Estonia, Sept. 1998.

[8] S. Gehring and S. Ludwig. Fast Integrated Tools for Circuit Design with FPGAs. In *International Symposium on Field Programmable Gate Arrays*, Monterey, Ca., Feb. 1997.

[9] A. Koch. Structured Design Implementation - A Strategy for Implementing Regular Datapaths on FPGAs. In *International Symposium on Field Programmable Gate Arrays*, Monterey, Ca., Feb. 1996.

[10] Y. Sankar and J. Rose. Trading Quality for Compile Time: Ultra-Fast Placement for FPGAs. In *International Symposium on Field Programmable Gate Arrays*, Monterey, Ca., Feb. 1999.

[11] C. Sechen. *VLSI Placement and Global Routing Using Simulated Annealing.* Kluwer Academic Publishers, Boston, Ma, 1988.

[12] K. Shahookar and P. Mazumder. VLSI Cell Placement Techniques. *ACM Computing Surveys*, 23(2), June 1991.

[13] N. Sherwani. *Algorithms for Physical Design Automation.* Kluwer Academic Publishers, Boston, Ma, 1992.

[14] J. Swartz, V. Betz, and J. Rose. A Fast Routability-Driven Router for FPGAs. In *6th International Workshop on Field-Programmable Gate Arrays*, Monterey, Ca, Feb. 1998.

[15] R. Tessier. *Fast Place and Route Approaches for FPGAs.* PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 1998. also available as MIT LCS TR-768.

[16] T. Yamanouchi, K. Tamakashi, and T. Kambe. Hybrid Floorplanning Based on Partial Clustering and Module Rest ructuring. In *Proceedings, ACM/IEEE 33rd Design Automation Conference*, 1996.