An Architecture-Independent Approach to FPGA Routing Based on Multi-Weighted Graphs *

Michael J. Alexander, James P. Cohoon, Joseph L. Ganley, and Gabriel Robins

Department of Computer Science, University of Virginia, Charlottesville, VA 22903-2442

Abstract

We propose a general framework for FPGA routing, which allows simultaneous optimization of multiple competing objectives under a smooth designer-controlled tradeoff. Our approach is based on a new multi-weighted graph formulation, enabling a theoretical performance characterization, as well as a practical implementation. Our FPGA router is architecture-independent, computationally efficient, and performs well on industrial benchmarks.

1 Introduction

Field-programmable gate arrays (FPGAs) are an inexpensive and flexible design alternative to custom integrated circuits. FPGAs are reusable high-density ASICs that can be easily (re)configured by the user, which has made them a popular "low risk" way to implement digital designs [29] [32]. Although there are a number of different commercially available FPGA technologies, an FPGA architecture generally consists of a symmetrical array of user configurable logic "blocks" or "cells" (each of which implements a portion of the design logic) and a set of interconnection resources used for routing [8].

Partitioning and technology mapping in FPGAs has been extensively studied by e.g. [10] [15] [21] [26], where a typical goal is to minimize the maximum input-to-output circuit depth (which reduces delay) by varying the total number of logic blocks used (which in turn affects placement and routing feasibility), or some tradeoff between these two goals [27]. More recent work has addressed the issue of FPGA routability prediction during higher levels of the design cycle [6] [9] [28]. Routability is the likelihood of a particular placement being feasible to route using the available interconnect resources.

While technology mapping and routability have been studied extensively, less attention has been focused on the actual routing. This is surprising, since it has been noted that feasibility in FPGA designs is constrained by routing resources more than by logic resources [28]. Moreover it was observed that FPGA performance is often limited by routing delays, rather than by logic-block delays [6]. Much previous work centered around the CGE and SEGA routers [7] [8] [23], which use a global router [25] to select a sequence of candidate channel edges for each connection. In the technology mapping research of [6], Steiner routing is performed by a global router; other work has adopted a more abstract model of FPGA routing [24], or explored issues such as bend reduction [30].

We propose the first unified general framework for FPGA routing, where multiple competing objectives can be optimized simultaneously under a smooth designer-controlled tradeoff. Our architecture-independent approach is based on a new and general multi-weighted graph formulation and escapes the pitfalls of the conventional global/local routing dichotomy by offering a single unified and effective method. Our techniques may also be extended to address high-performance FPGA routing [2].

2 A Typical FPGA Architecture

A typical *symmetrical-array* FPGA [8] consists of a rectangular array of logic blocks, separated by channels containing routing resources (i.e., channel edges, connection edges and switchboxes), as illustrated in Figure 1. Logic blocks can be software-(re)programmed to implement arbitrary logic functions; the input and output pins of these logic blocks may then be connected using the software-(re)programmable routing resources.

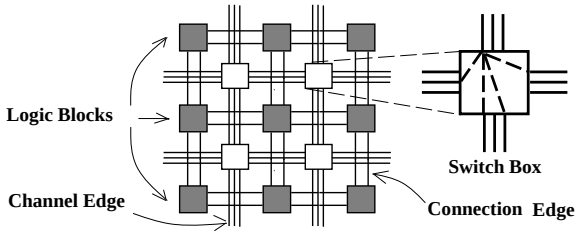

Figure 1: A symmetrical-array FPGA.

Each switchbox contains internal edges which can be programmed to connect channel edges on different sides of the switchbox, allowing routing paths to pass through the switchbox. A switchbox allows interconnections from a given channel edge to only a subset of the channel edges on the other three sides and this interconnection pattern need not be symmetric [8]. Connection edges are used to connect logic-block pins to channel edges. The FPGA may be modeled as a graph, where the overall graph topology mirrors the complete FPGA architecture; paths in this graph correspond to feasible routes on the FPGA, and conversely (See Figure 2).

*Corresponding author is Professor Gabriel Robins, Department of Computer Science, Thornton Hall, University of Virginia, Charlottesville, VA 22903-2442, Email: robins@cs.virginia.edu, phone: (804) 982-2207, FAX: (804) 982-2214. Professor Robins is partially supported by NSF Young Investigator Award MIP-9457412. Professor Cohoon is partially supported by NSF grants MIP-9107717 and CDA-8922545.

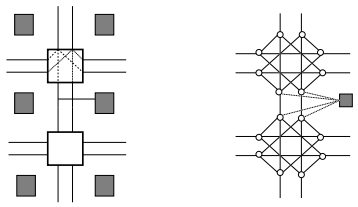

Figure 2: Construction of a routing graph for modeling symmetrical-array FPGAs.

3 FPGA Routing

Our work addresses the routing phase of FPGA design, where the appropriate sets of logic-block pins (i.e., *nets*) must be interconnected. We assume that partitioning, technology mapping, and placement have already been performed. The FPGA routing problem is therefore defined as follows:

The FPGA Routing Problem: Given an FPGA architecture along with a configuration of logic-block pin assignments and a collection of nets over these pins, route all the nets without exceeding the total available FPGA routing resources.

In traditional VLSI routing regimes, wires can be placed in any available space within the routing region; therefore, the main considerations in traditional routing are usually *geometrical* in nature (i.e., minimizing wirelength, avoiding collisions with other nets, compacting the overall layout, etc.). In contrast, an FPGA architecture offers a limited discrete set of fixed routing channels, and the utilization of a given channel edge in one route excludes its usage in the routing of other nets; thus, the FPGA routing problem is essentially *combinatorial* in nature (i.e., satisfying the allowable connectivity, maintaining feasibility, etc.). A purely topological approach to FPGA routing may fail to exploit certain available geometrical information, such as physical proximity of pins/blocks, distance from the FPGA "border", etc. On the other hand, a strictly geometric approach may fail to take into account the topological/combinatorial constraints induced by the discrete/limited nature of the available FPGA routing resources.

With this in mind, we propose a hybrid FPGA routing framework that combines both geometric and combinatorial techniques, and which simultaneously enjoys the attractive properties of each while retaining computational efficiency. In particular, we hybridize (i) the Iterated 1-Steiner routing method of [20] (which is known to have both excellent empirical performance as well as an efficient implementation [4]) with (ii) the provably-good graph Steiner approximation scheme of [22], which can also be implemented efficiently [31]. We show that the resulting combination of these two methods inherits the best characteristics of its two component methods.

3.1 Overview of Iterated 1-Steiner

The *cost* of an edge between two points in the Manhattan plane is the rectilinear distance between them. A *spanning tree* over a pointset P is a connected graph containing $|P|-1$ edges. The cost of a tree T, denoted $\overline{\mathrm{T}}$, is the sum of the costs of its edges. A *minimum spanning tree* (MST) is a spanning tree having least cost. A *Steiner tree* is a spanning tree over the original pointset P and a (possibly empty) additional pointset S (i.e., the *Steiner points*). We are now ready to define the minimum rectilinear Steiner tree problem:

Minimum Rectilinear Steiner Tree (MRST) problem: Given a set P of n points in the Manhattan plane, find a set S of Steiner points such that the MST over $P \cup S$ has minimum cost.

Research on the MRST problem has been guided by several fundamental results. First, Hanan [17] has shown that there always exists an MRST with Steiner points chosen from the intersection of all the horizontal and vertical lines passing through all the points in P. A second major result established that despite this restriction on the solution space, the MRST problem remains NP-complete [16], prompting a large number of heuristics, as surveyed in [19].

The best known MRST heuristic is the Iterated 1-Steiner (I1S) algorithm [20], which always performs strictly better than $\frac{3}{2}$ times optimal and achieves almost 11% average improvement over MST cost. Moreover, for typical nets, I1S has average performance within 0.25% of optimal and produces optimal solutions up to 90% of the time [4] [5].

For two pointsets P and S, define the MST savings of S with respect to P as $\Delta\overline{\mathrm{MST}}(P,S) = \overline{\mathrm{MST}}(P) - \overline{\mathrm{MST}}(P \cup S)$. We use $H(P)$ to denote the set of Hanan Steiner point candidates (i.e., the intersections of all horizontal and vertical lines passing through points of P). For a pointset P, a 1-*Steiner point* $x \in H(P)$ maximizes $\Delta\overline{\mathrm{MST}}(P,\{x\}) > 0$. The I1S method repeatedly finds 1-Steiner points and includes them in S. The cost of the MST over $P \cup S$ will decrease with each added point, and the construction terminates when there is no x with $\Delta\overline{\mathrm{MST}}(P \cup S,\{x\}) > 0$. Figure 3 describes the algorithm formally.

3.2 Overview of the KMB Method

Often one encounters the graph version of the Steiner problem, which is embedded in a given graph $G = (V, E)$, where V is the node set and $E \subseteq V \times V$ is a set of weighted edges. Here we are asked to span a subset of the nodes $N \subseteq V$, while using the remaining nodes as Steiner points. Each edge $e_{ij} \in E$ has weight w_{ij}, and our goal is to minimize the total spanning cost. Figure 4 shows a graph and a Steiner minimum tree spanning the highlighted subset of the nodes; the graph Steiner problem is defined as follows:

The Iterated 1-Steiner (I1S) Algorithm [20]

Input: A set P of n points
Output: A rectilinear Steiner tree over P

$S = \emptyset$
While $T = \{x \in H(P) | \Delta\overline{\text{MST}}(P \cup S, \{x\}) > 0\} \neq \emptyset$ **Do**
  Find $x \in T$ with maximum $\Delta\overline{\text{MST}}(P \cup S, \{x\})$
  $S = S \cup \{x\}$
Output $\text{MST}(P \cup S)$

Figure 3: The Iterated 1-Steiner (I1S) algorithm.

The Graph Steiner Minimum Tree (GSMT) problem: Given a weighted graph $G = (V, E)$ and a net of terminals $N \subseteq V$ to connect, find a tree $T' = (V', E')$ with $N \subseteq V' \subseteq V$ and $E' \subseteq E$ such that $\sum_{e_{ij} \in E'} w_{ij}$ is minimized.

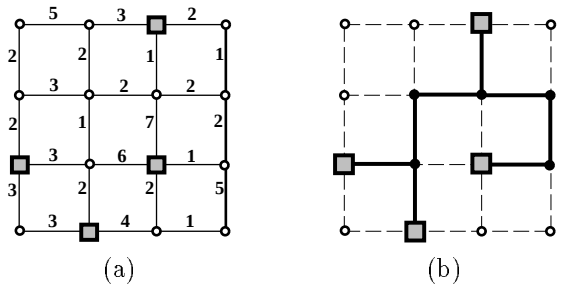

Figure 4: A graph (a) and a Steiner minimum tree (b) spanning the highlighted nodes.

The GSMT problem is NP-complete [19], and thus in order to remain computationally efficient, we must resort to heuristic solutions. The algorithm of Kou, Markowsky and Berman [22] solves the GSMT problem in polynomial time and is guaranteed to yield solutions never more than $2 \cdot (1 - \frac{1}{L})$ times optimal, where L is the minimum number of leaves in any optimal solution. We refer to this as the KMB algorithm: first, construct the complete graph G' over N with the weight of each edge e_{ij} equal to $\text{dist}_G(N_i, N_j)$, the cost of the corresponding shortest path in G between N_i and N_j. Second, compute $\text{MST}(G')$, the minimum spanning tree of G', and expand each edge e_{ij} of $\text{MST}(G')$ into the corresponding shortest path, denoted $\text{path}_G(N_i, N_j)$, yielding a subgraph G'' that spans N. Finally, compute $\text{MST}(G'')$, and delete pendant edges from $\text{MST}(G'')$ until all leaves are members of N. We denote the resulting tree as KMB and the cost of this tree as $\overline{\text{KMB}}$. A formal description of the KMB method is given in Figure 5.

The KMB Algorithm [22]

Input: A graph $G = (V, E)$ with edge weights w_{ij} and a net $N \subseteq V$
Output: A low-cost tree $T' = (V', E')$ spanning N (i.e. $N \subseteq V' \subseteq V$ and $E' \subseteq E$)

$G' = (N, N \times N)$, with edge weights $w'_{ij} = \text{dist}_G(N_i, N_j)$
Compute $T = (N, E'') = \text{MST}(G')$
Let the graph $G'' = \cup_{e_{ij} \in E''} \text{path}_G(N_i, N_j)$
Compute $T' = \text{MST}(G'')$
Delete from T' all leaf nodes that are not in N
Output T'

Figure 5: The KMB heuristic for the GSMT problem.

3.3 A New FPGA Routing Algorithm

Our approach to FPGA routing is based on combining the geometric I1S heuristic with the graph-based KMB algorithm. This strategy allows us to model the FPGA architecture in a very natural and general way (i.e., as a graph), as well as to exploit geometrical information about the physical layout of the FPGA in order to yield improved routing solutions. Indeed the resulting hybrid method inherits the excellent average empirical behavior of I1S, as well as the provably-good theoretical performance of KMB. We refer to this hybrid method as the *Iterated-KMB* (IKMB) algorithm. Note that the present discussion assumes that the graph edges each have a single weight. In Section 4 we explain how to extend our routing methodology to multi-weighted graphs.

Our overall IKMB method is an adaptation of I1S to graphs; but when we need to span a subset N of the nodes in a graph, the notion of "MST" is no longer well-defined. In essence, a "spanning tree" for N is now actually a "Steiner" tree, and can no longer be computed efficiently (since this is NP-complete and is what we are trying to compute in the first place). The key to this dilemma is to replace the "MST" construction with the KMB construction. In other words, instead of using an "MST" subroutine to determine the "savings" of a candidate Steiner point/node, we use the KMB algorithm for this purpose. Thus, given a graph $G = (V, E)$, the net $N \subseteq V$, and a set S of potential Steiner points, we define the following:

$$\Delta\overline{\text{KMB}}_G(N, S) = \overline{\text{KMB}}_G(N) - \overline{\text{KMB}}_G(N \cup S)$$

The IKMB algorithm starts by computing the KMB tree. Then, at each iteration the IKMB method repeatedly finds additional Steiner node candidates that reduce the overall KMB cost and includes them in the growing set of Steiner nodes S. The cost of the KMB tree over $N \cup S$ will decrease with each added node, and the construction terminates when there is no $x \in V$ with $\Delta\overline{\text{KMB}}(N \cup S, \{x\}) > 0$. The overall IKMB method is formally described in Figure 6.

Since an optimal Steiner tree for a 3-pin net in a

graph can have at most one Steiner point with degree > 2 , our IKMB algorithm, which selects the best Steiner point candidate, is guaranteed to find the optimal solution for any 3-pin net. In general, given a weighted graph, and a single arbitrary net, we can show the following:

Theorem 3.1 *The IKMB algorithm will find a routing solution with cost less than* $2\cdot(1-\frac{1}{L})$ *times optimal, where* L *is the minimum number of leaves in of any optimal solution.* □

We can bound the time and space that IKMB requires to route a single n-pin net by $O(|G|\cdot n^3 \log n)$ and $O(n\cdot|G|)$, respectively, where $|G|$ is the size of the routing graph. We note that a method similar to IKMB has recently been applied to "escape graphs" to address routing in the presence of obstacles [13].

The Iterated-KMB (IKMB) Algorithm
Input: A weighted graph $G=(V,E)$ and net $N \subseteq V$ **Output:** A low-cost tree spanning N
$S = \emptyset$ **While** $C = \{x \in V - N \mid \Delta\overline{\mathrm{KMB}}_G(N \cup S, \{x\}) > 0\} \neq \emptyset$ **Do Find** $x \in C$ with maximum $\Delta\overline{\mathrm{KMB}}_G(N \cup S, \{x\})$ $S = S \cup \{x\}$ **Return** $\mathrm{KMB}_G(N \cup S)$

Figure 6: Iterated-KMB algorithm (IKMB).

3.4 Routing Multiple Nets

We use the notion of *congestion* as a measure of resource utilization. Clearly as congestion increases, future routing feasibility decreases, with routing becoming altogether impossqible when congestion reaches a high enough value. Rather than risk formation of concentrated congestion, we prefer to "spread" the congestion around, so as to retain the routing feasibility of future nets. Initially, congestion values on all edges are zero; after a net has been successfully routed, the graph is updated by increasing the congestion values of nearby edges, and the edges used to route the net are then removed from the graph. We use a move-to-front rule when infeasibility is encountered, although rip-up and re-route can also be used. After an attempt has been made to route all nets, those nets which could not be routed are given a higher priority by moving them to the front of the net-routing order. The routing graph is then re-initialized and the new routing order is tried.

4 Multi-Objective Optimization

Recall that ideally we wish to simultaneously optimize k multiple (possibly competing) objectives (i.e., wirelength, jogs, congestion, etc.). We now show how to accomplish this, by generalizing the IKMB heuristic to multi-weighted graphs, where each optimization criterion has a separate set of edge weights. The simultaneous optimization is accomplished by transforming these multiple edge weights into a single weighted average, which is then used by IKMB in the normal way. The relative magnitudes of the weighing factors $d_1, d_2, \cdots, d_k$ (i.e., tradeoff parameters) are designer controlled, which enables a smooth tradeoff among the various competing objectives.

This technique is flexible in that new criteria are easily incorporated into the model by introducing additional weight sets into the graph. Such a framework subsumes e.g., "alpha-beta" routing (which has been used for jog minimization in IC design [11] [18]), and also has practical application in non-VLSI domains [12].

Let $V = \{v_1, v_2, \cdots, v_n\}$ be a set of nodes, and let $E \subseteq V \times V$ be a set of edges. We define a k-weighted graph $G = (V, E)$ to be a weighted graph with a vector-valued weight function $\vec{w} : E \rightarrow \Re^k$. In other words, associated with each edge $e_{ij} \in E$ is a vector of k real-valued weights $\vec{w}_{ij} = (w_{ij1}, w_{ij2}, \cdots, w_{ijk})$. Note that ordinary weighted graphs are a special case of k-weighted graphs, with $k = 1$.

Let $\vec{d} = (d_1, d_2, \cdots, d_k)$ be a vector of k real-valued *tradeoff parameters*, where $0 \leq d_i \leq 1$ for $0 \leq i \leq k$, and $\sum_{i=1}^{k} d_i = 1$. From the k-weighted graph $G = (V, E)$ and the tradeoff parameters $\vec{d}$ we construct a new weighted *tradeoff graph* $\widehat{G}(\vec{d}) = (V, E)$ with weight function $w'_{ij} = \vec{d} \cdot \vec{w}_{ij} = \sum_{m=1}^{k} d_m \cdot w_{ijm}$. The tradeoff graph $\widehat{G}$ is an ordinary weighted graph having the same topology as G, but whose single edge weights represent the weighted averages of the multi-weights of G, with respect to $\vec{d}$.

Let $\vec{u} = (1, \ldots, 1)$, and $\vec{v}_i = (0, \cdots, 0, v_i, 0, \cdots, 0)$ denote the vector obtained from the vector $\vec{v}$ by using v_i in the i-th place, and the rest of the places being set to zero. Thus, $\vec{u}_i$ denotes the vector consisting of zeros everywhere except the i-th place, which will contain a 1. A k-weighted graph G induces k distinct graphs $G_i = \widehat{G}(\vec{u}_i)$, each with an identical topology but with edge weights restricted to only one of the k components of vector-valued weight function $\vec{w}$.

We define the minimum spanning tree for a multi-weighted graph G with respect to the tradeoff parameters $\vec{d}$ as the "normal" MST over the tradeoff graph $\widehat{G}(\vec{d})$, and denote it by $\mathrm{MST}(\widehat{G}(\vec{d}))$. Similarly, we can compute the MST on each of the k induced graphs G_i, and we denote these $\mathrm{MST}(G_i)$.

We have the following bounds (proofs are omitted due to space limitations; see [1] [3] for details).

Theorem 4.1 *For any* k*-weighted graph* G *and tradeoff parameters* $\vec{d}$, $\sum_{i=1}^{k} d_i \cdot \overline{\mathrm{MST}}(G_i) \leq \overline{\mathrm{MST}}(\widehat{G}(\vec{d}))$. □

We can also show the non-existence of general upper bounds:

Theorem 4.2 *For any k-weighted graph G and tradeoff parameters $\vec{d}$, $\overline{\text{MST}}(\widehat{G}(\vec{d}))$ can not be bounded by any function of only $\overline{\text{MST}}(G_i)$'s, $\vec{d}$, n, and k.* □

On the other hand, we can show a general upper bound for *metric* graphs:

Theorem 4.3 *For any metric k-weighted graph G and tradeoff parameters $\vec{d}$, $\overline{\text{MST}}(\widehat{G}(\vec{d})) \leq (n-1) \cdot \sum_{i=1}^{k} d_i \cdot \overline{\text{MST}}(G_i)$* □

Since most nets in typical VLSI designs contain three pins or less [13], we derive a tighter upper bound for 3-pin nets where metricity holds (i.e., graphs with weight functions satisfying the triangle inequality $\text{dist}(a,b) + \text{dist}(b,c) \geq \text{dist}(a,c)$, $\forall a, b, c \in V$):

Theorem 4.4 *For 2-weighted metric graphs with three nodes, and any scaling vector $\vec{d} = (d_1, d_2)$, the following holds: $d_1 \cdot \overline{\text{MST}}(G_1) + d_2 \cdot \overline{\text{MST}}(G_2) \leq \overline{\text{MST}}(\widehat{G}(\vec{d})) \leq \frac{4}{3} \cdot [d_1 \cdot \overline{\text{MST}}(G_1) + d_2 \cdot \overline{\text{MST}}(G_2)]$* □

Recently, tighter bounds on MSTs over multi-weighted graphs were derived [14].

5 Experimental Results

We compared the performance of KMB and IKMB over 10,000 random nets of cardinality 3, 4, 5, 7 and 10, with coordinates uniformly distributed in the range $[0, \ldots, 10000]$. We routed each net in the grid graph induced by the intersection of the vertical and horizontal lines passing through the pins, and compared the wirelength required by each algorithm. The results in Table 1 indicate that IKMB consistently uses less wirelength than KMB.

Average wirelength savings of IKMB w.r.t KMB					
Net Size	3-pin	4-pin	5-pin	7-pin	10-pin
Savings	3.31%	4.64%	5.53%	6.45%	6.72%

Table 1: Avg. wirelength savings of IKMB vs. KMB.

We have also implemented an FPGA router, based on the IKMB algorithm, using C++ in the SUN IPC workstation environment. The code is available from the authors upon request. A common criterion used to evaluate the quality of FPGA routing solutions is the maximum width of the channels (i.e., how many edges wide are they) required to successfully route all nets of a circuit [8]. We have therefore compared the maximum channel width our router required to that required by CGE [8], using identical switchbox and interconnection options.

We have tested our router on the five industrial benchmarks used in [8] (see Table 2). Table 3 compares our router with CGE on these benchmarks. Note that in three of the five cases, our router is able to successfully route all nets using fewer edges per channel. This indicates that our router is relatively thrifty in its utilization of the available resources, and can produce feasible routings where other routers cannot. FPGAs are available in several standard sizes, each with a fixed number of edges per channel [29]; clearly being able to successfully route designs using fewer resources will enable the usage of smaller (and cheaper) standard-sized parts, or may allow designs to be implemented using fewer fixed-size FPGAs when the design cannot fit on a single FPGA chip. Figure 7 shows the solution produced by our router for the smallest of the benchmark circuit.

Breakdown of nets by number of pins					
Circuit	#nets	#2-3	#4-10	#over 10	FPGA size
BUSC	151	115	28	8	12 × 13
DMA	213	139	52	22	16 × 18
BNRE	352	255	70	27	21 × 22
DFSM	420	361	26	33	22 × 23
Z03	608	398	176	34	26 × 27

Table 2: A breakdown of the benchmark circuits by net size. Also shown is the size of the FPGA used to route each circuit (the number of rows and columns of FPGA logic blocks).

Maximum channel width required to route all nets		
Circuit	CGE	**IKMB**
BUSC	10	**8**
DMA	10	**9**
BNRE	12	**11**
DFSM	10	**11**
Z03	13	**13**

Table 3: The maximum channel width required by our router to successfully route all nets in each industry benchmark. For comparison we also give the analogous maximum channel widths required by the CGE detailed router.

6 Conclusion

We proposed a unified general framework for FPGA routing, allowing the simultaneous optimization of multiple competing objectives under a smooth designer-controlled tradeoff. Our approach is based on a new general multi-weighted graph formulation, resulting in an architecture-independent and computationally efficient methodology. Finally, multi-weighted graphs may be applied to other areas of CAD, as well as to many other classic combinatorial optimization problems (e.g., traveling salesman, matching, partitioning, spanning and Steiner trees, etc.)

7 Acknowledgments

The authors would like to thank Jonathan Rose and Stephen Brown for their code and advice.

References

[1] M. J. Alexander and G. Robins, *An Architecture-Independent Unified Approach to FPGA Routing*, Tech. Rep. CS-93-51, Department of Computer Science, University of Virginia, October 1993.

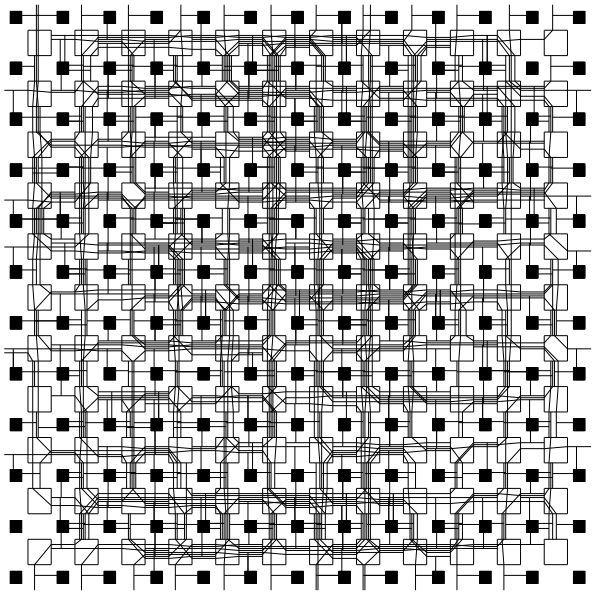

Figure 7: IKMB router solution for BUSC circuit.

[2] M. J. Alexander and G. Robins, *High-Performance Routing for Field-Programmable Gate Arrays*, in Proc. IEEE Intl. ASIC Conf., Rochester, NY, September 1994.

[3] M. J. Alexander and G. Robins, *A New Approach to FPGA Routing Based on Multi-Weighted Graphs*, in Proc. ACM/SIGDA International Workshop on Field-Programmable Gate Arrays, Berkeley, CA, February 1994.

[4] T. Barrera, J. Griffith, S. A. McKee, G. Robins, and T. Zhang, *Toward a Steiner Engine: Enhanced Serial and Parallel Implementations of the Iterated 1-Steiner Algorithm*, in Proc. Great Lakes Symp. VLSI, Kalamazoo, MI, March 1993, pp. 90–94.

[5] T. Barrera, J. Griffith, G. Robins, and T. Zhang, *Narrowing the Gap: Near-Optimal Steiner Trees in Polynomial Time*, in Proc. IEEE Intl. ASIC Conf., Rochester, NY, September 1993, pp. 87–90.

[6] N. B. Bhat and D. D. Hill, *Routable Technology Mapping for LUT FPGAs*, in Proc. IEEE Intl. Conf. Computer-Aided Design, 1992, pp. 95–98.

[7] S. Brown, J. Rose, and Z. G. Vranesic, *A Detailed Router for Field-Programmable Gate Arrays*, IEEE Trans. Computer-Aided Design, 11 (1992), pp. 620–628.

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

[9] P. K. Chan, M. D. F. Schlag, and J. Y. Zien, *On Routability Prediction for Field-Programmable Gate Arrays*, in Proc. ACM/IEEE Design Automation Conf., 1993, pp. 326–330.

[10] K. C. Chen, J. Cong, Y. Ding, A. B. Kahng, and P. Trajmar, *DAG-Map: Graph-Based FPGA Technology Mapping for Delay Optimization*, IEEE Design & Test of Computers, 9 (1992), pp. 7–20.

[11] J. P. Cohoon and D. S. Richards, *Optimal Two-Terminal $\alpha-\beta$ Wire Routing*, Integration: the VLSI Journal, 6 (1988), pp. 35–57.

[12] W. C. Collier and R. J. Weiland, *Smart Cars, Smart Highways*, IEEE Spectrum, 31 (1994), pp. 27–33.

[13] J. L. Ganley and J. P. Cohoon, *Routing a Multi-Terminal Critical Net: Steiner Tree Construction in the Presence of Obstacles*, in Proc. IEEE Intl. Symp. Circuits and Systems, London, England, May 1994, pp. 1.113–1.116.

[14] J. L. Ganley, M. J. Golin, and J. S. Salowe, *Minimum Spanning Trees for Multiply-Weighted Graphs.* unpublished manuscript, 1994.

[15] T. Gao, K. C. Chen, J. Cong, Y. Ding, and C. L. Liu, *Placement and Placement Driven Technology Mapping for FPGA Synthesis*, in Proc. IEEE Intl. ASIC Conf., Rochester, NY, September 1993, pp. 87–91.

[16] M. Garey and D. S. Johnson, *The Rectilinear Steiner Problem is NP-Complete*, SIAM J. Applied Math., 32 (1977), pp. 826–834.

[17] M. Hanan, *On Steiner's Problem With Rectilinear Distance*, SIAM J. Applied Math., 14 (1966), pp. 255–265.

[18] T. C. Hu and T. Shing, *The α-β Routing*, in VLSI Circuit Layout: Theory and Design, New York, 1985, IEEE Press, pp. 139–143.

[19] F. K. Hwang, D. S. Richards, and P. Winter, *The Steiner Tree Problem*, North-Holland, 1992.

[20] A. B. Kahng and G. Robins, *A New Class of Iterative Steiner Tree Heuristics With Good Performance*, IEEE Trans. Computer-Aided Design, 11 (1992), pp. 893–902.

[21] K. Karplus, *Xmap: a Technology Mapper for Table-lookup Field-Programmable Gate Arrays*, in Proc. ACM/IEEE Design Automation Conf., 1991, pp. 240–243.

[22] L. Kou, G. Markowsky, and L. Berman, *A Fast Algorithm for Steiner Trees*, Acta Informatica, 15 (1981), pp. 141–145.

[23] G. G. Lemieux and S. D. Brown, *A Detailed Routing Algorithm for Allocating Wire Segments in Field-Programmable Gate Arrays*, in Proc. ACM/SIGDA Physical Design Workshop, Lake Arrowhead, CA, April 1993.

[24] F. D. Lewis and W. C. Pong, *A Negative Reinforcement Method of PGA Routing*, in Proc. ACM/IEEE Design Automation Conf., 1993, pp. 601–605.

[25] J. Rose, *Parallel Global Routing for Standard Cells*, IEEE Trans. Computer-Aided Design, 9 (1990), pp. 1085–1095.

[26] K. Roy, B. Guan, and C. Sechen, *FPGA MCM Partitioning and Placement*, in Proc. ACM/SIGDA Physical Design Workshop, Lake Arrowhead, CA, April 1993, pp. 211–212.

[27] M. Schlag, J. Kong, and P. K. Chan, *Routability-Driven Technology Mapping for LookUp Table-Based FPGAs*, in Proc. IEEE Intl. Conf. Computer-Aided Design, 1992, pp. 86–90.

[28] S. Trimberger and M. R. Chene, *Placement-Based Partitioning for Lookup-Table-Based FPGAs*, in Proc. IEEE Intl. Conf. Computer-Aided Design, 1992, pp. 91–94.

[29] S. M. Trimberger, *Field-Programmable Gate Array Technology, S. M. Trimberger, editor*, Kluwer Academic Publishers, Boston, MA, 1994.

[30] B. Tseng, J. Rose, and S. Brown, *Improving FPGA Routing Architectures Using Architecture and CAD Interactions*, in Proc. IEEE Intl. Conf. Computer Design, 1992, pp. 99–104.

[31] Y. F. Wu, P. Widmayer, and C. K. Wong, *A Faster Approximation Algorithm for the Steiner Problem in Graphs*, Acta Informatica, 23 (1986), pp. 223–229.

[32] Xilinx, *The Programmable Gate Array Data Book*, Xilinx, Inc., San Jose, California, 1993.