Fast Placement Approaches for FPGAs

Russell Tessier
University of Massachusetts, Amherst

Recent trends in FPGA development indicate a strong shift toward design reuse through the use of intellectual property (IP). This design shift has motivated the development of Frontier, a timing-driven FPGA placement system that uses design macro-blocks in conjunction 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, FPGA routability and performance metrics are used to evaluate the quality of the initial placement. Finally, if the floorplan is determined to be insufficient from a routability or performance standpoint, a feedback-driven placement perturbation step is employed to achieve a lower cost placement. For a collection of large reconfigurable computing benchmark circuits our timing-driven placement system exhibits a 2.6× speedup in combined place *and* route time versus commercial FPGA CAD software with improved design performance for most designs. It is shown that floorplanning, placement evaluation, and back-end optimization are all necessary to achieve high-performance placement solutions.

Categories and Subject Descriptors: B.7.1 [**Integrated Circuits**]: Types and Design Styles—*Gate arrays*; B.7.2 [**Integrated Circuits**]: Design aids—*Placement and routing*; J.6 [**Computer Applications**]: Computer-Aided Engineering

General Terms: Algorithms, Design, Experimentation, Measurement, Performance

Additional Key Words and Phrases: Computer-aided design of VLSI, field-programmable gate arrays, layout, synthesis

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 tens of 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

This work was supported by National Science Foundation grant CCR-0081405.
Author's address: R. Tessier, Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003; email: tessier@ecs.umass.edu.

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 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, timing-driven 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 targeted to devices with architectures similar to the Xilinx Virtex [Xilinx Corporation 2001] and XC4000 [Xilinx Corporation 1998] and Lucent Orca [Lucent Technologies 1996] 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, 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 and performance evaluator, based on wire length and static timing information, is used to determine if subsequent routing for a given target device is likely to complete successfully with pre-specified timing constraints. If this evaluation 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, enhanced design routability, and improved design performance.

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

2. BACKGROUND

2.1 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 Virtex and XC4000 families and the Lucent Orca family. These architectures are characterized

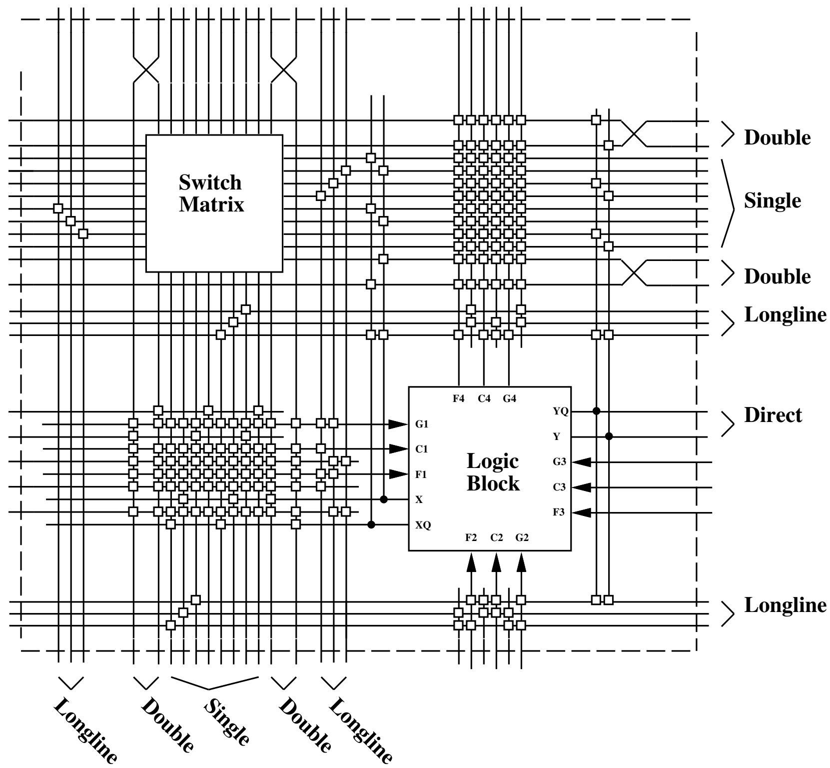

Fig. 1. Xilinx XC4000 logic and routing cell

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* macros are groups of logic blocks with a fixed width w_i and height h_i. Each logic block within a hard macro has a specific δx and δy position relative to the macro origin. Although the macro can be located at many positions throughout the logic array, the relative placement of hard macro logic blocks with respect to the macro origin remains fixed throughout design placement. *Soft* macros are more flexible than hard macros. Initially, like hard macros, soft macros have h_i and w_i rectangular dimensions and each logic block within a soft macro is assigned to a specific location relative to the macro origin. Unlike hard macros, during placement a soft macro can be reshaped into a rectilinear shape that has the same logic block area as the original rectangular 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 and associated timing constraints are met. 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 subject to prespecified timing constraints. Initially, floorplanning considers, among other criteria, minimizing the length of all inter-macro nets $Nets_M$ and minimization of the design inter-macro critical path length. Subsequently, during placement refinement, the length of all design wiring, $Nets_{blocks}$, is considered for both routability and critical path length determination.

2.2 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 and performance for array layout is typically related to 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 [Sechen 1988] 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 [Tessier 1998]. In [Sankar and Rose 1999], 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 [Shahookar and Mazumder 1991] [Sherwani 1992]. 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 [Callahan, Chong, Dehon, and Wawrzynek 1998] [Gehring and Ludwig 1998] [Koch 1996] 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 [Tessier 1998], 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 [Emmert, Randhar, and Bhatia 1998], 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 [Emmert and Bhatia 1999]. 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 targeted 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.

2.3 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 which commu-

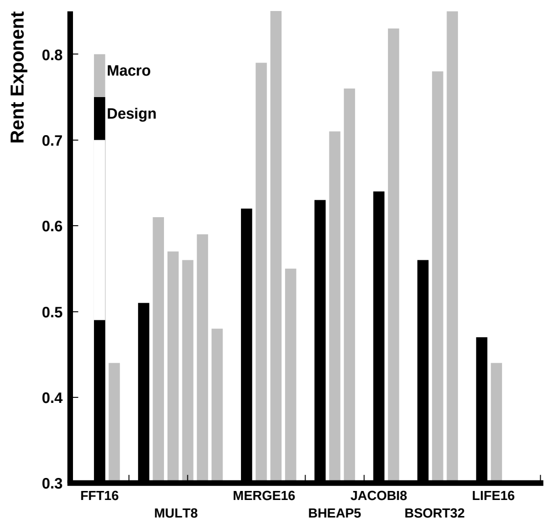

Fig. 2. Rent exponents for macros and designs

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

2.4 Macro-based designs

Most large digital designs created today have a well-defined internal structure. This structure frequently takes the form of large, tightly-connected macro-blocks which can be created and stored in a library. It is possible to motivate floorplanning by examining the circuit structure of reconfigurable computing designs. In general, placement techniques, such as simulated annealing, are based on the notion that wiring demands are roughly equal throughout the device and that minimizing overall wire length will lead to a routable design. Experimental analysis shows, however, that this may not always be the case. For a given design, a known relationship exists between the amount of logic (or number of logic blocks) and the number of wires associated with the design. This relationship is Rent's Rule [Landman and Russo 1971]:

$$\textbf{Rent's Rule}: \quad N = KG^p \tag{1}$$

where N is the number of wires emanating from a region, G is the number of circuit components (or logic blocks), K is Rent's constant, and p is Rent's exponent. The Rent exponent, p, can be used to characterize the routing density in a circuit since it defines the growth requirements of the interconnect.

To determine variations in wiring density, all benchmark designs and component macros were recursively bipartitioned using a mincut partitioner to determine their Rent exponent. In Figure 2, it can be seen that the Rent exponents for the entire

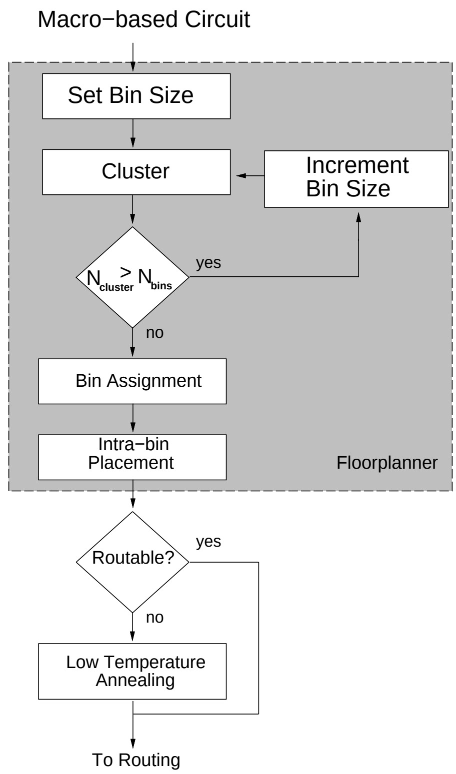

Fig. 3. Frontier placement flowchart

designs listed in Table I in Section 5 may vary significantly from the parameters of individual component macros. Note that even though a macro may appear several times in a design it is shown only once in the graph for each design. The Rent exponent results impact floorplanning in two ways. First, the result indicates that macros are likely to be tightly-connected so that the pre-placement of macros will likely speed up the placement process. Second, the result shows that interconnect resources may be used unevenly across the FPGA device due to placed macros, thus necessitating a placement *smoothing* step to more equally distribute routing globally. The multi-phase Frontier placement system has the capability to address both of these needs though floorplanning and low-temperature annealing.

3. FRONTIER IMPLEMENTATION

3.1 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 3, the layout process starts with a macro-based netlist of soft and hard macros targeted 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 4. To

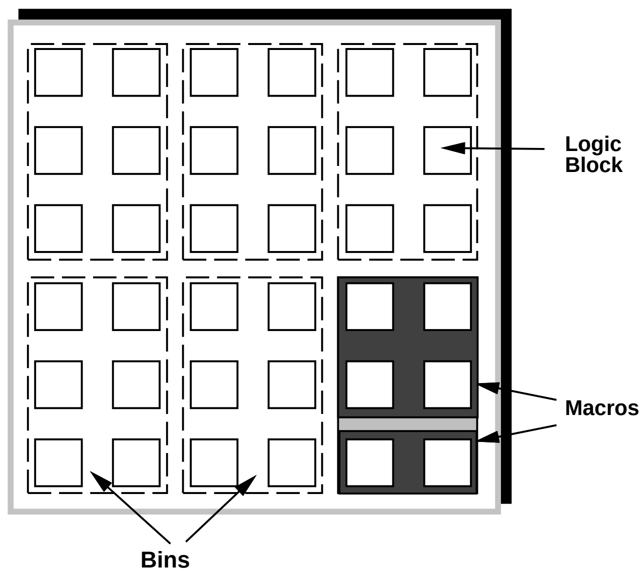

Fig. 4. Bin-based cluster assignment

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 is 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 targeted to contemporary FPGA devices, in some cases additional placement perturbation is required. In Frontier, following floorplanning, a detailed estimate of the placement wire length and post-route design performance is determined, taking into account the special features of the FPGA device. As described previously in [Swartz, Betz, and Rose 1998], these wire length and performance estimates can be used to evaluate whether subsequent device routing will complete quickly, require a long period of time, or ultimately fail. 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 and routability tradeoffs are determined.

3.2 Placement Cost

In order to achieve an efficient layout, placement cost must be efficiently modeled during various placement steps. Traditionally, *routability* for island-style FPGAs has been modeled using bounding box wire length to determine the amount of wiring needed to complete multi-sink connections [Marquardt, Betz, and Rose 2000]. To support *timing-driven* compilation, however, additional source-sink path delay information is needed. From [Marquardt, Betz, and Rose 2000], overall system placement cost can be represented linearly as:

$$Cost = \lambda \times cost_{wiring} + (1 - \lambda) \times cost_{timing} \tag{2}$$

where λ is a weighting factor determined experimentally to be approximately 0.5 for timing-driven layout an 1.0 for routability-driven layout. While the formulation of the above equation remains the same across a number of Frontier placement steps, the accuracy of the cost formulation varies and is described in detail in Section 3.3 for each placement step. Initial placement steps model cost using coarse-grained array distance and estimated critical-path measurements to achieve relative positioning. This inexact metric, used to mitigate congestion and critical path delay during coarse-grained steps, is refined in later fine-grained placement steps in which delay is modeled much more accurately.

Timing cost plays an important role in the determination of timing-driven placement. In order to minimize the critical path of a circuit, timing analysis must be performed at various stages of the placement process. Over the course of placement the critical path in the circuit may change, leading to a need for path re-evaluation. Key aspects in determining timing cost include how often to evaluate path delay and how accurately the timing cost is specified. While the accuracy of the modeled delay may change from step to step, the basic formulation of the cost remains the same. For Frontier, we follow a cost approach [Marquardt, Betz, and Rose 2000] which repetitively evaluates critical path lengths at fixed intervals in determining timing cost. In order to determine circuit delays, placement logic and wiring delay may be represented with a timing graph with each edge given a specific delay. By performing a depth-first search through this graph, it is possible to determine the *arrival time*, T_{arr}, of each signal at destination points in the graph. Following this analysis, critical paths which exceed the *user-specified* clock cycle can be determined.

The next step in timing analysis is determining the time, T_{req}, when each signal must arrive at its destination to meet required timing constraints. This can be determined by performing a breadth-first backwards search from combinational path sinks to sources. At each intermediate point in the search, the *slack* is determined to be the difference between the required and actual arrival times:

$$slack = T_{req} - T_{arr} \tag{3}$$

Timing cost can be determined by evaluating source-sink net delays. To accurately reflect timing, the criticality of each source-sink delay must be weighted

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

Fig. 5. Weighted clustering algorithm

based on slacks determined previously. From [Marquardt, Betz, and Rose 2000], criticality can be determined to be:

$$criticality = 1 - \frac{slack}{D_{max}} \tag{4}$$

where D_{max} is the longest combinational delay in the circuit. In turn, this criticality can be used to determine timing cost based on the relation:

$$cost_{timing} = net\_delay \times criticality^{crit\_exp} \tag{5}$$

where crit_exp is a criticality exponent determined experimentally to be 1. This cost, along with bounding box cost, $cost_{wiring}$, can be combined, as expressed in Equation 2, to determine overall system cost.

3.3 Placement Steps

3.3.1 *Bottom-Up Clustering.* In the first step of placement, macros are clustered together into placement bins of identical dimensions and logic block 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. 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 previously developed in [Tessier 1998], [Yamanouchi, Tamakashi, and Kambe 1996] and [Emmert and

Bhatia 1999] that is weighted to take logic block counts and interconnectivity into account:

$$Cost_{ij} = feas(i,j) \times \frac{N_{blocks}}{N_{M_i} + N_{M_j}} \times \frac{min(N_{M_i}, N_{M_j})}{max(N_{M_i}, N_{M_j})} \times \sum Nets_{ij} \qquad (6)$$

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 6 determines if a candidate cluster can be feasibly shaped during intra-bin placement, using criteria described in Section 3.3.3, 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 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 5.

If, following clustering, more clusters, $N_{cluster}$, than bins, N_{bins}, exist, bin dimensions are modified by increasing bin horizontal and vertical dimensions by 1 logic block and clustering is started again from scratch.

While Equation 6 takes cluster-cluster interconnect into account, its effect is limited for timing-driven costs. To address this additional cost, the $Nets_{ij}$ value in the equation is scaled based on net slack. In general, it was found that the use of simple adjacency rather than scaled adjacency was sufficient for clustering since most design critical paths passed through at most two macros.

3.3.2 *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 [Betz and Rose 1997] with the cost function described in Section 3.2 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.

3.3.3 *Internal Bin Placement.* Once each cluster of macro-blocks is assigned to a specific bin, intra-bin placement is performed to assign logic blocks in macros 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 $\sum_i w_i$, the space between hard macros occupied by soft macros is determined to be $X_{soft} =$

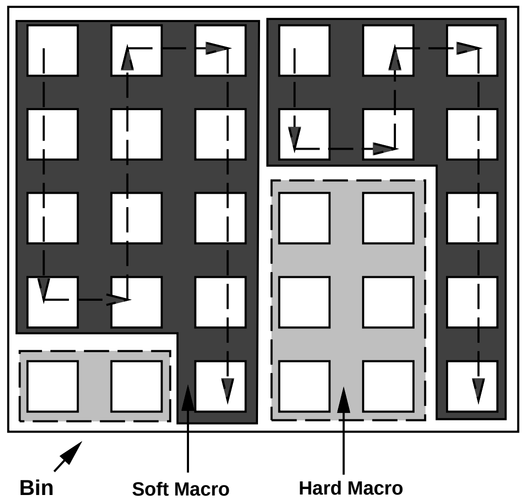

Fig. 6. Internal bin placement

$\lfloor\frac{w_{bin}-\sum_i w_i}{N_{soft}}\rfloor$. This equation leads to an X_{soft} value of 1 for the bin shown in Figure 6. 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. As shown in Figure 6, 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 placement cost outlined in Section 3.2. By following this methodology, up to 100% logic block utilization can be achieved in each bin. For timing-driven floorplanning, slack update within the bin is performed iteratively after each logic block assignment given the small number of logic blocks generally assigned to each bin.

3.3.4 *Routability and Performance Estimation.* 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 [Swartz, Betz, and Rose 1998]. 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 and performance metric [Swartz, Betz, and Rose 1998] 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 [Swartz, Betz, and Rose 1998] 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:

$$W_{min-est} = \lceil wirelength/(2 \times N_{cells} \times U) \rceil \tag{7}$$

where $wirelength$ is the total estimated shortest-path 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.

Post-route performance can be predicted before routing by evaluating the critical path in the placement determined by floorplanning. An estimation of routing delay can be made by evaluating the shortest path from design sources to sinks along wire segments, multiplexers, and look-up tables with a static timing analyzer. For Frontier, a highly accurate model of a Xilinx XC4000XL device was used to determine delay values based on capacitance and resistance values derived from process information.

3.3.5 *Low-Temperature Annealing.* As will be shown in Section 5, simply performing floorplanning is often sufficient to create a placement in the low-stress routing range that meets timing constraints for most designs. In some cases, however, routability and performance evaluation may reveal that the current placement is difficult or impossible to route given available target device routing resources and a desired clock rate. 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. This approach allows for smoothing of wire length across soft macros and bins without destroying the high-level placement structure achieved by the floorplanner. While detailed discussions of the simulated annealing algorithm for placement [Sechen 1988] and associated controlling parameters [Betz and Rose 1997] [Marquardt, Betz, and Rose 2000] 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.

—Range limit, R_{lim} - For annealing, the range limit indicates how far a particular logic block can be moved in a linear dimension during a specific block swap. Typically, this is set to include the entire chip initially and then reduced to include only a small subset of blocks as the annealing temperature is reduced [Betz and Rose 1997]. For low temperature annealing, our system experimentally determined that R_{lim} should initially be set to a value of $0.2 \times \sqrt{N_{cells}}$.

—Inner number, β - This value varies the number of swaps made at each annealing temperature [Sankar and Rose 1999]. In the annealing formulation used to perturb logic block placement, the number of moves at each temperature is set to $\beta \times N_{blocks}^{4/3}$ [Betz and Rose 1997]. 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.

4. EXPERIMENTAL APPROACH

To validate our approach, results from the newly-developed placement system were compared against placement and routing results obtained using Xilinx PAR, version M2.1 software. The experimental approach included internal macro-block placement, Frontier placement, as described in Section 3, and PAR placement and routing using both macro and logic block-based placement modes. The following subsections describe specific experimental steps in detail.

4.1 Macro-block Creation

Prior to design placement, an RTL description of each macro-block was synthesized and mapped to Xilinx XC4000 logic blocks by the Synopsys Design Compiler. Following designation of macro h_i and w_i dimensions, placement for each macro-block was determined using the simulated annealing-based VPR tool set [Betz and Rose 1997]. The result of the macro-block creation step for each macro was the determination of a relative offset from the macro origin for each component logic block. Each *relationally-placed* macro-block was treated as a single unit during subsequent floorplanning, as described in Section 3. Both hard and soft macros were created using this method.

4.2 Frontier Placement

In an initial set of design placement experiments, hierarchical designs containing instantiations of the macro-blocks were placed, as described in Section 3. For each design, following placement with Frontier, a Xilinx user constraints file (UCF) was written specifying the exact array location of each design logic block in the target FPGA array. This file was input to Xilinx PAR-M2.1 to set logic block positions for subsequent PAR routing. For each design, a hierarchical XNF netlist containing logic blocks mapped during the macro-block creation phase was used as input to the PAR router to define logic mapping.

4.3 Xilinx PAR-M2.1 Macro-based Placement

The Xilinx PAR-M2.1 placement tool allows limited support for relationally-placed macros. In a second set of design placement experiments, a hierarchical XNF netlist was created for each design from a macro-level netlist and component macro-block netlists consisting of logic blocks mapped by the Synopsys Design Compiler. Logic

Design	Device	CLBs	Macros	Device Utilization
bheap5	4085XL	2715	30	87%
bubble32	4085XL	2608	63	83%
fft16	4085XL	3032	48	97%
jacobi8	4085XL	2624	64	84%
life16	4085XL	2560	32	82%
merge16	4085XL	2315	63	74%
spm8	4085XL	2425	23	77%
ssp32	4085XL	2370	79	76%

Table I. Macro-based design statistics

blocks in *hard* macros, defined in Section 2.1, were annotated with the relative placement positions determined during the macro creation phase. For each design, this annotated netlist was input to the Xilinx PAR-M2.1 placer. During subsequent PAR placement, the *relative* position of logic blocks in the hard macros was fixed, although the macros could be placed anywhere in the array by the PAR software. No relative positioning constraints were placed on the logic blocks of soft macro-blocks; these were allowed to float around the array independently. Unlike Frontier, which uses multiple placement algorithms and integrated routability and timing estimators, the PAR-M2.1 placer uses a single algorithm, simulated annealing [Trimberger 1994], for both macro and individual logic block placement. Macros and individual logic blocks are simultaneously placed in one annealing-based step.

4.4 Xilinx PAR-M2.1 Flat Placement

In a third set of design placement experiments, hierarchical netlists containing macro instantiations were flattened to include only the logic blocks mapped by Design Compiler during the macro creation stage. No placement information was annotated in the resulting XNF netlists or in UCF constraint files. The *flat* logic block netlists were subsequently placed and routed using Xilinx PAR-M2.1 tools. No macro-block placement information was used during these experiments.

5. RESULTS

In order to show the effectiveness of the floorplanner, we compared its performance to available commercial software operating in both timing-driven and routability-driven mode. In both cases it was possible to achieve both compile time and performance improvements versus off-the-shelf Xilinx PAR-M2.1 software.

The placement system outlined previously was applied to eight macro-based reconfigurable computing benchmarks from the RAW Benchmark Suite [Babb, Frank, Lee, Waingold, and Barua 1997]. Macro-based netlists were input to both the Frontier system and to Xilinx PAR-M2.1, as described in Section 4. Design statistics for the benchmarks appear in Table I. All run time results for both Frontier and PAR were obtained using a 140 MHz UltraSparc I with 288 MB of memory. Routing for all designs was performed using Xilinx PAR-M2.1 software with default placement and routing effort settings.

Execution times (s)										Tracks
	PAR-M2.1 - flat			PAR-M2.1 - macro			Floorplan Only			W_{min}
Design	Place	Rte	Tot	Place	Rte	Tot	Fplan	Rte	Tot	$-est$
bheap5	441	**120**	561	275	1034	1309	16	**329**	345	36
bubble32	428	138	566	195	240	435	28	120	148	31
fft16	537	90	627	120	92	212	10	83	93	22
jacobi8	242	41	283	108	73	181	18	40	58	18
life16	304	109	413	81	144	225	5	89	94	16
merge16	257	45	302	155	72	227	42	100	142	27
spm8	317	262	579	160	159	319	4	158	162	26
ssp32	344	102	446	180	106	286	38	76	114	22
total	48m	15m	63m	21m	32m	53m	3m	16m	19m	

Table II. Design layout statistics - routability-driven

5.1 Routability-driven Results

To explore the efficiency of using the new Frontier placement system, a set of experiments was performed without explicit specification of timing constraints. These experiments focused on completing design place and route as quickly as possible and used design wire length as the sole cost metric ($\lambda = 1$ in Equation 2). Following floorplanning, the routability estimator was used to determine if low-temperature simulated annealing was necessary to smooth the placement.

Execution times for Frontier floorplanning (without low temperature refinement), PAR placement for a netlist with relationally-placed hard macros and PAR placement for a flat netlist appear in Table II in seconds, unless otherwise noted. All designs were routed using PAR with default settings. In general, Frontier provided the fastest layout and PAR placement with relationally-placed hard macros provided the second-fastest. For all designs except one (*bheap*5), routing times for the floorplanned designs were comparable to those found by the PAR-M2.1 placer that required 7× 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, as shown in Table II. $W_{min-est}$ values were determined by measuring the post-floorplan wire lengths of designs and then directly correlating them to required track counts via Equation 7. From Table II 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 in Table II) 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 for routability-based layout. Starting from a floorplanned placement, each design underwent low-temperature annealing with parameters indicated in Figure 7 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 7 represents the geometric average of all eight designs over a collection of parameter values. Note that the absolute value of T_{init} is dependent on the exact formulation

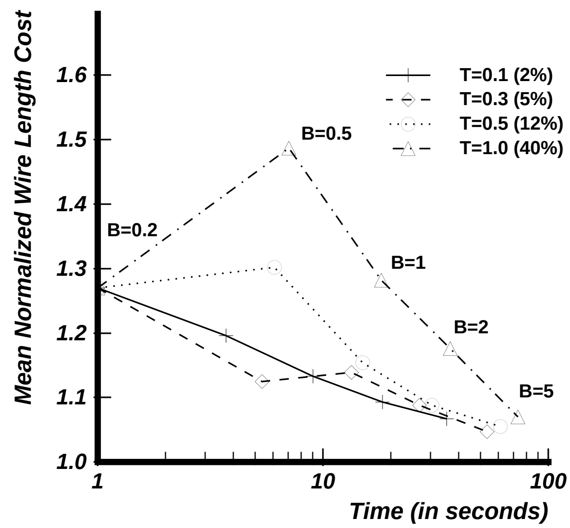

Fig. 7. Simulated annealing parameter variation

Execution times						Performance
Flow	PAR	Fplan	Low Temp Anneal	Route	Total	(MHz)
PAR-flat	441s	0	0	120s	561s	10.5
PAR-macro	275s	0	0	1034s	1309s	5.8
Fplan Only	0	16s	0	329s	345s	11.5
Fplan/Anneal	0	16s	34s	177s	227s	12.8

Table III. Routability-driven layout execution time/performance comparison - design bheap5

of the cost function shown in Equation 2. As a means to promote comparison to other annealing formulations, we have included the percentage of first-annealing-iteration cost-increasing swaps caused by each start temperature in the legend of Figure 7. 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 constraints. From the data collected, it was determined that for our system and cost formulation 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 8 for $T_{init} = 0.3$ and $\beta = 1$. Cost points associated with impossible, difficult and low-stress routing, as defined in Section 3.3.4, 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 III. Even though placement time has been extended by

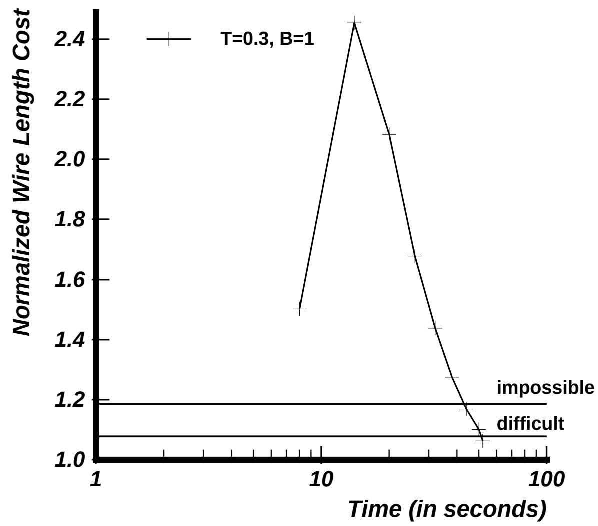

Fig. 8. Placement versus design quality - bheap5

	PAR-M2.1 - flat				PAR-M2.1 - macro			
	Perf.	Execution time			Perf.	Execution time		
Design	(MHz)	Place	Route	Total	(MHz)	Place	Route	Total
bheap5	13.6	41m	125m	166m	8.2	19m	130m	149m
bubble32	14.9	50m	105m	155m	15.6	25m	35m	60m
fft16	24.3	37m	65m	102m	26.0	5m	9m	14m
jacobi8	29.4	16m	71m	87m	31.9	5m	6m	11m
life16	23.8	37m	142m	179m	26.6	2m	14m	16m
merge16	16.4	55m	78m	133m	16.7	27m	13m	40m
spm8	10.9	35m	128m	163m	11.5	11m	21m	32m
ssp32	15.1	46m	59m	105m	14.1	12m	30m	42m
total	16.8	317m	773m	1090m	18.8	106m	258m	364m

Table IV. Best possible timing-driven performance - Xilinx PAR-M2.1

34 seconds, routing time has now been significantly reduced due to the refined placement.

5.2 Timing-driven Results

While the optimization of FPGA place and route time is important for some applications, for many others it is considerably more important to achieve specific design performance objectives. In a set of timing-driven experiments, the cost function described in Equation 2 is used to control floorplanning and subsequent low-temperature annealing. For timing-driven experimentation the value λ in the equation is set to 0.5 to balance the effect of wire length and delay.

Before timing-driven experiments with the Frontier system were performed, each design was placed and routed using Xilinx PAR-M2.1 software for both flat and

	Floorplan/Anneal				
Design	Perf.	Fplan	Anneal	Route	Total
	(MHz)				
bheap5	14.5	19s	78s	34m	36m
bubble32	15.6	26s	0s	15m	15m
fft16	25.0	11s	50s	10m	11m
jacobi8	33.3	19s	27s	9m	10m
life16	28.6	5s	0s	5m	5m
merge16	16.7	43s	21s	19m	20m
spm8	11.5	4s	0s	32m	32m
ssp32	16.4	35s	26s	10m	11m
ave/total	20.2	2.7m	3.4m	134m	140m

Table V. Best possible timing-driven performance - fplan/anneal

	Xilinx PAR-M2.1 - flat		Xilinx PAR-M2.1 - macro		Floorplan/Anneal	
Design	Perf.	Total	Perf.	Total	Perf.	Total
	(MHz)	(m)	(MHz)	(m)	(MHz)	(m)
bheap5	13.6	166m	8.2	149m	14.5	36m
bubble32	14.9	155m	15.6	60m	15.6	15m
fft16	24.3	102m	26.0	14m	25.0	11m
jacobi8	29.4	87m	31.9	11m	33.3	10m
life16	23.8	179m	26.6	16m	28.6	5m
merge16	16.4	133m	16.7	40m	16.7	20m
spm8	10.9	163m	11.5	32m	11.5	32m
ssp32	15.1	105m	14.1	42m	16.4	11m
ave/total	16.8	1090m	18.8	364m	20.2	140m

Table VI. Best possible timing-driven performance - summary

relationally-placed hard macro netlists, as described in Section 4. For each design, timing constraints were set to a series of target clock cycles and the best possible clock cycle that successfully routed was determined. Resulting design clock frequencies and place-and-route times appear in Table IV. Following this analysis, the Frontier placement system in timing-driven mode was applied to the same benchmark designs. As shown on the right side of Table V, timing-driven floorplanning of each design was completed in less than one minute. Following floorplanning, it was possible to determine if the designs would successfully route and meet post-route performance requirements through the use of routability and performance estimators described in Section 3. Based on performance and routability metrics gained from initial placements, low-temperature annealing was performed on the five designs that did not meet either routability or performance constraints. Of the five benchmarks that required annealing, design *bheap5* required adjustment due to routability considerations, while the remaining four were adjusted due to performance concerns.

Before performing low-temperature annealing, it was necessary to determine the appropriate annealing start temperature for performance-driven low-temperature annealing. The results of this experimentation for the five affected designs are shown in Figure 9. As mentioned in the previous section, the absolute value of the

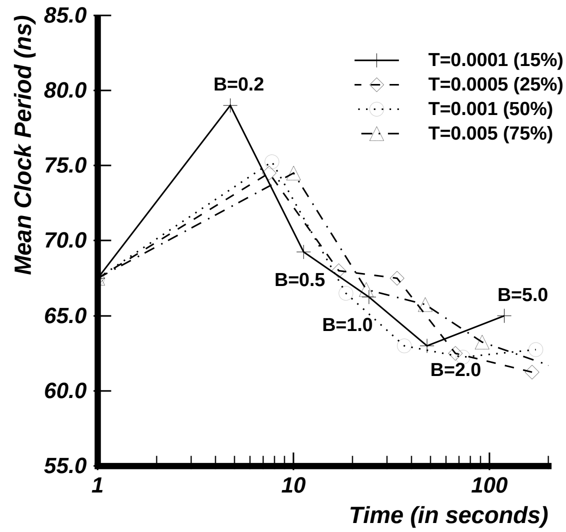

Fig. 9. Timing-driven annealing parameter variation

Execution times						Performance
Flow	PAR	Fplan	Low Temp Anneal	Route	Total	(MHz)
PAR-flat	41m	0	0	125m	166m	13.69
PAR-macro	19m	0	0	130m	149m	8.22
Fplan Only	0	19s	0	52m	52m	14.28
Fplan/Anneal	0	19s	78s	34m	36m	14.49

Table VII. Timing-driven layout execution time/performance comparison - design bheap5

start temperatures is dependent on the formulation of the cost function. As a result, the *percentage* of cost-increasing swaps that are accepted during the first annealing iteration are provided in the legend of the figure to promote comparison to other annealing formulations. In general, the best results for performance improvement were achieved for $T_{init} = 0.0005$ and $\beta = 1$.

Overall, it was found that the Frontier system could create a layout with performance characteristics similar to those created by the Xilinx PAR-M2.1 software in a fraction of the time. A summary of total place-and-route time and design performance for the three approaches appears in Table VI. In seven out of eight cases the Frontier placement approach was able to match or better the design performance result of PAR placement for both macro and flat netlists. Overall placement time was accelerated by a factor of 17 and combined place-and-route time was improved by a factor of 2.6 versus the PAR-M2.1 software in macro-based mode. Overall placement time was accelerated by a factor of 50 and combined place-and-route time was improved by a factor of 8 versus the PAR-M2.1 software for flattened designs.

The importance of the low-temperature annealing step for timing-driven performance improvement can be seen in Table VII. By performing a small amount of low-temperature refinement on design *bheap5* not only was the total place-and-route time reduced, but also the design performance was improved.

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 efficient in most cases, for some designs additional placement refinement may be necessary to achieve a routable placement that meets given timing constraints. The system that has been introduced exhibits this capability by first identifying if a design placement meets timing constraints and is routable. If one of these conditions is not met, the initial floorplan is perturbed 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

Babb, J., Frank, M., Lee, V., Waingold, E., and Barua, R. 1997. The RAW benchmark suite: computation structures for general purpose computing. In *Proceedings, IEEE Workshop on FPGA-based Custom Computing Machines* (Napa, Ca, Apr. 1997). 161–171.

Betz, V. and Rose, J. 1997. VPR: A new packing, placement, and routing tool for FPGA research. In *Proceedings, Field Programmable Logic, Seventh International Workshop* (Oxford, UK, Sept. 1997). 213–222.

Callahan, T., Chong, P., Dehon, A., and Wawrzynek, J. 1998. Fast module mapping and placement for datapaths in FPGAs. In *International Symposium on Field Programmable Gate Arrays* (Monterey, Ca., Feb. 1998). 123–132.

Emmert, J. and Bhatia, D. 1999. A methodology for fast FPGA floorplanning. In *International Symposium on Field Programmable Gate Arrays* (Monterey, Ca., Feb. 1999). 47–56.

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

Gehring, S. and Ludwig, S. 1998. Fast integrated tools for circuit design with FPGAs. In *International Symposium on Field Programmable Gate Arrays* (Monterey, Ca., Feb. 1998). 133–139.

Koch, A. 1996. Structured design implementation - a strategy for implementing regular datapaths on FPGAs. In *International Symposium on Field Programmable Gate Arrays* (Monterey, Ca., Feb. 1996). 151–157.

Landman, B. and Russo, R. 1971. On a pin versus block relationship for partitions of logic graphs. *IEEE Transactions on Computers C-20*, 12 (Dec.), 1469–1479.

Lucent Technologies. 1996. *Field-Programmable Gate Arrays Data Book*. Lucent Technologies.

Marquardt, A., Betz, V., and Rose, J. 2000. Timing-driven placement for FPGAs. In *International Symposium on Field Programmable Gate Arrays* (Monterey, Ca., Feb. 2000). 203–213.

SANKAR, Y. AND ROSE, J. 1999. Trading quality for compile time: ultra-fast placement for FPGAs. In *International Symposium on Field Programmable Gate Arrays* (Monterey, Ca., Feb. 1999). 157–166.

SECHEN, C. 1988. *VLSI Placement and Global Routing Using Simulated Annealing*. Kluwer Academic Publishers, Boston, Ma.

SHAHOOKAR, K. AND MAZUMDER, P. 1991. VLSI cell placement techniques. *ACM Computing Surveys 23*, 2 (June), 145–220.

SHERWANI, N. 1992. *Algorithms for Physical Design Automation*. Kluwer Academic Publishers, Boston, Ma.

SWARTZ, J., BETZ, V., AND ROSE, J. 1998. A fast routability-driven router for FPGAs. In *6th International Workshop on Field-Programmable Gate Arrays* (Monterey, Ca, Feb. 1998). 140–149.

TESSIER, R. 1998. *Fast Place and Route Approaches for FPGAs*. Ph. D. thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science. also available as MIT LCS TR-768.

TRIMBERGER, S. 1994. *Field-Programmable Gate Array Technology*. Kluwer Academic Publishers, Boston, Ma.

Xilinx Corporation. 1998. *The Programmable Logic Data Book*. Xilinx Corporation.

Xilinx Corporation. 2001. *Virtex II Data Sheet*. Xilinx Corporation.

YAMANOUCHI, T., TAMAKASHI, K., AND KAMBE, T. 1996. Hybrid floorplanning based on partial clustering and module restructuring. In *Proceedings, IEEE International Conference on Computer-Aided Design* (Nov. 1996). 478–483.