Trading Quality for Compile Time: Ultra-Fast Placement for FPGAs

Yaska Sankar and Jonathan Rose
Department of Electrical and Computer Engineering
University of Toronto
Toronto, ON, Canada M5S 3G4
{yaska, jayar}@eecg.toronto.edu

Abstract

The demand for high-speed FPGA compilation tools has occurred for three reasons: first, as FPGA device capacity has grown, the computation time devoted to placement and routing has grown more dramatically than the compute power of the available computers. Second, there exists a subset of users who are willing to accept a reduction in the quality[1] of result in exchange for a high-speed compilation. Third, high-speed compile has been a long-standing desire of users of FPGA-based custom computing machines, since their compile time requirements are ideally closer to those of regular computers.

This paper focuses on the placement phase of the compile process, and presents an ultra-fast placement algorithm targeted to FPGAs. The algorithm is based on a combination of multiple-level, bottom-up clustering and hierarchical simulated annealing. It provides superior area results over a known high-quality placement tool on a set of large benchmark circuits, when both are restricted to a short run time. For example, it can generate a placement for a 100,000-gate circuit in 10 seconds on a 300 MHz Sun UltraSPARC workstation that is only 33% worse than a high-quality placement that takes 524 seconds using a pure simulated annealing implementation. In addition, operating in its fastest mode, this tool can provide an accurate estimate of the wirelength achievable with good quality placement. This can be used, in conjunction with a routing predictor, to very quickly determine the routability of a given circuit on a given FPGA device.

1. Introduction

One of the reasons the use of FPGAs and CPLDs has risen dramatically is because they provide quick manufacturing turnaround times [Brow92]. This advantage has been reduced, however, as the capacities of the programmable devices grow, because the compilation times for large circuits is growing more rapidly than the available computer power. This adversely impacts FPGA hardware designers (who must wait longer), emulation system users (who must compile hundreds of FPGAs at a time), and FPGA-based custom computing machine users, who really want compilation times similar to those of a microprocessor.

The placement and routing times for large FPGAs (those with more than 5000 LUT/flip-flop pairs) can last many hours of a day with no guarantee of successful completion. For example, an 8383 LUT (approximately 100,000 gates) circuit requires almost 1.2 hours for placement and routing using the Xilinx M1 (version 4.12) tools on a 300 MHz Sun UltraSPARC [Swar98b]. With million-gate capacity FPGAs on the horizon, these prohibitively long compile times may nullify the time-to-market advantage of FPGAs. We contend that there is a subset of designers who are willing to trade quality for speed of compilation.

In this paper, we focus on the placement phase of the FPGA compile process, and present an ultra-fast placement tool that aims to minimize area. Although a timing-driven placement tool is likely also important, we believe that area-based minimization is a prudent first step.

It is instructive to describe the scenario in which fast compile would be used: a user has designed a circuit and chosen a target FPGA of a specific size. If the user explicitly states a compile time restriction, then the tool provides a prediction as to whether the circuit will successfully route or not in the given time. Swartz et al. [Swar98a] provide a method for making the "fit/no-fit" prediction, given a placement and the total wirelength. Our work provides both a fast way of obtaining the placement and a very fast way of measuring the wirelength. A different scenario is that the user is supplied with an area-compile time trade-off curve, and selects the point appropriate to his goals. In this case there must be sufficient space in the FPGA.

Those users willing to sacrifice area of the circuit mapped to the FPGA for compile time (and can do so via a tunable "knob" on the CAD tools), can accommodate the increased area in several ways depending upon the field of application: hardware designers can reduce the complexity of a single design by partitioning the circuit onto multiple FPGAs, or can select an FPGA with greater logic capacity. They can also eliminate part of the circuit by reducing the amount of parallelism in the hardware.

1.1 Background

There exists a great deal of previous work on VLSI placement algorithms that can be applied to FPGAs [Hana72] [Dunl85] [Sech88] [Sun95] [Klei91] [Shah91]. These algorithms endeavour to minimize the wiring area occupied by a circuit, and succeed to varying degrees. However, few of these algorithms have as their primary goal the minimization of run time. Gehring and Ludwig [Gehr98] describe a fast placement tool for the Xilinx XC6200 FPGA architecture that converts an HDL specification into an FPGA programming bitstream. Their constructive placement algorithm operates only on a hierarchical description of a circuit with regular sub-circuits. It takes user-specified position hints and proceeds in a bottom-up fashion to place the inner-most subcircuits, and then recursively places the larger structures and expression

[1] We define quality as the wiring area required by the circuit or the speed at which the circuit can operate when mapped to the FPGA. Greater wirelength will require the use of a larger FPGA or the use of more resources on a given FPGA than is otherwise necessary.

trees. The placement algorithm is of linear complexity and is fast - a circuit of 11,748 CLBs was placed in 33.5 seconds on a 166 MHz Pentium, with the Xilinx XC6264 as the target device.

Callahan et al. [Call98] combine fast placement with module mapping for datapath circuits by treating the problems jointly as a tree covering problem. Dataflow graph representations of circuits are split into trees, and a linear-time implementation of dynamic programming is used to perform the simultaneous module mapping and relative module placement, with a greedy heuristic being employed to do global placement of the trees. They obtain good results when targeting the Xilinx XC4000 and explore the trade-off between optimizing for area and delay.

[Sun95] and [Betz97] offer methods to speed up simulated-annealing-based placement algorithms, some of which we employ in our tool. The hierarchical clustering and placement algorithm proposed in [Sun95] first performs two levels of clustering to condense a netlist by collapsing as many nets as possible into clusters. A three-stage annealing schedule is subsequently employed to place the different levels of clustered netlists. This entails first performing a high temperature anneal on the highest-level clusters. Then, the next lower level of clusters are annealed across the cluster boundaries set by the previous stage of annealing. Finally, a low-temperature anneal is conducted using the original flat netlist. This hierarchical clustering and simulated annealing-based placement technique is used in TimberWolfSCv7.0, a placement tool for standard-cells.

In [Betz97], a novel, dynamic, adaptive annealing schedule is described for the simulated annealing-based placement algorithm within the placement tool named VPR. The annealing parameters are adjusted automatically depending upon the size of the circuit. A bounding box wirelength cost function is used, with correction factors for multi-terminal nets. The initial temperature is computed as being proportional to the standard deviation in cost after a set of N pairwise swaps are made, where N is the total number of logic blocks and I/O pads in the circuit. At each temperature, $10 \cdot N^{4/3}$ moves are attempted, by default, and the temperature is reduced in such a way as to maintain a constant, useful acceptance rate.

The application of clustering and simulated annealing to the partitioning of FPGA circuits is described in [Roy93], with emphasis on both wirelength and execution time. In [Tess98], compile-time efficient placement for FPGAs is approached using ASIC floorplanning techniques. By considering portions of the circuit being mapped to the FPGA as pre-placed and pre-routed macrocells, the compile times for large designs can be decreased from an hour to mere minutes, although there is both a severe area and circuit speed penalty. As the other portion of the Fast Compile Project at the University of Toronto, [Swar98a] addresses the routing phase of the FPGA compile.

1.2 Paper Organization

This paper is organized as follows: Section 2 describes the ultra-fast placement algorithm and the features that enable the area-time trade-offs. Section 3 describes the target FPGA architecture and the suite of test circuits, and compares the run time and quality of our fast placement tool to those of VPR [Betz97] [Betz98]. It also demonstrates the accuracy of our tool as a high-speed wirelength predictor. Section 4 concludes and offers direction for future work.

2. Ultra-Fast Placement Algorithm

In this section, we describe the ultra-fast placement algorithm and the parameters that allow us to exchange wirelength for compile time. We then describe how we determined a stable set of these parameters that give us the best quality-time trade-off. More elaborate details of the algorithm and parameters may be found in [Sank99].

2.1 Overview of Approach

The placement problem for FPGAs begins with a technology-mapped netlist of logic blocks[1], I/O pads, and their interconnections. The output is an assignment of the blocks and pads to specific physical locations of the FPGA. To achieve ultra-high-speed placement for FPGAs, we build upon the clustering and hierarchical simulated annealing algorithm described in [Sun95] and the adaptive annealing schedule of [Betz97] [Betz98] and integrate it into the infrastructure provided by VPR (the Versatile Place and Route tool presented in [Betz97]).

Figure 1 shows the framework for our ultra-fast placement algorithm. The first stage is a multi-level, bottom-up clustering of the logic blocks based on their connectivity. (Note that we do not incorporate I/O pads into the clusters of logic blocks, since they have special restrictions upon where they can be placed on the physical FPGA.) The bottom-up clustering is parameterized as follows: a total of L different levels of clustering will be performed. At each level i, s_i blocks (or clusters) at the previous level are grouped into a cluster. If a circuit contains a total of N logic blocks, after a single level of clustering (level 1), there are $\lceil N/s_1 \rceil$ clusters. These clusters can be grouped again to create a second level of clustering, with s_2 first-level clusters in each second-level cluster, giving $\lceil \lceil N/s_1 \rceil / s_2 \rceil$ clusters at the top level (level 2), and so on.

Once all the required clustering is done, placement must be performed at each level of the hierarchy. We employ a two-step approach at each level: an initial constructive placement followed by an iterative improvement step using simulated annealing. The parameters of the anneal are tuned to acquire a good quality-time trade-off, as described below. Figure 2 illustrates an abstract view of multi-level clustering and placement. Our goal is to achieve high-speed placement by quickly making good and fast global decisions at the higher levels of the hierarchy, and following this with iterative local improvement at the different levels of granularity. Our choices of algorithms are guided by the following objective: reduce the complexity of large placement problems by dividing them into manageable portions, and then employ known heuristics that are simple, fast, and effective on each portion.

2.2 Multiple-Level Clustering

The first step of the ultra-fast placement algorithm is a multi-level bottom-up clustering of logic blocks based on their connectivity. The input to the clustering step is a netlist of N logic blocks and their interconnections, the number of clustering levels, L, and the cluster size at each level, s_1, s_2, ... s_L. We restrict the cluster sizes (s_i) to be perfect squares (4, 16, 25, 64...) in order to simplify the grid resizing operations at the various levels. The task is to create L separate netlists of clusters of logic blocks and their

[1] For this paper, a logic block is one 4-input lookup table (4-LUT) and one D flip-flop.

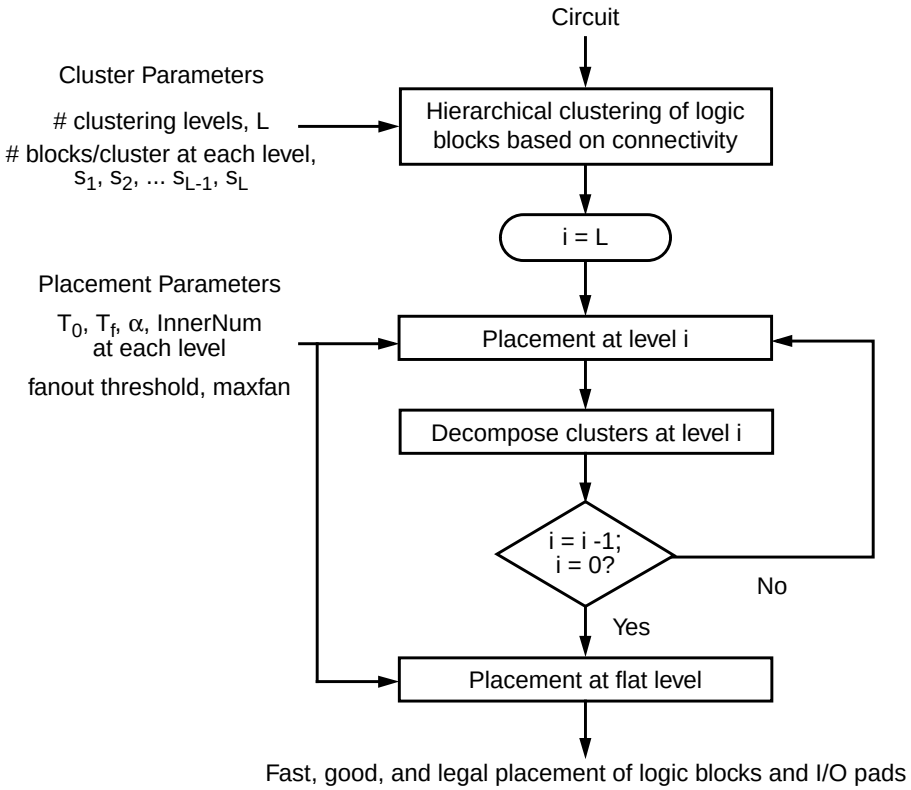


Figure 1. High-level view of fast placement algorithm.

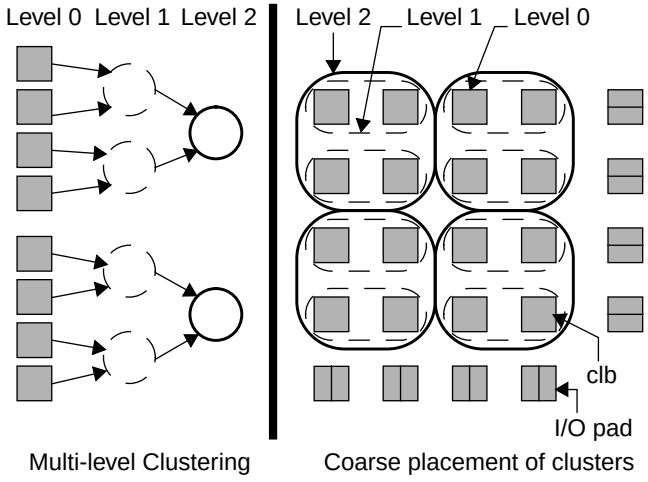


Figure 2. Abstract view of multi-level clustering and placement.

interconnections where each block or lower-level cluster is assigned to a unique higher-level cluster exactly once, and each cluster $c_{i,k}$ (the kth cluster at level i) has at most s_i blocks or clusters from the previous level.

The clustering algorithm begins by randomly choosing a logic block as a seed, and assigning it to the first slot in a cluster. Each unclustered block connected to that seed is assigned a score that rates how much the block belongs to this cluster. This score, w_b, for each candidate block b has two components: (1) the number of connections between the candidate and the cluster being constructed, with each connection weighted by the fanout of the net on which it lies, as in [Sun95], and (2) the number of nets that would be completely absorbed if this candidate were added to the current cluster. We say that a net is *absorbed* by a cluster if all the blocks on that net are contained within that single cluster. If we denote J to represent the set of nets shared between the candidate block b and the cluster c under construction, P_j as the set of pins on net $j \in J$, and A_{bc} as the set of nets absorbed by adding candidate block b to cluster c, then the score can be expressed as

$$w_b = \sum_{j \in J} \frac{1}{|P_j| - 1} + |A_{bc}| \qquad (1)$$

With this function, blocks on low-fanout nets and on nets that are about to be absorbed are preferred when building the clusters. The candidate block with the highest score is added to the next available cluster slot, and if the cluster is full, a new one is started with a new randomly selected seed block. This process is repeated until all the blocks are clustered. The result is a netlist of clusters with absorbed nets removed. We proceed in a similar manner to create further levels in the clustering hierarchy.

The number of clustering levels, L, and the size of the cluster at each level, s_i, can be varied to allow the trade-off of compile time and quality. As the size of the clusters increases, the placement problems become simpler because more is hidden, but there is less accurate representation of the netlist and therefore lower quality may result.

2.2.1 Complexity of Clustering

The score assigned to any candidate block changes only when a net is first connected to a cluster or when a net is about to be absorbed (i.e. all but block *b* of the pins on net *j* are contained in cluster *c*, and the cluster has an available slot). We can maintain a list of the best scores and associated candidates in a bucket data structure in order to perform fast updates. The bucket structure only needs to be flushed when a cluster is full. Let N be the number of logic blocks, K be the number of nets on each logic block, f_{max} be the maximum fanout of a net in the circuit and s be the size of the cluster. The complexity of the algorithm can be derived by observing that when generating the clusters, the algorithm must examine each of the N blocks once, each of the K nets connected to the block, and each of the other pins on those nets. This examination occurs either upon adding a block to a cluster or when a net is about to be absorbed. The complexity of the clustering algorithm is thus $O(N \cdot K \cdot f_{max})$. If we clip the value of f_{max} by restricting the clustering algorithm from examining nets above a certain fanout threshold, this is a linear-time algorithm. This bound is satisfied at higher levels of clustering as well, since N is scaled down by a factor of the cluster size s, K is scaled up by at most a factor of s (and is often less than that), and f_{max} is likely to decrease. Practically, the clustering is very fast: a 20,000 LUT circuit can be grouped into clusters of size 64 in 2.1 seconds on a 300 MHz Sun UltraSPARC.

2.3 Placement of Clusters at Each Level

Once we have constructed the hierarchy of clusters, placement must occur at each level. The placement algorithm consists of two steps: constructive placement followed by annealing-based iterative improvement.

2.3.1 Constructive Placement of Clusters

Given a netlist of clusters and their interconnections, we first perform a random placement of all the I/O pads in the circuit at the highest level of the hierarchy. This provides anchor points for the clusters. Note that subsequent optimization steps will change the pad placement.

The constructive placement determines positions for three separate groups of clusters: (1) those connected to output pads, (2) those connected to input pads, and (3) those connected to other logic clusters. It computes, for each cluster in each group in succession, the arithmetic mean position of all the clusters and pads it is connected to that have already been placed. The cluster is placed as close to this "center of gravity" as possible. The initial placement of the pads provides the initial guidance for this construction. We have found that this method provides a superior starting point for the subsequent iterative improvement step than a simple random placement. Experiments also show that this placement results in a slightly better time/quality trade-off than a random placement.

At lower levels in the hierarchy, the same constructive approach is used, with three exceptions: (1) there is no initial pad placement - pads are placed in the same way logic clusters are; (2) if a block has not yet been placed and its position is needed for the mean calculation, the center of the higher-level cluster it is contained within is used as the position; (3) each of the cluster contents is placed as close to its calculated "center of gravity" while remaining within the prescribed cluster boundaries.

2.3.2 Simulated-Annealing-Based Iterative Improvement of Placement

Following the constructive placement of clusters and pads at any level in the hierarchy, we improve its quality using simulated annealing-based [Kirk83] [Sech85] iterative improvement. We will assume that the reader is familiar with the basic simulated annealing method as it is applied to placement. We have adapted the annealing implementation in VPR described in [Betz97][Betz98].

One important issue is whether or not to restrict the motion of blocks to remain within the cluster boundaries of the most recent cluster level. We have experimentally determined that it is much better to allow the blocks being placed to *move across* the cluster boundaries. This still means that the coarse placement from the previous level is useful; if the boundaries are enforced, however, then quality suffers.

The key parameters that control the quality-time trade-off for simulated annealing are:

1. The starting temperature, T_0. This is a crucial parameter, because if the temperature is too high, the annealing will destroy the placement structure developed at previous levels in the hierarchy. If it is too low, then insufficient optimization will be done. We employ three different mechanisms for determining T_0. The first is to employ the temperature "measurement" mechanism suggested in [Rose90] - here the temperature is determined by finding the temperature at which the placement appears to be at equilibrium (simulated thermometer). The second is to do a simple quench, and the third is to set the starting temperature to a fixed value. In the next section, we explore which of these approaches is most appropriate for different time-quality trade-off points.

2. The number of "moves" per temperature, called "*InnerNum*." The basic annealing algorithm of VPR [Betz97] makes $InnerNum \cdot N_{blocks}^{4/3}$ moves at each temperature, where N_{blocks} is the number of blocks and pads. The parameter *InnerNum* determines how much work is done per temperature.

3. The temperature update factor, α. The lower α is, the faster the anneal, but the worse the quality. VPR [Betz97] automatically adjusts α as described in the introduction; we have found that squaring the automatic α increases speed with little reduction in quality.

4. The exit criterion - what causes the annealing to stop - is either a specified temperature at which the annealing terminates (T_f) or when either of the following two conditions are met: (i) the temperature is less than 1% of the average cost per net or (ii) the average cost over the last three temperatures remains unchanged.

In summary, we have identified 3 types of schedules that permit us to explore the quality-time space thoroughly: (1) an aggressive dynamic adaptive schedule with automatic calculations for T_0, T_f, and α; (2) a quench (all temperature 0 moves), where no hill-climbing is permitted; (3) a manually-specified schedule where the values of T_0, T_f and α are fixed. Schedule (1) is an anneal tailored

to the current placement of the circuit, whatever its level of granularity, schedule (2) is a greedy heuristic, and schedule (3) is a short, fixed anneal. In all three cases, we can trade quality for compile time by varying the *InnerNum* parameter.

2.3.3 Fanout

Another enhancement that we implement to speed up the placement is to ignore nets with large fanout. This is useful because a high-fanout net will likely cover much of the FPGA and so it is harder to reduce that area. By ignoring nets above a certain fanout threshold, we make the placement problem simpler. If we set the threshold too low, however, we may lack enough information to create a good placement. Note that both the clustering and placement steps ignore the nets above the threshold.

2.3.4 Complexity of Placement

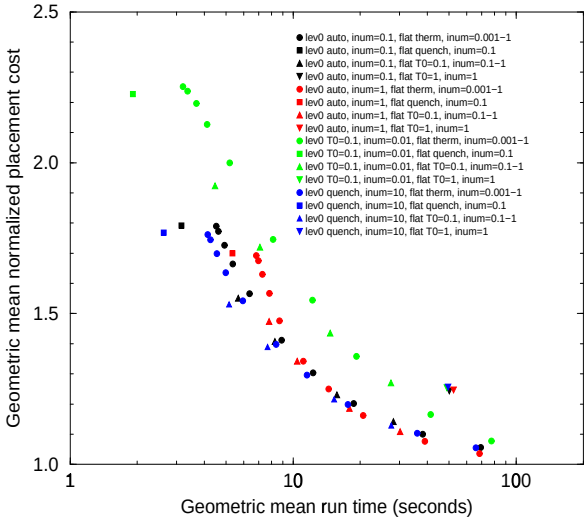
At any level in the hierarchy, our initial constructive placement algorithm has worst-case time complexity $O(N_{blocks} \cdot K \cdot f_{max})$, with N_{blocks} logic blocks and pads, K pins per block, and a maximum fanout of f_{max} for any net in the circuit. Just as with the clustering algorithm, this is because we must examine each block or cluster exactly once, each net connected to that block or cluster, and every other block or cluster connected by that net. Furthermore, by examining only those nets below a certain fanout threshold, we can ensure that it remains a linear-time algorithm. Assume there are N logic blocks and $(PI+PO)$ pads in the circuit, and that we choose a uniform cluster size of s at each level of the hierarchy. For the follow-up simulated annealing algorithm, we explore at each level i (i = 0...L) at most $InnerNum \cdot ((N/s^i)+PI+PO)^{4/3}$ configurations per temperature, and our starting temperature calculation and aggressive adaptive annealing schedule typically ensure that we do not search through many temperatures per level in the clustering hierarchy. This means that the annealing algorithm's worst-case time complexity is bounded by $O(N_{blocks}^{4/3})$ and is typically less than that.

2.4 Determination of the Quality-Time Envelope Parameters

In this section, we describe the experiments used to identify the set of parameters for the ultra-fast placement tool and choose those parameters. There are two sets of parameters: those that control the clustering, and those that control the iterative improvement of the placement. Our goal is to determine the parameters that lead to the best quality-time trade-off, which we call the *envelope* parameters. Please note that the details of the actual FPGA architecture and the other parts of the CAD flow are given in Section 3.1 and Section 3.2.

2.4.1 Cluster Parameter Experiments

The key parameters of the multiple-level clustering approach are the number of clustering levels (L) and the cluster sizes at each level (s_1 ... s_L). We first explored a single level of clustering - L = 1. To determine the cluster size value (s_1) that provides the best quality-time trade-off, we ran the tool on a set of benchmark circuits and varied the cluster size from 4 to 4096 by powers of 4. For the subsequent iterative improvement placement, many different annealing schedules were run in order to determine the complete quality-time trade-off possibilities. For example, Figure 3 is a plot of the mean normalized placement wirelength (with respect to the best possible placement obtained by VPR [Betz97]) versus the mean run time, across a set of 20 benchmark circuits. In that figure, the clustering size s_1 was set to 64.



Legend: **lev0**: annealing schedule for top level clusters; **flat**: follow-up annealing schedule at flat level; **therm**: automatic anneal using simulated thermometer; **inum**: range of InnerNum values at each level; **auto**: automatic anneal; **quench**: zero temperature anneal; **T0**: manual anneal with specific starting temperature.

Figure 3. Placement quality-time plot (20 circuit average) for ultra-fast placement tool using different combinations of annealing schedules on 1-level, size-64 clustered circuits.

We performed similar experiments and generated the same curve for values of s_1 = 4, 16, 64, 256, 1024 and 4096, and determined that the values of 64 and 16 resulted in the best (lowest) quality-time trade-off curve. Figure 4 shows the comparison of the time-quality curves for each value of s_1, and we chose to use 64 as our 1-level cluster size in the placement parameter experiments that follow. This results in fewer clusters that are larger in size, compared to a 1-level cluster size of 16.

We performed similar studies for L = 2 and 3 levels of clustering. These studies are problematic as there are many more parameters and combinations of annealing schedules to explore: for L = 2, there is the setting of s_1 and s_2; for L = 3, there is the setting of s_1, s_2 and s_3. The experiments show that for L = 2, the values of s_1 = 64 and s_2 = 4 were found to be best, and in a few cases, the quality-run time trade-off was superior to the best of the L = 1 envelope. For L = 3, the values for (s_1,s_2,s_3) of (64,4,4), (64,16,4) and (256,4,4) were all found to behave about the same, but all of these settings yielded results that were no better than those obtained across L = 1 and 2. This may be due to the sizes of the large circuits in our benchmark suite; after 2 levels of clustering, the circuits have already been transformed into a few very large clusters (tens of clusters with 256 total flat logic blocks in each). So, an additional level of clustering does little to further simplify the placement problem, and may even cost both time (because of the extra processing at level 3) and area (an additional level of grid resizing must be performed, which may adversely affect the grid size at the flat level).

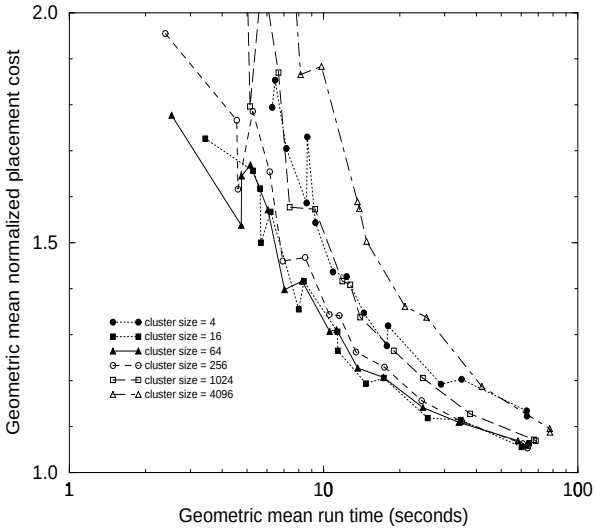
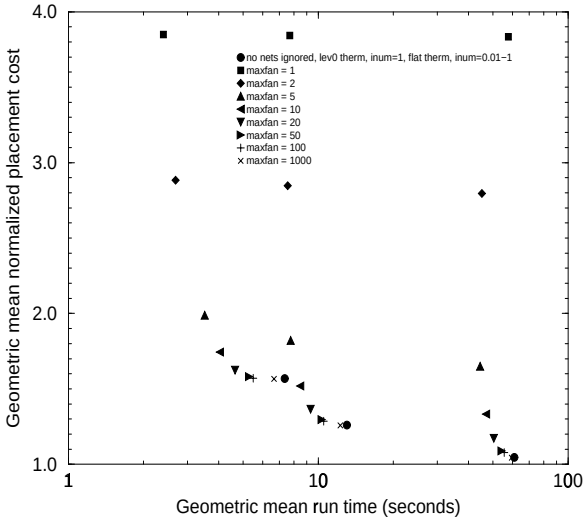


Figure 4. Placement quality-time curves (20 circuit average) for ultra-fast placement tool using a sample of annealing parameters and varying 1-level cluster sizes from 4 to 4096.



Legend: **maxfan**: fanout threshold; **lev0**: annealing schedule for top level clusters; **flat**: follow-up annealing schedule for flat level netlist; **therm**: automatic anneal using simulated thermometer; **inum**: range of InnerNum values at each level.

Figure 5. Placement quality-time plot (20 circuit average) for ultra-fast placement tool using different fanout thresholds above which nets are ignored on circuits with 3 sets of fixed cluster and placement parameters.

2.4.2 Placement Parameter Experiments

The next set of parameters to tune is the set of simulated annealing parameters described in Section 2.3.2. Recall that we settled on the set of 3 types of schedules described in Section 2.3.2: (1) an automatic anneal that uses a simulated thermometer to compute T_0, dynamically calculated values for T_f and α, and variable *InnerNum*; (2) a quench with variable *InnerNum*; (3) a fixed anneal with T_0 = 0.1, T_f = 0.01, α = 0.8, and a variable *InnerNum*. We explored the combinations of these schedules at the clustered and flat levels of the hierarchy, for circuits clustered with L = 1 and s_1 = 64 blocks per cluster. The scatter plot of geometric mean normalized placement cost vs. geometric mean run time is given in Figure 3, and note the complexity of the various combinations of schedules. For short run times, the envelope is comprised of a quench (schedule 2) at the top level with *InnerNum* = 10, and the short, fixed annealing schedule (schedule 3) with *InnerNum* of 0.1 to 0.5. For longer run times, the envelope consists of the automatic anneal (schedule 1) at the top level with *InnerNum* = 1 and the automatic anneal at the flat level with *InnerNum* from 0.2 to 1. In each case, though, it is evident that there are alternative schedules that come reasonably close to providing the same quality-time trade-off as the envelope. Similar combinations of schedules were attempted for 2 and 3-level clustered circuits.

In order to determine the best value of the fanout threshold (the value of fanout above which the nets are ignored), we performed an experiment with L = 1 and s_1 = 64, and varied the fanout threshold. Figure 5 is a scatter plot of quality versus run time for various values of fanout threshold and annealing schedules. The circular dots represent the quality when no nets are ignored, and the other points show the quality when more nets are ignored - from fanout thresholds ranging from 1000 to 1. It is evident that excessively low fanout thresholds eliminate far too much placement information from the circuit, hence the area degradation is huge. However, when nets with fanout over 100 are ignored, we save a few seconds of placement time with almost no degradation in quality.

3. Experimental Results

In this section, we compare the new fast placement tool to an existing and known high-quality placement tool, VPR [Betz97]. We first describe the FPGA architecture used in the experimental comparisons, and the overall CAD flow.

3.1 Target FPGA Architecture

We use an island-style FPGA with a logic block that contains a single 4-LUT and a single D flip-flop. Each block has 6 pins: 4 inputs, 1 output, and 1 clock. We will assume the FPGA has dedicated resources for routing the clock, reset, and other global nets. We assume an I/O pad pitch-to-logic block ratio of 2.

3.2 Benchmark Circuits and CAD Flow

We have collected 20 circuits from a variety of sources: 14 of the largest circuits from the MCNC suite [Yang91], one comes from the RAW suite [Babb97], one is a synthetic circuit generated by GEN [Hutt97], and the remaining four are designs created for the Transmogrifier-2 rapid prototyping system [Lewi97] at the University of Toronto [Ye98] [Hame98] [Leve98]. Each circuit was optimized using SIS [Sent92], and technology mapped into 4-LUTs using Flowmap and Flowpack [Cong94]. VPACK [Betz97] was used to pack the netlists of 4-LUTs and flip-flops into logic blocks. The sizes of the 20 benchmark circuits range from 3000 to 20,000 logic blocks.

We have implemented our fast placement tool within the framework of VPR. We use the bounding box wirelength of all nets in the circuit to compare the quality of placement of each circuit from each tool. We measure only the time used to perform clustering and placement, and do not include the initial input file reading time and parsing (this is no more than 5 seconds for the

largest circuit). All experiments are run on a 300 MHz Sun UltraSPARC workstation.

3.3 Basis of Comparison

We use the pure simulated annealing-based VPR as the basis for comparison to our new placement algorithm. In order to compare the quality-time trade-off curve for VPR, we needed to vary the schedule parameters for VPR itself, in a similar manner to that described above for our tool.

To obtain the envelope of the quality-time curve for VPR, we varied each of the key simulated annealing parameters - initial temperature (T_0), exit temperature (T_f), temperature update factor (α), and scaling factor for the number of moves to attempt per temperature (*InnerNum*). We used the three types of schedules described in Section 2.3.2: (1) an automatic annealing schedule (T_0, T_f, and α calculated dynamically and adjusted depending upon the quality of the placement) with variable *InnerNum*; (2) a quench (greedy heuristic) with variable *InnerNum*; (3) a fixed annealing schedule, where we either sweep T_0, keeping T_f, α, and *InnerNum* constant, or sweep α, keeping T_0, T_f, and *InnerNum* constant.

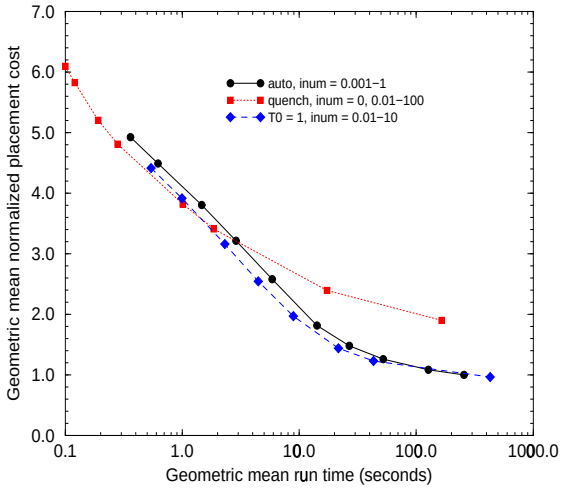
We ran each unique annealing schedule on all 20 circuits, recorded the run time and wirelength, and normalized the wirelength for each run on a given circuit to that achieved by VPR when run under its "-fast" option on that same circuit. This specific VPR option is similar to its default parameters that are tuned to generate high-quality placements, except that one-tenth of the configurations are explored at each temperature. Typically, this increases the placement cost by at most 10%, but with a factor of 10 speedup in placement time. Essentially, it is a very high quality placement that is obtained in a reasonable amount of time. It is from these experiments that we determined the envelope of the best VPR annealer parameters to specify across all 20 circuits.

The envelope containing the annealing schedules that produced the best quality-time trade-off consisted of parts of 3 types of schedules with variable *InnerNum*: a quench, an anneal with T_0 = 1, T_f = 0.01, and α = 0.8, and an automatic anneal with dynamically-updated T_0, T_f, and α. Figure 6 illustrates the geometric mean normalized placement cost (bounding-box wirelength) versus geometric mean run time across all 20 of our benchmark circuits for the 3 schedules that form the quality-time envelope for VPR.

There is not much difference in wirelength and run time among the schedules for extremely short run times (< 3 sec). We observe that, for run times in the 10 to 100 second range, there is ample room for improvement; an average of 80-100% extra wiring area is likely unacceptable to a circuit designer even within 10 seconds of placement time.

3.4 Comparisons Between New Algorithm and VPR

A head-to-head comparison between the ultra-fast placement tool and VPR is possible by running each set of placement parameters that lies on the envelope of the respective tool on every circuit in the benchmark suite, normalizing the placement quality results to those obtained by running VPR under its "-fast" option, and calculating the geometric mean placement cost and run time. Figure 7 is a plot of both the best VPR quality-time envelope and the new ultra-fast placement tool quality-time envelope. Each point



Legend: **auto**: automatic annealing schedule; **quench**: zero temperature anneal; **T0**: manual anneal with starting temperature = 1; **inum**: range of InnerNum values.

Figure 6. VPR placement quality-time trade-off (20 circuit average) using only those annealing schedules that form the envelope.

on the fast placement envelope refers to a unique set of clustering and annealing parameters. It shows that the ultra-fast placement tool has a clear advantage for both short run times (10 seconds or less) and medium run times (from 10 to 100 seconds). In 10 seconds, our placement tool requires only 30% more wirelength on average (than the best possible placement), while VPR requires at least 80% more wirelength on average. Furthermore, while VPR can achieve an average area penalty of 10% in over 100 seconds, our placement tool can attain this level in approximately 30 seconds. If we allowed our placement tool to run without a compile time restriction, it would produce placements that would be very nearly what VPR can achieve, since both tools are based on similar implementations of simulated annealing. This is apparent from the plot: within 60 seconds on average, the ultra-fast placement tool yields an average wirelength that is within 5% of VPR's high-quality anneal. Figure 7 also demonstrates that by manipulating the fast placement tool parameters, we can realize a smooth trade-off between placement quality and execution time.

Table 1 provides a comparison between VPR and the ultra-fast placement tool with one particular set of parameters: L= 2 levels of clustering with cluster sizes s_1 = 64 and s_2 = 4, with the top-level and level-1 annealing schedules being a quench (*InnerNum* = 10), a flat anneal with T_0 = 1 (*InnerNum* = 0.5), and nets above fanout=100 ignored. The geometric mean run time across all circuits for this set of parameters is 11.37 seconds, and the geometric mean area penalty is 22%. It is difficult to find directly comparable run times between the two tools; we then select a schedule from the VPR envelope that is as close as possible. The first column of Table 1 gives the circuit name, its size in number of logic blocks, the run time and normalized placement cost obtained using our fast tool, and the comparable data using VPR. It shows that the ultra-fast placement algorithm wins in a comparison with VPR for every circuit in our suite, posting a superior wirelength in a significantly shorter run time. Note that for this particular set of ultra-fast placement parameters, the reduction in wirelength compared to VPR ranges from 13% to 50%.

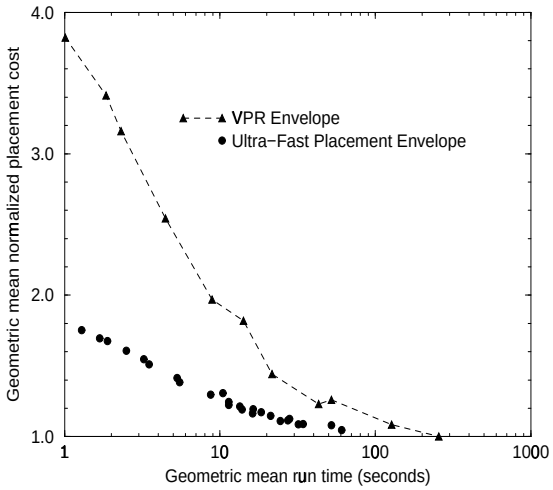


Figure 7. Placement quality-time envelope curves (20 circuit average) for VPR and new ultra-fast placement tool.

The true measure of quality of a given placement is whether or not it can be successfully routed on the target FPGA. Although we have not attempted to route any of the ultra-fast placements, [Swar98b] has shown that wirelength and routability correlate extremely well. Therefore, we are satisfied that our ultra-fast placements are superior to those produced by VPR, based solely on wirelength for the range of compile times of interest.

3.5 Wirelength Estimation and Accuracy

One way to use a fast placement tool, even if the user is not interested in sacrificing any final circuit quality, is to use it as a routability estimator for a given netlist. Swartz et al. [Swar98a] show how to predict if a circuit will route on a given FPGA, given the wirelength of the placement of a circuit and the number of tracks per channel in the target FPGA. The drawback of their approach is that the placement must be known beforehand. We propose that our fast placement algorithm be used to obtain very fast and accurate estimates of the final *best* placement wirelength. The idea is that we can run the fast placement tool in one of its very fastest modes, measure the wirelength of that placement, and then decrease the wirelength by the typical amount that the fast mode is usually worse than the best mode. The quality of the result depends on the consistency of difference in wirelength between the fast mode and the best mode. This can be measured by determining how much the normalized placement cost for each circuit, in the fast mode, varies from the mean normalized placement cost across all circuits.

Figure 8 is a plot of the average difference of each circuit's normalized wirelength from the mean over all circuits versus different run times of the ultra-fast placement tool obtained from the quality-time envelope parameters. (To obtain this graph, we calculate the absolute difference between the geometric mean normalized placement cost and the actual normalized placement cost for each of the 20 circuits for each particular set of fast placement parameters. We then compute the arithmetic mean of these differences (and call it mean absolute error) and plot it versus the geometric mean run time that was obtained for the set of circuits for this set of parameters.)

Circuit	# Logic Blocks	Ultra-Fast Placement		VPR	
		Run Time (s)	Norm. Place Cost	Run Time (s)	Norm. Place Cost
clma	8383	21.71	1.20	29.79	1.83
spla	3690	6.37	1.22	7.26	1.63
s38584.1	6447	14.55	1.29	18.35	2.33
s38417	6406	13.33	1.22	16.88	1.87
frisc	3556	6.15	1.21	7.04	1.63
pdc	4575	8.43	1.20	10.35	1.52
ex1010	4598	7.69	1.23	10.53	1.67
elliptic	3604	6.05	1.15	7.16	1.60
beast20k	19600	108.34	1.16	128.10	1.34
bubble sort	12293	41.08	1.29	53.65	2.14
fir16	6975	16.31	1.32	20.86	2.19
iir16	3739	6.93	1.15	7.57	2.16
mac64	4307	8.94	1.19	10.19	1.69
ochip64	4083	6.85	1.13	10.63	2.30
ralu32	3662	5.96	1.25	6.69	1.66
spsdes	3363	5.22	1.21	6.47	1.70
des_fm	4786	9.25	1.34	13.25	1.69
des_sis	5351	11.12	1.24	14.14	1.67
wood	7432	17.54	1.24	22.15	2.00
marb	5535	11.61	1.26	13.72	2.15
Geometric Average		11.37	1.22	14.17	1.82

Table 1: Comparison between ultra-fast placement tool and VPR for 20 circuits. One set of placement parameters was employed for each tool such that their run times were close and they formed the quality-time envelope for their respective tools.

Figure 8 shows that, as we would expect, longer compile times produce more accurate wirelength estimates. Impressively, even short run times result in accurate estimates - for example, an average run time of just over 10 seconds results in a mean absolute error of less than 5%.

We can therefore use the fast-placement run time as an accurate estimator of the final best wirelength. Table 2 illustrates an example of fast wirelength estimation for each of the circuits in our benchmark suite. We used the same set of ultra-fast placement parameters as that used to generate the data in Table 1, and

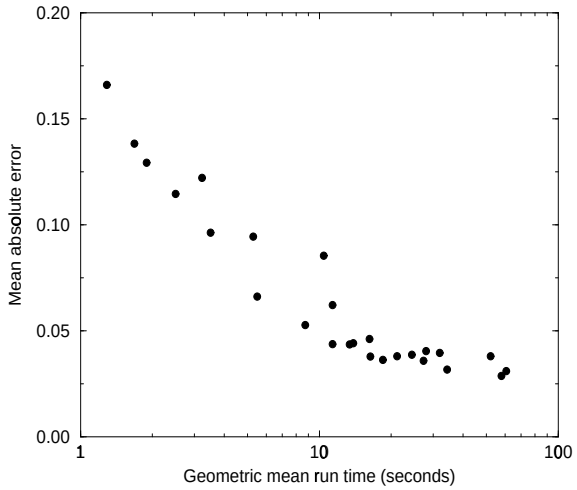


Figure 8. Mean absolute difference in wirelength (between mean wirelength and individual circuit results) vs. mean run time for parameters forming ultra-fast placement tool envelope.

recorded both the run times and raw wirelength result in each case. From the envelope curve in Figure 7, we know the mean normalized wirelength for this set of parameters across all circuits to be 1.22, or 22% larger than the highest-quality wirelength attainable by VPR. The mean run time is 11.37 seconds. Figure 8 indicates that the mean absolute error for that set of parameters is 0.044 (4.4%). Our pessimistic prediction of high quality wirelength can be written as:

$$Wirelength_{predicted} = \beta \cdot Wirelength_{ultra\text{-}fast}$$
$$\text{where} \quad \beta = 1 \Big/ \left(\overline{Wirelength_{normalized}} - \overline{Absolute\ Error} \right) \tag{2}$$

The predicted high-quality wirelength for a given circuit is expressed as being proportional to the wirelength obtained from the ultra-fast placement tool. The scaling factor, β, is composed of the difference between the normalized wirelength for the specific set of ultra-fast placement parameters chosen (geometrically averaged over all circuits) and the previously described mean absolute error (arithmetic average over all circuits of the absolute differences between the mean normalized wirelength and the actual normalized wirelengths) for the same set of placement parameters. The scaling factor denotes by what fraction the fast mode wirelength should be reduced to obtain a pessimistic estimate of the best mode wirelength.

So for the case in Table 2, the formula reduces to $Wirelength_{predicted} = Wirelength_{ultra\text{-}fast} / (1.22\text{-}0.044)$. We employ this to compute a wirelength estimate for each circuit based on the fast placement wirelength result, and compare it to the known high-quality wirelength for each circuit from VPR. For 16 of the circuits, our pessimistic estimate is between 0.89% and 13.75% higher than the actual high-quality wirelength, and in only two cases is the error greater than 10%. In four cases, the estimator was not pessimistic enough, predicting a wirelength that was between 1.71% and 3.93% less than the actual high-quality wirelength. Overall, the average absolute error of the wirelength estimator was under 5% for the set of placement parameters that yielded a mean run time of just over 11 seconds.

Circuit	Run Time (s)	Ultra-Fast Wire-length	Pred. High-Quality Wire-length	VPR High-Quality Wire-length	% Error
clma	21.71	1786	1514	1491	+1.55
spla	6.37	763	646	625	+3.37
s38584.1	14.55	901	763	696	+9.70
s38417	13.33	883	748	726	+3.12
frisc	6.15	685	580	566	+2.58
pdc	8.43	1096	929	917	+1.35
ex1010	7.69	843	715	688	+3.84
elliptic	6.05	588	499	513	-2.75
beast20k	108.34	7522	6374	6485	-1.71
bubble sort	41.08	1632	1383	1262	+9.57
fir16	16.31	1108	939	841	+11.62
iir16	6.93	464	393	404	-2.63
mac64	8.94	660	560	555	+0.89
ochip64	6.85	350	297	309	-3.93
ralu32	5.96	506	429	405	+5.85
spsdes	5.22	527	447	434	+2.88
des_fm	9.25	857	727	639	+13.75
des_sis	11.12	826	700	665	+5.33
wood	17.54	1085	920	873	+5.31
marb	11.61	617	523	492	+6.39
Arithmetic Average absolute error					4.91

Table 2: Example of quality of wirelength prediction capability of ultra-fast placement tool.

4. Conclusions and Future Work

We have demonstrated that an ultra-fast placement algorithm based on multiple-level clustering, constructive placement, and simulated-annealing-based refinement works very well in relation to an existing high-quality pure simulated annealing placement tool. It provides superior area results across an entire set of large circuits compared to VPR when both tools are instructed to take approximately the same amount of time to formulate a placement. For example, in 10 seconds on a 300 MHz Sun UltraSPARC, our ultra-fast tool can achieve an average area penalty of 30%, while VPR can manage no better than 80%. The algorithm uses several key clustering and placement parameters to permit the user to

smoothly trade quality of placement for compile time. We explored the vast space covered by these parameters to find the fast tool's best quality-time envelope and showed that its envelope is significantly better than that possible with the pure simulated annealing formulation of VPR.

If we have no compile-time restrictions, then our algorithm produces placements that approach the same quality as VPR. We also showed that the fast placement tool can be used as a fast estimator of high-quality wirelength, with a mean absolute error of less than 5%, in an average run time of less than 11.5 seconds.

In the future, it would be useful to explore a fast quadratic-programming-based placement algorithm or one based on top-down mincut partitioning, and determine their quality-time trade-off relationships. Another interesting area to pursue is the refinement and integration of the fast wirelength estimator with the difficulty predictor provided by an existing fast router [Swar98b]. Finally, a timing-driven fast placement tool should also be developed.

5. Acknowledgments

We gratefully acknowledge the contributions of Dr. Vaughn Betz to this work, not only for providing the infrastructure and support of VPR that was used to house the new algorithm, but also for the advice and guidance he offered throughout. We thank Paul Leventis for his translation from EDIF to BLIF of the large benchmark circuits created and kindly donated by Andy Ye and Ivan Hamer. We also thank Dr. Steve Wilton and Dr. Russ Tessier for providing some of the large benchmark circuits that were used here. This work was supported by funding from Lucent Technologies, MICRONET, and NSERC.

6. References

[Babb97] J. Babb et al., "The RAW Benchmark Suite: Computation Structures for General Purpose Computing," *FCCM*, 1997, pp. 161-171.

[Betz97] V. Betz and J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research," *Proc. Intl. Workshop on FPL*, 1997, pp. 213-222.

[Betz98] V.Betz, "Architecture and CAD for Speed and Area Optimization of FPGAs," *Ph.D. Thesis*, University of Toronto, 1998.

[Brow92] S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic, *Field-Programmable Gate Arrays*, Kluwer Academic Publishers, 1992.

[Call98] T. J. Callahan, P. Chong, A. DeHon, and J. Wawrzynek, "Fast Module Mapping and Placement for Datapaths in FPGAs," *FPGA*, 1998, pp. 123-132.

[Cong94] J. Cong and Y. Ding, "Flowmap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs," *IEEE Trans. on CAD*, Jan. 1994, pp. 1-12.

[Dunl85] A. E. Dunlop and B. W. Kernighan, "A Procedure for Placement of Standard-Cell VLSI Circuits," *IEEE Trans. on CAD*, vol. 4, no. 1, Jan. 1985, pp. 92-98.

[Gehr98] S. Gehring and S. Ludwig, "Fast Integrated Tools for Circuit Design with FPGAs," *FPGA*, 1998, pp. 133-139.

[Hame98] I. Hamer, "Implementation of DES on Transmogrifier-2a," *Personal Communication*, 1998.

[Hana72] M. Hanan and J. M. Kurtzberg, "Placement Techniques," in *Design Automation of Digital Systems, Volume 1: Theory and Techniques*, M. A. Breuer, Ed., Prentice-Hall, 1972, pp. 213-281.

[Hutt97] M. Hutton, J. Rose, and D. Corneil, "Generation of Synthetic Sequential Benchmark Circuits," *FPGA*, 1997, pp. 149-155.

[Kirk83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, May 13, 1983, pp. 671-680.

[Klei91] J. M. Kleinhans, G. Sigl, F. M. Johannes, and K. J. Antreich, "GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization," *IEEE Trans. on CAD*, vol. 10, no. 3, Mar. 1991, pp. 356-365.

[Lewi97] D. M. Lewis, D. R. Galloway, M. van Ierssel, J. Rose, and P. Chow, "The Transmogrifier-2: A 1 Million Gate Rapid Prototyping System," *FPGA*, 1997, pp. 53-61.

[Leve98] P. Leventis, "Using edif2blif Version 1.0," University of Toronto, 1998. (*Available for download from http://www.eecg.toronto.edu/~leventi/edif2blif/edif2blif.html*).

[Roy93] K. Roy and C. Sechen, "A Timing Driven N-Way Chip and Multi-Chip Partitioner," *ICCAD*, 1993, pp. 240-247.

[Rose90] J. Rose, W. Klebsch, and J. Wolf, "Temperature Measurement and Equilibrium Dynamics of Simulated Annealing Placements," *IEEE Trans. on CAD*, vol. 9, no. 3, Mar. 1990, pp. 253-259.

[Sank99] Y. Sankar, "Ultra-Fast Automatic Placement for FPGAs," *M.A.Sc. Thesis*, University of Toronto, in preparation, 1999.

[Sech85] C. Sechen and A. Sangiovanni-Vincentelli, "The TimberWolf Placement and Routing Package," *IEEE Journal of Solid-State Circuits*, vol. 20, no. 2, Apr. 1985, pp. 510-522.

[Sech88] C. Sechen, *VLSI Placement and Global Routing Using Simulated Annealing*, Kluwer Academic Publishers, 1988.

[Sent92] E. M. Sentovich et al., "SIS: A System for Sequential Circuit Analysis," *Tech. Report No. UCB/ERL M92/41*, University of California, Berkeley, 1992.

[Shah91] K. Shahookar and P. Mazumder, "VLSI Cell Placement Techniques," *ACM Computing Surveys*, vol. 23, no. 2, Jun. 1991, pp. 143-220.

[Sun95] W. Sun and C. Sechen, "Efficient and Effective Placement for Very Large Circuits," *IEEE Trans. on CAD*, vol. 14, no. 3, Mar. 1995, pp. 349-359.

[Swar98a] J. S. Swartz, V. Betz, and J. Rose, "A Fast Routability-Driven Router for FPGAs," *FPGA*, 1998, pp. 140-149.

[Swar98b] J. S. Swartz, "A High-Speed Timing-Aware Router for FPGAs," *M.A.Sc. Thesis*, University of Toronto, 1998.

[Tess98] R. Tessier, "Fast Place and Route Approaches for FPGAs," *Ph.D. Thesis*, MIT, 1998.

[Yang91] S. Yang, "Logic Synthesis and Optimization Benchmarks, Version 3.0," *Tech. Report*, Microelectronics Centre of North Carolina, 1991.

[Ye99] A. Ye, "Procedural Texture Mapping on FPGAs," *M.A.Sc. Thesis*, University of Toronto, in preparation, 1999.