AUTOMATED PLACEMENT AND ROUTING OF CELL MATRIX CIRCUITS

by

Dimitri V. Yatsenko

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Science

Approved:

_____
DONALD H. COOLEY
Major Professor

_____
NICHOLAS MACIAS
Committee Member

_____
SCOTT CANNON
Committee Member

_____
LISA DURBECK
Committee Member

_____
THOMAS KENT
Dean of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2003

ABSTRACT

Automated Placement and Routing of Cell Matrix Circuits

by

Dimitri Yatsenko, Master of Science

Utah State University, 2003

Major Professor: Dr. Donald Cooley
Department: Computer Science

Cell Matrix is a parallel self-configurable architecture. It is similar to field programmable gate arrays (FPGAs) in that Cell Matrix elements can be programmed. Cell Matrix circuits differ from FPGAs because groups of their elements can dynamically reprogram adjacent elements. Advanced behaviors arise from this ability, setting Cell Matrix apart from architectures lacking self-configurability.

This thesis specifies the first Cell Matrix compiler to automatically generate Cell Matrix circuits from logic specifications. Given a schematic design, a set of available cells, and an optional seed placement, the Cell Matrix compiler generates a complete and efficient code to load into Cell Matrix cells.

The placement algorithm combines self-organizing maps and force-directed optimization. An efficient gate placement emerges from the collective action of individual gates following simple parameterized rules. The wire routing uses an A* shortest-path algorithm. The overall approach is shown to be highly parallelizable, flexible, and versatile.

(45 pages)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

What is a Cell Matrix?

Cell Matrix [Durbeck, L. and Macias, N. The Cell Matrix: An architecture for nanocomputing. Nanotechnology, 12 (2000), 217-230., Durbeck, L. and Macias, N. Defect-tolerant, fine-grained parallel testing of a Cell Matrix. SPIE ITCom, Series 4867 (2002), 71-85. Ed Schewel, J., James-Roxby, P., Schmit, H., and McHenry, Macias, N. The PIG paradigm: The design and use of a massively parallel fine grained self-reconfigurable infinitely scalable architecture. In Proceedings of The First NASA/DOD Workshop on Evolvable Hardware, (1999), 175-80. Ed Stoica, A., Keymeulen D., Lohn J.] is a massively parallel self-configurable architecture developed by Cell Matrix Corporation. Its elements can be configured to perform logic operations, just as field-programmable gate arrays (FPGA) can be. Unlike FPGAs, Cell Matrix lacks the conventional control superstructure. Instead, Cell Matrix elements are capable of dynamically programming neighboring elements. These innovative properties of Cell Matrix permit advanced hardware-level behaviors such as fault tolerance, evolvable functionality, and dynamic configurability between "computation in time" and "computation in space" [Durbeck, L. and Macias, N. The Cell Matrix: An architecture for nanocomputing. Nanotechnology, 12 (2000), 217-230.].

Problem Statement

Until now, the placement and routing of Cell Matrix circuits have been essentially handcrafted.

This thesis introduces the first Cell Matrix compiler. The compiler translates conventional schematic designs into Cell Matrix code that can then be programmed into a Cell Matrix using existing utilities. The core steps of the compilation process are placement and routing. The

placement and routing problem is stated as follows: Given (a) a schematic design, (b) a set of available cells, and (c) an optional seed placement, the Cell Matrix compiler generates a complete and efficient Cell Matrix layout. The seed placement is introduced, primarily, to enable the designer to specify the locations of terminal wires in order to connect the circuit to other circuits or partitions.

The Cell Matrix Compiler effectively circumvents defective regions. The algorithm is extendable to other topologies, including three-dimensional ones. No attempt is made to implement the algorithm using the Cell Matrix itself. The algorithm is made highly parallelizable by keeping all information and operations localized.

Outline of Contribution

1. This thesis specifies a placement and routing algorithm for an innovative hardware platform previously lacking an automatic routing tool.
2. An original placement and routing algorithm is employed based on self-organizing maps.
3. A scheme is developed to blend placement and routing to efficiently utilize the Cell Matrix architecture's non-differentiated logic and routing resources.
4. The algorithm is developed in such a way that the distinct stages of the placement and routing process are performed by fundamentally identical iterations of the algorithm, with only its parameters changing from phase to phase.
5. An A* wire routing algorithm is demonstrated to increase the efficiency and optimality of the conventional maze solving algorithms cited in literature on circuit routing.

Present Procedure for Placement and Routing Process of Cell Matrix Circuits

Presently, Cell Matrix circuits are programmed by explicit specification of each cell's function. The circuit designer executes the steps outlined in Table 1. The objective of this thesis is to fully automate steps 4-8.

Table 1. Current Cell Matrix design steps.

1. Produce a schematic design to be implemented (manually or using schematic entry tools).
2. If necessary, partition the netlist into smaller parts to be implemented separately.
3. Allocate a Cell Matrix area for each netlist partition and define connections between the partitions.
4. Assign placement to logical elements within each Cell Matrix area.
5. Route wires connecting logical elements according to the netlist.
6. If some wires cannot be routed, make adjustments to the results of steps 4 and 5.
7. Specify the truth tables of each cell according to logical elements and wires residing in them.
8. Configure Cell Matrix hardware using the cell specifications from step 7. (Done by Cell matrix Corporation tools).

Target Topology

To limit the scope of the project, I will focus on a single baseline topology, the two-dimensional Cell Matrix with square cells. Each square cell is connected to its four adjacent cells as shown in Figure 1. In this baseline architecture, each side of every cell has a control input and a control output (marked as C), a data input and a data output (marked as D). I will keep the discussion sufficiently general and solutions sufficiently versatile that they still apply to more complex architectures.

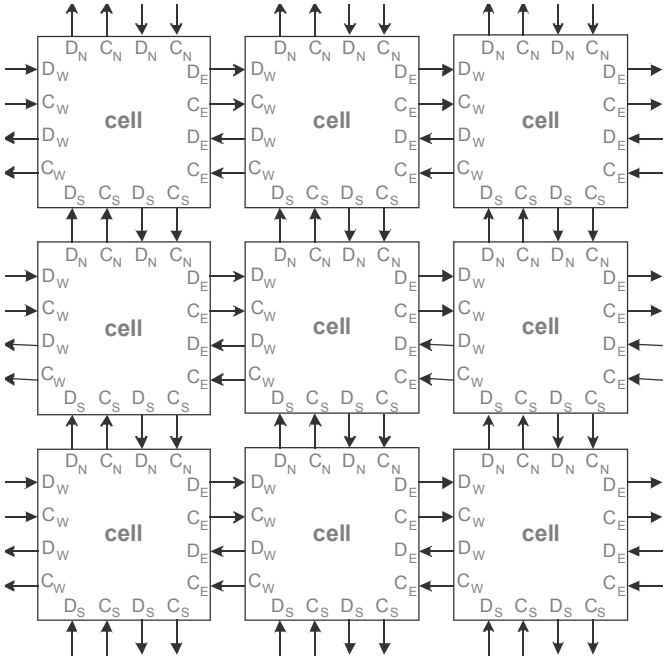


Figure 1. The four-connected Cell Matrix topology.

Cell Matrix Operation

All Cell Matrix functionality is accomplished by groups of cells without any centralized control.

D-Mode Operation

Cells function as wires or logic gates when in data mode (or D-mode).

A cell is in data mode when all its C inputs are 0. When in D mode, the cell uses its internal truth table to produce D and C outputs based on D inputs.

C-Mode Operation

In control mode (C mode), cells can be reprogrammed by adjacent cells to perform new functions or can serve as data storage units.

A cell enters C mode when one or more of its C inputs are set to one. When a C input is set

to one, the D output on that side of the cell becomes active. The cell's logic table acts as a FIFO queue. At each clock cycle, the Boolean sum of active D inputs is pushed onto one side of the table, the last bit of the truth table is popped out to the active D outputs, and the logic table shifts by one bit. Inactive D inputs are ignored.

The functionality afforded by a cell's C-mode sets the Cell Matrix apart from traditional field-programmable gate arrays. Although simple in operation on the single-cell level, collectively, Cell Matrix circuits can develop complex behavior such as obstacle avoidance, evolvable hardware, efficient scheduling of computational resources, and so forth.

Characteristics of Cell Matrix Related to Placement and Routing

Although similar in concept to the compilation of other architectures, Cell Matrix circuit compilation stands out in several important ways. The following sections describe properties of the Cell Matrix that may affect the choice of the circuit compilation approach.

Homogeneity

Cell Matrix components are not intrinsically differentiated. No superstructures exist for cell programming, routing, or synchronization. The Cell Matrix does not by itself dictate a direction for computation flow. It is symmetric and homogenous.

Non-differentiated Connectivity

Modern FPGA architectures provide rich routing circuitry that is separate from logic elements. This richness of routing circuitry affords separate algorithmic approaches for placement and routing, avoiding the complexity of the iterative or interlaced placement and routing processes. On the contrary, Cell Matrix components are used for both logic and connectivity, and the distinction between placement and routing is less defined. In general, near-optimal placement and routing can be only achieved in parallel or by iterating between the two processes.

Scalability

Thanks to its homogeneity, Cell Matrix is fully scalable. By increasing the size of the matrix, the system designer can increase its computational power almost proportionally. This property renders the partitioning step of the design process somewhat less critical. If the hardware compiler runs out of matrix resources, the designer has the option of simply extending the matrix.

Fault Tolerance

Unlike most conventional FPGA architectures, Cell Matrix hardware can be used even if many of its cells are faulty. Techniques are being developed to identify and map faulty cells [Durbeck, L. and Macias, N. Defect-tolerant, fine-grained parallel testing of a Cell Matrix. SPIE ITCom, Series 4867 (2002), 71-85. Ed Schewel, J., James-Roxby, P., Schmit, H., and McHenry]. Once faulty cells are identified, circuits must be routed differently for individual matrices. No one precompiled layout will work for all matrices.

Multiple Topologies

Although only the two-dimensional square-cell four-connected topology is being considered in this thesis (as shown in Figure 1), other conceptual topologies have been studied [Macias, N. The PIG paradigm: The design and use of a massively parallel fine grained self-reconfigurable infinitely scalable architecture. In Proceedings of The First NASA/DOD Workshop on Evolvable Hardware, (1999), 175-80. Ed Stoica, A., Keymeulen D., Lohn J.].

- Hexagonal six-connected cells constitute another natural choice for a planar topology.
- 3D topologies with cubic, tetrahedral, and other cell shapes. Three-dimensional topologies will yield a great increase of performance over 2D topologies as they pack greater numbers of cells within smaller distances of each other.
- 2.5-dimensional circuits are defined as a set of parallel 2D matrices with sparse wires connecting the two. The term “2.5-dimensional” is borrowed from the animation industry but applies as well in hardware compilation [Ruben, S. Computer Aids for VLSI Design,

2nd edition, 1994. First printed as part of the Addison-Wesley VLSI Systems Series. Addison-Wesley Publishing Company, 1987. Out of print, 1993. Copyright returned to Steven M. Rubin. 1997. http://www.rulabinsky.com/cavd/].

These alternative topologies retain most of the basic Cell Matrix characteristics. The Cell Matrix Compiler algorithm is designed to extend easily into these alternative topologies.

Terminology

Since the Cell Matrix has much in common with FPGAs, most FPGA terminology applies. However, to avoid ambiguity in the inherently diverse terminology of circuit routing, I will use the following definitions throughout this thesis.

Cell: A cell is the fundamental physical unit of Cell Matrix uniquely identified by its location on the circuit.

Port: A port is a physical one-directional connection between two cells. Only D ports for data mode operation are considered in the scope of this project.

Gate: A gate is a logical unit without memory or internal loops with one or more inputs and/or one or more outputs. Multiple gates may be placed on the same cell until the cell runs out of ports to route the connected wires or feedback loops form within the cell.

Wire: A wire is a logical directed relation between gates. A wire logically associates a source gate with a destination gate.

Netlist: A netlist is a set of gates and wires connecting them.

Route: A route is a path through multiple gates and ports from the origin cell to the destination cell. A route is associated with one and only one wire. No two valid routes can pass through the same port, but multiple routes may cross a gate. A route does not take up any ports if its source and destination gates are different and reside on the same cell.

Terminal: A terminal is a port that connects to the outside world. Input and output terminals are

distinguished from each other.

Functional distance: The functional distance between two gates is the smallest number of wires separating the two gates. The direction of wires is not considered.

Functional layer: The n-th functional layer for a gate G consists of all gates removed by exactly n wires from G.

Functional neighborhood: The functional neighborhood of radius r comprises the first r functional layers.

Functional neighbor: The functional neighbor of gate G is a gate that belongs to some functional layer of G.

Manhattan distance: Also known as rectilinear distance, the Manhattan distance between two points is defined as the sum of distances along each coordinate axis.

PREVIOUS WORK

The computational complexity of exhaustive placement algorithms is proportional to n! where n is the number of modules to be placed. Thus the placement problem is NP-complete [Garey, M. and Johnson, D. The rectilinear Steiner tree problem is NP-complete. SIAM Journal of Applied Mathematics, 32 (1977) 826-834, Cai, H. Gridless Routing System for Macro-Cell Design. Delft University of Technology, Delft, The Netherlands. Published in Zobrist, G. Routing, Placement, and Partitioning, chapter 2. Alex Publishing Corporation, 1994.]. Despite this, a number of approaches are used in commercial software to generate nearly optimal placement and routing in polynomial time. Although significant research has been done on the subject of automated hardware generation [Ruben, S. Computer Aids for VLSI Design, 2nd edition, 1994. First printed as part of the Addison-Wesley VLSI Systems Series. Addison-Wesley Publishing Company, 1987. Out of print, 1993. Copyright returned to Steven M. Rubin. 1997. http://www.rulabinsky.com/cavd/.], implementations remain proprietary and their details undisclosed. As such, they are not part of academic scholarship or discourse. Limited work has been done on general placement and routing algorithms to generate circuits for arbitrary hardware architectures [Betz, V. and Rose, J. VPR: A new packing, placement, and routing tool for FPGA Research. In Proceedings of International Workshop on Field Programmable Logic and Application, 1997]. Commercial hardware generation remains architecture-specific and application-specific to take advantage of the strengths of the target system and to utilize proven standard pre-routed blocks.

Placement algorithms in the literature fall into two categories: constructive (or opening) and iterative (or improving). Because of the high computational complexity of the placement and routing problem, constructive algorithms are typically feasible only for small or well-understood and regular circuits. Furthermore, constructive algorithms typically require global planning and

centralized processing, while iterative algorithms achieve similar goals using local information only. Minimal global communication makes iterative algorithms more suitable for parallel implementation.

Limited global communication better suits a future Cell Matrix implementation of the Cell Matrix Compiler. This Cell Matrix module would program a neighboring Cell Matrix according to a schematic design provided as input.

For these reasons, this overview is limited to iterative placement and routing algorithms.

Automated Placement Algorithms

Simulated Annealing

Simulated annealing [Cai, H. Gridless Routing System for Macro-Cell Design. Delft University of Technology, Delft, The Netherlands. Published in Zobrist, G. Routing, Placement, and Partitioning, chapter 2. Alex Publishing Corporation, 1994., Zhang, C. VLSI Placement, Institute of Theoretical Electrical Engineering, Unversity Karlsruhe, Kralsruhe, Germany. Published in Zobrist, G. Routing, Placement, and Partitioning, chapter 4. Alex Publishing Corporation, 1994] is a general-purpose optimization technique well-suited for the placement problem. An initial placement is generated either randomly or by a primitive constructive placement algorithm. Thereafter, simulated annealing iterations are performed. At each iteration, new gate locations are chosen randomly. A quality function is used to evaluate Δq the improvement in placement quality of a gate. The gate is relocated to the new cell with probability $p = \exp(\Delta q / T)$, where T is the global temperature. Gates always move toward improved placement, but the higher the temperature, the higher the probability of relocating into lower-quality placements. The algorithm starts with a high value of T, which prevents the algorithm from getting trapped in local minima. Over time, the

temperature is lowered and the algorithm converges on a local minimum. If the cooling schedule is gradual enough, the algorithm is likely to converge very close to the optimum solution.

Zhang in [Zhang, C. VLSI Placement, Institute of Theoretical Electrical Engineering, Unversity Karlsruhe, Kralsruhe, Germany. Published in Zobrist, G. Routing, Placement, and Partitioning, chapter 4. Alex Publishing Corporation, 1994] points out that, although variations of the simulated annealing algorithm have enjoyed a wide acceptance in placement and routing, their convergence rates are generally slow.

Because in Cell Matrix compilation placement and routing are inseparable, both have to be performed simultaneously further slowing the convergence rates.

Energy Minimization and Force-directed Methods

Energy-minimization and force-directed methods [Zhang, C. VLSI Placement, Institute of Theoretical Electrical Engineering, Unversity Karlsruhe, Kralsruhe, Germany. Published in Zobrist, G. Routing, Placement, and Partitioning, chapter 4. Alex Publishing Corporation, 1994] apply an attracting force between connected gates and repelling forces between all other gates.

This is a very straightforward and fast algorithm; however, it lacks the exploratory properties of the simulated annealing algorithm, quickly settling in a local optimum.

Neural Network Routers or Self-Organizing Maps

Neural network placement algorithms (sometimes described as self-organizing maps) [Ritter, H. and Shulten, K. Kohonen's self-organizing maps: Exploring their computational capabilities. In Proceedings of the IEEE International Conference on Neural Networks (1988), 109-116] represent gates as neurons. Each neuron s weight value is a vector specifying the gate s placement. Excitations take place at random cells in the placement space. A gate physically closest to the excitation site becomes active. The excitation then propagates from the active gate to its functional neighbors, decreasing in strength with functional distance from the active gate. Each excited gate s position is adjusted toward the original excitation point proportionally to the remaining excitation

strength and the capacity of the excited cell to host the excited gate. Fractional adjustments may accrue until they result in moving gates into new discrete positions. Figure 2 illustrates an iteration of this process.

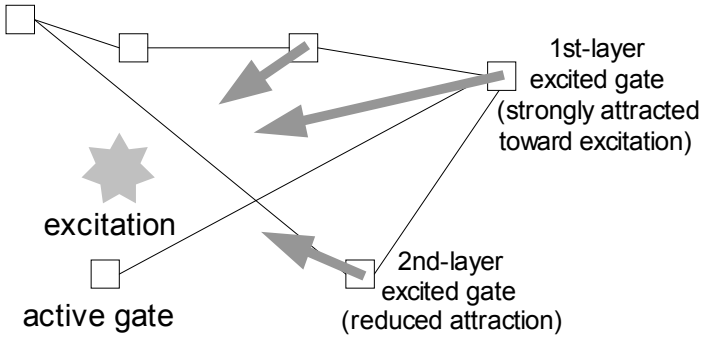


Figure 2. Mapping adjustment iteration.

The two main factors that dictate the convergent behavior of the algorithm are the *excitation distribution function* and the *functional layer function*.

The *excitation distribution function* determines how the next excitation is selected. At early stages of the placement process, the excitation distribution function is uniformly distributed over the available chip area. As the placement convergence, the excitation distribution function is generated to be more densely occupied cells, producing more compact placements.

The *functional layer function* determines how excitations diminish in strength as they propagate away from the active gate. Typically, an exponentially decaying function is used. Fast decay rates work well early in the placement process (macroplacement), and fast decay rates are used late in the placement process (microplacement and optimization).

Steinberg Algorithm

The Steinberg algorithm [Steinberg, L. The backboard wiring problem: A placement algorithm. SIAM Review, 3 (1961) 37-50., Chen, W. and Wang, K. Placement. Department of Electrical

Engineering and Computer Science, University of Illinois at Chicago. Published in Zobrist, G. Routing, Placement, and Partitioning, chapter 5. Alex Publishing Corporation, 1994., Brixius, N. and Anstreicher, K. The Steinberg Wiring Problem. University of Iowa] is a powerful technique that quickly improves an existing placement. First, a set of gates is generated in which no gate is connected to any other gate in the set. To produce such a set, the algorithm starts out with a random gate and then keeps adding gates that are not connected to the gates already in the set. This is done without worrying much about the yet unplaced gates. Next, these gates are unplaced and then placed optimally in relation to their placed functional neighbors. Finally, a new independent set is generated, and the process is repeated. The Steinberg algorithm converts the placement optimization problem from a combinatorial NP-complete problem into iterations of O(n) problems, each improving the placement quality.

Selection Criteria

The listed algorithms are not all mutually exclusive. A successful solution may involve a combination of these algorithms executed sequentially or iteratively. For example, my experimentation suggests that the effectiveness of simulated annealing can be greatly increased by interspersing its iterations with iterations of the Steinberg algorithm.

To select a combination of algorithms for placement and routing of Cell Matrix circuits, I evaluated how effectively they addressed a set of intermediate goals that I found to be essential for a successful Cell Matrix compiler. This section describes these intermediate goals.

Moving Connected Gates Close to Each Other

The algorithm must quickly and effectively move connected cells close to each other. This minimizes the total wire length and the number of Cell Matrix cells dedicated to routing.

Distributing Gates Through Available Chip Area

To route a group of gates, it becomes necessary to spread them more sparsely to free cells between

gates for routing. Unused portions of the chip may have to be found and utilized. Some algorithms (e.g. energy minimization methods, force-directed methods, Steinberg□s algorithm) do not attempt to spread gates apart and, if left to their own devices, will result in a tight knot of gates and wires in a small area. Other algorithms (e.g simulated annealing, neural networks) are very effective at exploring all available chip area.

Tightening Near-Optimal Placements

An algorithm may not take advantage of □trivial□ and inexpensive placement adjustments. For example, nudging the placement of a gate by one cell may reduce its routing demands. Simulated annealing in its conventional implementation does not □see□ such trivial adjustments, while force-directed methods immediately find them all.

Obstacle Avoidance

The complex geometry of a chip or the presence of faulty cells must not degrade the effectiveness of placement. A force-directed method, for example, may continually pull gates into dead-end areas where routing is hampered by obstacles.

Escaping from Local Optima

A placement algorithm must not give up seeking better solutions when a good solution is found. It may be necessary to □move uphill□ or, in other words, reduce the quality of the placement temporarily while in search of the global optimum. As an example, simulated annealing algorithms are very effective at avoiding local optima when the temperature is high.

Avoiding Deadlocks

To optimize the placement of a single gate, it often becomes necessary to displace multiple gates. As a worst-case scenario, consider a placement in which no more free cells are available. A simulated annealing algorithm that moves one gate at a time would seize up without having free cells to perform the next step. The Steinberg algorithm, for example, deals very effectively with such deadlocks.

Summary of Existing Placement Techniques when Applied to the Cell Matrix

Table 2 summarizes the strengths and weakness of each approach, based on my experimentation and literature review. 'H' means high effectiveness, 'M' means medium effectiveness, and 'L' low.

Table 2. Summary of placement techniques when applied to the Cell Matrix.

	Simulated Annealing	Steinberg	Force-directed & Energy Minimazation	Neural Networks
moving connected gates closer	M	H	H	M
distributing gates through available chip area	H	L	L	H
tightening up of near-optimal placements	L	H	H	L
escaping from local optima	H	L	L	M
avoiding deadlocks & twisted wires	M	L	L	M

This analysis predicts, for example, that a combination of iterations of the simulated annealing algorithm with iterations of the Steinberg algorithm or a forced-directed method is likely to result in a robust algorithm. The Cell Matrix compiler combines a variation of the neural network placement algorithm (or self-organizing map) with a force-directed method, which, according to this analysis, is also a strong symbiosis.

Automated Routing Algorithms

Lee-Moore Maze Routing

The routing problem can be described as a minimal spanning tree problem, for which Dijkstra's minimal spanning tree algorithm is an effective and well-known solution. When applied to maze routing, Dijkstra's minimal spanning tree algorithm is known as the Lee-Moore algorithm [Moore, E.F. Shortest Path Through a Maze. Harvard University Press (1959), 285-292, Lee, C. Y. An algorithm for path connections and its applications. IRE Transactions on Electronic Computers, EC-10 (1961), 346-365]. This algorithm works by propagating a concentric wave around the origin cell. The wave circumvents unavailable or faulty cells. When the destination cell is reached, the shortest path is traced back through the successive steps of the wave. The algorithm is so common and straightforward that hardware implementations have been developed [Blank, T. A survey of hardware accelerators used in computer-aided design. IEEE Design and Test, 1, 3 (1984) 21-39.].

The Lee-Moore algorithm guarantees finding the shortest path. It straightforwardly extends into various connectivity schemes and higher dimensions.

Maze Walking

A downside of the Lee-Moore algorithm is its unacceptably slow execution and large memory requirements on large topologies, particularly in three-dimensional architectures. Some authors [Ruben, S. Computer Aids for VLSI Design, 2nd edition, 1994. First printed as part of the Addison-Wesley VLSI Systems Series. Addison-Wesley Publishing Company, 1987. Out of print, 1993.

Copyright returned to Steven M. Rubin. 1997. http://www.rulabinsky.com/cavd/, Hightower, D. A solution to line-routing problems in the continuous plane. In Proceedings of the 6th Design Automation Workshop (1969), 1-24.] suggest using maze-solving algorithms (e.g. line search routing) that walk toward the destination and circumvent obstacles by walking along their walls. However, these algorithms do not necessarily find the shortest path and may not be directly extendable into higher dimensions and more complex architectures.

A* Maze Routing

If the Lee-Moore algorithm is a breadth-first algorithm (slow but sure) and maze-walking algorithms are depth-first (fast but suboptimal), then the field of artificial intelligence suggests that the A* algorithm has the advantages of both approaches: fast but still guaranteeing the best solution [Russell, S. and Norvig, P. Artificial Intelligence: A Modern Approach. Prentice Hall, 2nd edition (2002).]. Although the A* algorithm is known as a good solution to the general shortest path and navigation problems, I was surprised not to find it cited in the placement and routing literature.

The Cell Matrix Compiler uses a variation of this algorithm to quickly find the shortest route. A more detailed description of the algorithm is given later in the paper.

The Circuit Design Process

Figure 3 depicts the circuit design process proposed in this thesis. The two steps performed manually by the circuit designer are schematic entry and matrix resource specification. The step numbers in italic correspond to the steps in the existing Cell Matrix development process outlined in Table 1. The Cell Matrix Compiler automates the steps in the gray box.

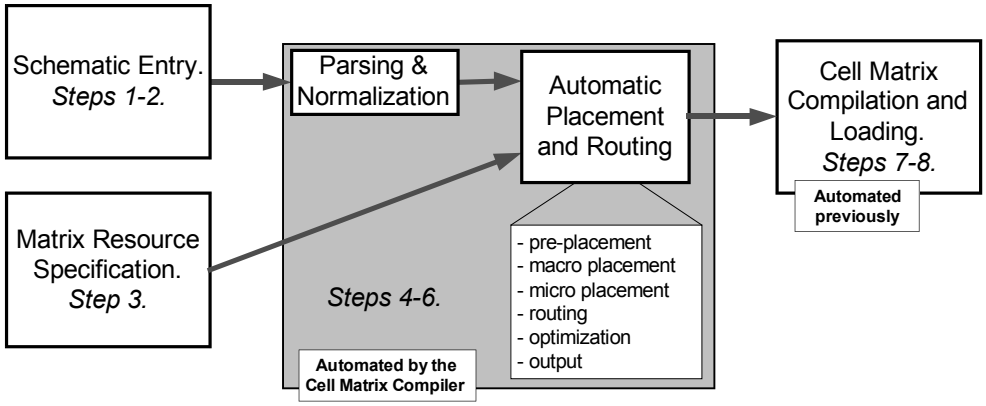


Figure 3. Proposed Cell Matrix circuit development process.

Schematic Entry

With the introduction of the Cell Matrix compiler, Cell Matrix designers have the option of using a standard schematic entry software package to specify the schematic design to be implemented. For this thesis I developed a library of logic elements in Orcad Splice schematic entry software.

Figure 4 depicts a simple circuit specified with the use of this library. Schematic entry tools then save the schematic design as a netlist in a hardware definition language. The Cell Matrix Compiler uses VHDL, but others can be used as well. Users may choose to bypass schematic entry and develop circuits directly in VHDL or another hardware definition language.

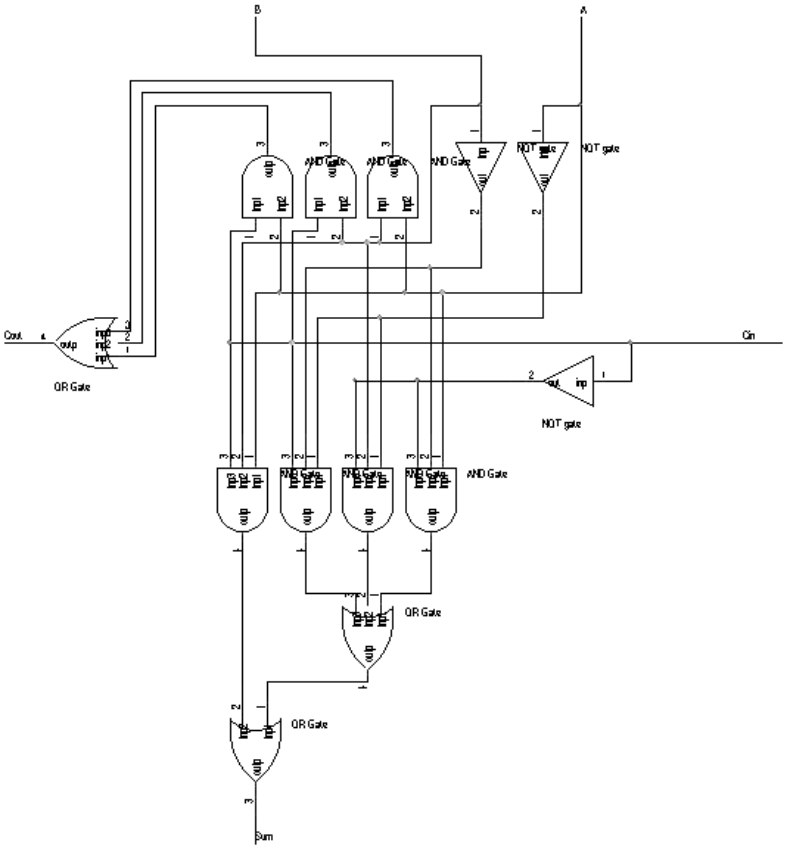


Figure 4. A four-bit adder circuit entered using the Cell Matrix element library in Orcad.

Not all legal netlists or schematic designs are legal inputs to the Cell Matrix compiler. To be routed successfully by the Cell Matrix Compiler, a netlist must comply with the following set of constraints:

1) All gates must be simple enough to be hosted on a single cell. More complex components must be decomposed into their elementary gates. This constraint applies when new elements are added to the Cell Matrix Library.
2) No wire may remain dangling. Terminal wires must be connected to designated input or output terminal gates.
3) No pin may remain disconnected.

Matrix Resource Specification

Matrix Resource Specification consists of specifying the following items:

1) Cell Matrix geometry
2) Map of faulty cells
3) Seed placements (manual placement of input and output terminals or other gates)

Parsing and Normalization

The parser reads the VHDL specification of the circuit. Normalization is the process that ensures that each wire connects one and only one output pin to one and only one input pin. Wire splits are replaced with special “split” gates. Although wires merges do not make sense in the Cell Matrix architecture, VHDL allows them, and I made the program replace them with special “merge” gates. Figure 5 depicts the routing of the circuit specified in Figure 4 as generated by the algorithm. Notice the absence of wire splits or merges – they are replaced with auxiliary gates. Also notice that multiple gates are sometimes hosted on the same cell. Unless loops form inside a cell, such cell sharing is valid and efficient and arises commonly in optimized circuits. The black cells are defective and are not used for placement and routing. In the figure, the input terminal gates are positioned along the left edge of the chip, and the output terminal gates are positioned along the

right edge.

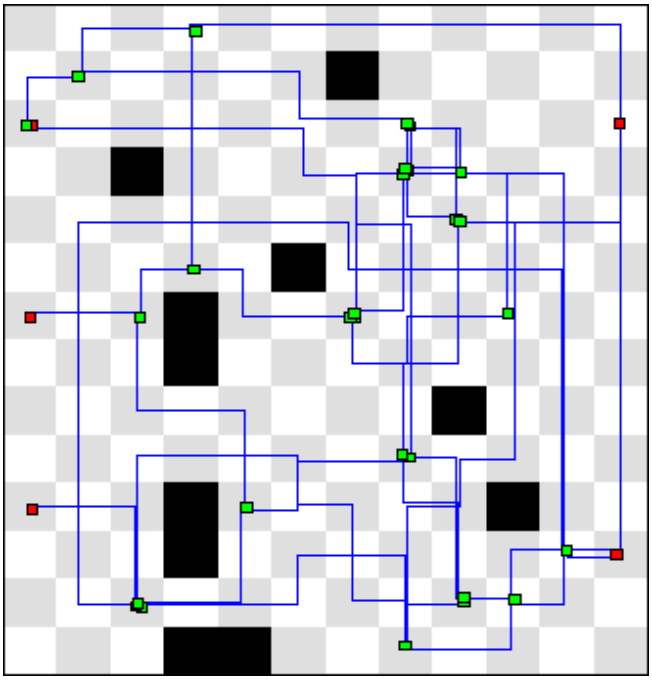


Figure 5. The netlist from the circuit in Figure 4 placed and routed by the placement algorithm on a 12x14 Cell Matrix with 7% defective cell rate. After normalization, the netlist has 46 wires and no wire splits.

General Approach: Interlaced Placement and Routing

Due to the non-differentiated connectivity of the Cell Matrix architecture, its logic blocks and wiring compete for the same hardware resources. This competition is less severe in conventional FPGA architectures in which dedicated routing structures connect logic elements. An interesting dilemma arises from this property of the Cell Matrix. Should routing be performed through iterations of the placement algorithm or should it be performed after the gate placement is completed? On the one hand, the validity or quality of the gate placement cannot be accurately estimated until the routing is performed, so gate placement is not complete until the routing is ensured. On the other hand, complete routing at each iteration of the placement algorithm will

result in a prohibitive increase in the computational complexity of gate placement.

I have identified the following three approaches to address this dilemma.

1. Attempt to route all wires at the end of each iteration of the placement algorithm. This increases the complexity of the placement algorithm, but enables the placement algorithm to allocate sufficient and not excessive number of cells for wire routing.
2. Use sophisticated heuristics to allocate enough space around placed gates for future routing. It is almost certain, however, that such heuristics will waste Cell Matrix resources by allocating excessive space for wire routing or will not guarantee successful routing at the end of the placement phase.
3. Start with minimal routing rates and gradually increase the routing rate as the placement algorithm converges. A simple heuristic predicts the placement quality for unrouted gates.

The third approach combines the placement efficiency of the first approach with the computational efficiency of the second approach. This reasoning led me to develop an iterative algorithm that gradually blends wire routing into gate placement as the placement converges.

At each iteration, all gates follow a set of simple rules. Small adjustments in behavioral propensities of individual cells result in drastic changes of their collective behavior. A single algorithm performs all stages of the placement and routing process with only parameter values changing. To estimate the placement quality of unrouted gates, I introduced a neighborhood quality map, which is described later in the paper.

Core algorithm of the Cell Matrix Compiler

Table 3 summarizes the rules applied to each *gate* at every iteration throughout the execution of the algorithm. The rules are parameterized in order to optimally blend placement and routing as the algorithm goes through compilation stages.

Table 3. Cell Matrix compiler iteration

9. With probability P_1, the *gate* attempts to route wires connected to it.
10. The *gate* evaluates its 'comfort' based on the lengths of all routed wires or the Manhattan distance to unrouted functional neighbors.
11. The *gate* evaluates its 'happiness' based on its current 'comfort', its past 'happiness', and the minimal comfort of its functional neighbors. The latter two criteria are controlled by the *forgetfulness* and *empathy* parameters.
12. If the *gate* is amongst P_2 portion of the unhappiest gates globally, it unroutes all its connected wires.
13. If the *gate* is completely unrouted, i.e. none of its wires are routed, it becomes available for a placement adjustment. Placement adjustments are performed by an iteration of a neural network placement algorithm.
14. If the *gate* is routed, then, with probability P_3, check whether one of the adjacent cells allows for shorter wire routing and, if so, move the *gate* to that cell.

Compilation Stages

A typical placement and routing algorithm comprises the following stages:

1. *Macroplacement or Partitioning*. During macroplacement, the general chip structure is established. Functionally connected groups of gates are placed in physical chip areas. *Partitioning* is a stricter form of macroplacement, in which groups of gates are permanently assigned to physical chip areas or separate chips and cannot migrate to other areas during later routing stages.
2. *Microplacement.* Detailed placement. Individual gates are assigned to specific cells.
3. *Routing.* Routes are found for each wire.
4. *Optimization (compaction).* The placement and routing is optimized for chip area and wire length without disrupting wire routing.

Most routers employ separate algorithms for each stage. An advantage of the Cell Matrix compiler algorithm is that a single algorithm implements all four of these stages. By simply modifying the control parameters P_1, P_2, P_3, *forgetfulness*, and *empathy* in the described rules, the emergent behavior undergoes a qualitative transformation.

The Cell Matrix Compiler algorithm moves to the next stage when a set percentage of wires is routed and a set mean wire length is reached. These two conditions constitute the stage transition criterion. In the new stage, new algorithm control parameter values and transition criterion values are assumed.

A set of algorithm stages with corresponding control parameter values and stage transition criteria constitute the algorithm schedule. The effectiveness of the overall algorithm is based on the algorithm schedule. The generation of the algorithm schedule is intuitive but difficult to optimize precisely.

Through extensive experimentation, I found that the algorithm schedule in Table 4 consistently yielded fast placement and routing of netlists with complexity levels varying from a few gates to thousands of gates.

Table 4. Algorithm schedule.

Algorithm Parameters and Stage Transition Criteria	**Valid Range**	**Algorithm Stages**				
		"Pre-placement"	*"Macro-placement"*	*"Micro-placement"*	*"Routing"*	*"Optimization"*
P_1	0.0 – 1.0	0.25	1.0	1.0	1.0	1.0
P_2	0.0 – 1.0	0.1	0.4	0.2	0.1	0.5
P_3	0.0 – 1.0	1.0	0.1	0.2	0.3	1.0
forgetfulness	0.0 – 1.0	0.9	0.5	0.2	0.2	0.2
empathy	0.0 – 1.0	0.1	0.5	0.5	0.5	0.2
% routing required to advance to next stage	0 – 100	60	84	90	97	100
Mean wire length required to advance to next stage	Based on user's desired degree of optimization	99	50	50	50	Dependent on size and complexity

Position Adjustment

Following the analysis of existing placement algorithms captured in Table 2 and characteristics of the Cell Matrix, I chose to combine a variation of the neural network algorithm in [Ritter, H. and Shulten, K. Kohonen's self-organizing maps: Exploring their computational capabilities. In Proceedings of the IEEE International Conference on Neural Networks (1988), 109-116.] with an energy minimization method to solve the placement portion of the compilation problem. The neural network algorithm performs the position adjustment of unrouted gates and the energy minimization algorithm optimizes the positions of routed gates.

In position adjustment algorithm, a cell on the chip is "excited." The gate closest to the excitation becomes the active gate. The functional neighbors of the gate are then moved toward the excitation (not toward the active gate) in proportion to the excited cell's available capacity. This approach forces cells to spread into underutilized portions of the matrix while also moving functional neighbors into physical proximity.

The convergence rate of the algorithm improved significantly when I added a "neighborhood quality" map. The neighborhood quality map helps determine the likelihood of routability of a gate should it be placed in a particular cell. This estimate is based on recent routing history in the neighborhood of the cell. When a gate moves toward an excited cell, it will prefer a place that does not have a recent history of unrouted wires or excessive wire traffic. Anytime a gate cannot route its wires, it updates the quality of its cell to a very low value. With each iteration, the neighborhood quality map gets convolved with a smoothing kernel and is normalized to keep the mean neighborhood quality constant. Thus, with time, the quality metric of a neighborhood decays in intensity and diffuses in space.

Wire Routing

A* Approach

For wire routing, the Cell Matrix compiler incorporates an A* maze routing algorithm. The algorithm is proven [Russell, S. and Norvig, P. Artificial Intelligence: A Modern Approach. Prentice Hall, 2nd edition (2002).] to possess the properties of completeness and optimality. This means that it guarantees the best solution while consistently visiting significantly fewer cells than breadth-first algorithms such as the Lee-Moore maze router.

The algorithm stores all paths that may lead to an optimal solution but preferentially visits cells that lead toward the destination cell, as determined by the Manhattan distance metric. When obstacles force the currently pursued route out of the direct path toward the destination, the algorithm begins pursuing other solutions. The A* algorithm realizes this decision process in an elegant and succinct manner.

Sample Execution

Figure 6 compares a sample run of the conventional Lee-Moore maze algorithm (a) and the A* maze algorithm (b).

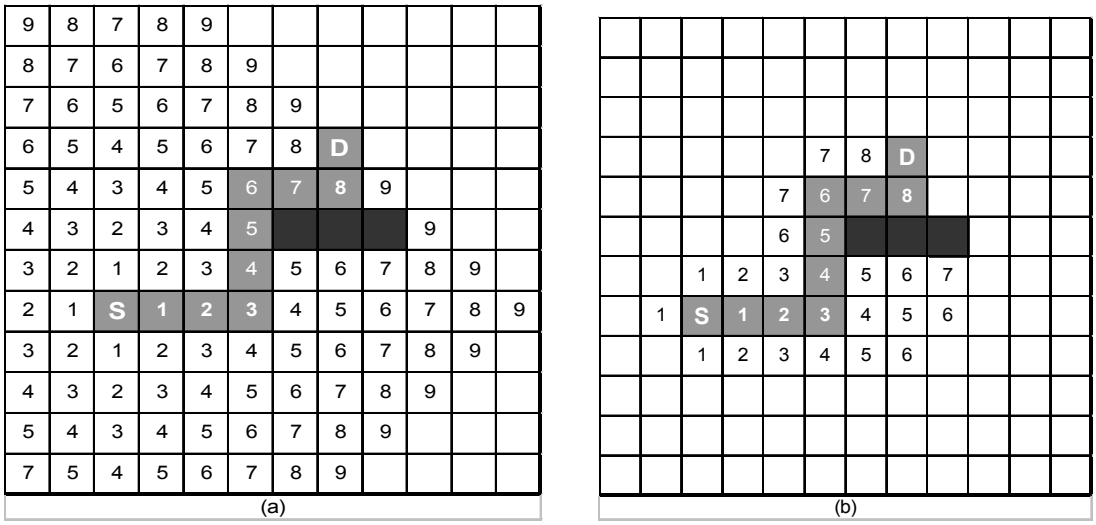


Figure 6. Sample execution results of the Lee-Moore router (a) and the A* router (b). ‘S’ marks the source cell. ‘D’ marks the destination cell. The dark cells are faulty. Cells that have been visited by the algorithms are marked with numbers indicating the number of cells traversed since the source cell.

Algorithm Listing

The following is the A* maze routing algorithm in its entirety:

function *find_route*(*cells, steps, destination*)

current_cell ← *pop_top*(*cells*)

if *current_cell* = Ø **then** return Ø // no route exists

new_cells ← *neighbor_cells*(*current_cell*)

compute_cost(*new_cells, steps*)

cells ← [*cells, better_than*(*new_cells, current_cell*)]

if *destination* **is in** *new_cells*

then

return [*destination*]

else

return [*current_cell*,*find_route*(*sort_by_cost*(*cells*), *destination, steps+1*)]

end

Invocation

To invoke the function, the cost values of all cells in the Cell Matrix are cleared (initialized to *null*) and the *find_route* function is invoked as follows:

route ← *find_route*([*source_cell*], 0, *destination_cell*)

Upon completion, the *route* variable will contain an ordered list of cells connecting the *source_cell* and the *destination_cell,* or Ø if no route exists between the two cells.

Obstacle Avoidance

Obstacle avoidance is accomplished by the *neighbor_cells*(*cell*) function, which returns all cells accessible from the given cell, even if they have already been visited. Cells are inaccessible if they are faulty or if the ports connecting them to the given cell are already in use. Note that cell

A may be accessible from cell B while cell B is not accessible from cell A because connections between cells are directional.

The cost function

The *compute_cost*(*cell*) routine associates the cell with a cost value computed as:

$$cost(\ cell\ ) \leftarrow \min(\ cost(\ cell\ ),\ Manhattan(\ cell, destination\ ) \cdot L + steps \cdot (\ L\text{-}1\ )\ )$$

where *Manhattan*(*cell*, *destination*) is the Manhattan distance between the cell and the destination and L is a very large number.

If the cell already has a cost value associated with it, it is replaced when the new cost value is smaller.

The *better_than*(*cells*, *cell*) function returns the subset of *cells* for which the cost value is lower than that of *cell*.

Functions *sort_by_cost*(*cells*) and *pop_top*(*cells*) work together to find the cell with the lowest cost value, remove it from *cells*, and use it in the current recursion of the algorithm.

Routing Interactions

Another challenge in wire routing is caused by the fact that each wire is routed without consideration for other wires. The Cell Matrix compiler resolves this problem by repetitive ordered rerouting. The wires connecting "happiest" gates are routed first or remain routed. An additional improvement in routing rates is achieved when the wires in the most congested areas are routed first, as suggested by Hightower in [Hightower, D. A solution to line-routing problems in the continuous plane. In Proceedings of the 6th Design Automation Workshop (1969), 1-24.].

Parallels with Existing Placement and Routing Algorithms

At various values of the algorithm parameters, the placement algorithm described here results in global behaviors similar to conventional placement and routing algorithms. An advantage of the Cell Matrix compiler algorithm is its flexibility and versatility. With its parameters adjusted,

the algorithm can perform similarly to simulated annealing, self-organizing maps, and energy-minimization methods taken separately.

Parallels with Simulated Annealing

The role of the happiness value is similar in function to the temperature value in the simulated annealing placement algorithm [Zhang, C. VLSI Placement, Institute of Theoretical Electrical Engineering, Unversity Karlsruhe, Kralsruhe, Germany. Published in Zobrist, G. Routing, Placement, and Partitioning, chapter 4. Alex Publishing Corporation, 1994]. The fact that 'happiness' is localized makes the algorithm dedicate its computational resources to the poorly placed gates. It also makes the algorithm susceptible to getting trapped in local minima when clusters of well-placed gates resist change. This problem is addressed by the concept of "empathy." The happiness of a cell becomes limited by the comfort of its neighbors. When the empathy coefficient is high, the inability of some gates to be placed and routed translates into the relative agility of the entire population, beginning with the immediate functional neighborhood. In this state, the algorithm closely resembles simulated annealing with good macroplacement performance.

Parallels with Neural Networks and Self-Organizing Maps

My early tests showed the Cell Matrix complier algorithm's incredible ability to untangle large netlists in a minimal number of iterations. I chose a uniform excitation distribution function, which resulted in excitations to be equally likely to happen in all available placement space. I used an exponentially decreasing layer function similar to [Ritter, H. and Shulten, K. Kohonen's self-organizing maps: Exploring their computational capabilities. In Proceedings of the IEEE International Conference on Neural Networks (1988), 109-116.]. Step 5 in Table 3 implements an idea very similar to the neural network algorithm described by Ritter [Ritter, H. and Shulten, K. Kohonen's self-organizing maps: Exploring their computational capabilities. In Proceedings of the IEEE International Conference on Neural Networks (1988), 109-116].

Parallels with Energy Minimization and Force-directed Methods

To compensate for the neural network algorithm's poor convergence to a local minimum, combining it with an energy minimization method proved very effective. Step 6 in Table 3 implements a variation of the energy minimization method. When a gate is fully routed, it explores, without disturbing the routing, whether one of the gate's adjacent cells allows for a more efficient routing. This process brings little benefit early in the placement process since the gate is likely to be relocated, so the value of P_3 approaches 100 percent in the optimization stage. The value of P_3 is set high in the pre-placement stage to optimize the positions of key gates serving as the skeleton for the rest of the placement.

Results

Validation

I used two distinct processes to validate and verify the performance of the Cell Matrix Compiler. The validation process was accomplished by going through the entire process of Cell Matrix design as depicted in Figure 3. Circuits were entered through the schematic capture tool, compiled by the Cell Matrix Compiler, and loaded into the Cell Matrix simulator. Then the circuits' functionality was tested in the Cell Matrix Simulator by verifying that they returned correct output for given input. Nick Macias and Lisa Durbeck of Cell Matrix Corporation also conducted a series of such tests using realistic useful circuit specifications. The circuit in Figure 4 and Figure 5 is an example of such a test.

The validation tests confirmed that, the Cell Matrix Compiler always generated valid layouts. In those cases when complete wire routing was not achieved, the Cell Matrix Compiler did not generate a layout.

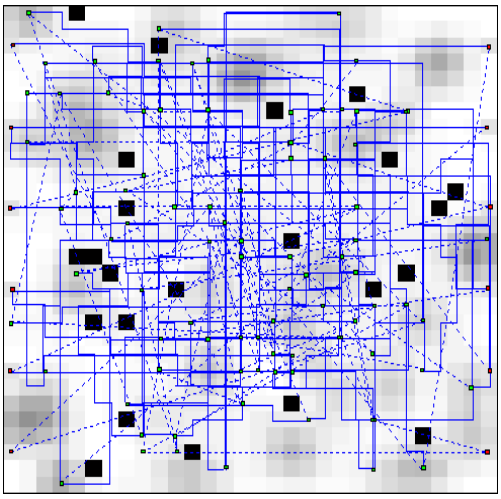
Verification

The purpose of the verification process was to measure the success rate of Cell Matrix

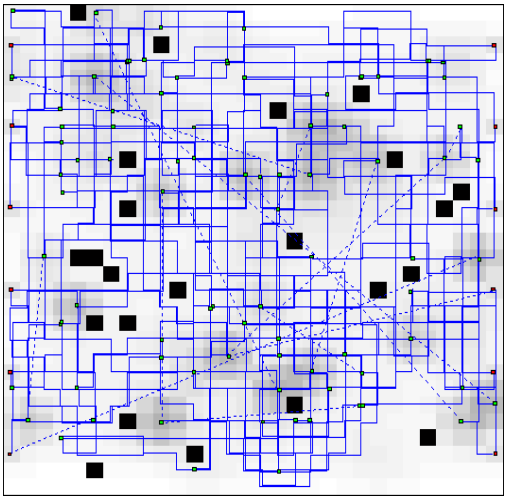
compilation with a wide variety of input netlists and available circuit areas. For greater control of the netlist complexity, I developed a synthetic netlist generator, which produced netlists with an arbitrary number of elements connected randomly so that each gate was connected to two or three other gates.

Through experimentation, I found several schedules resulting in fast and efficient placement and routing of a wide variety of circuit sizes (up to 2000 gates).

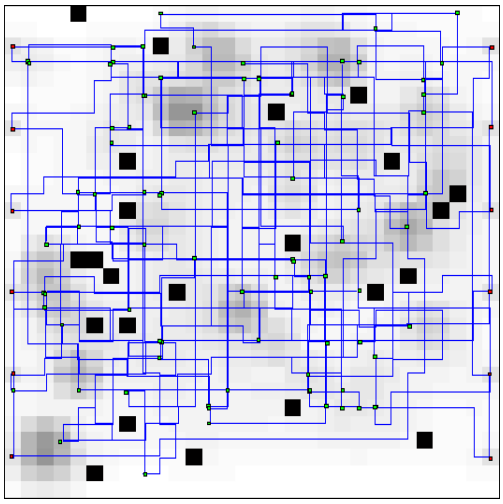
Figure 7 illustrates the routing iterations of a 100-gate synthetic circuit with 230 wires on a 30x30 Cell Matrix with a 3 percent bad cell rate using the schedule in Table 4. plots the percentage of routed wires as a function of iteration number averaged over ten runs, with netlists and bad cell maps of similar complexities generated randomly for each run. All runs but two completed within the 120-iteration limit. The unsuccessful compilations suffered from premature stage transitions and would have taken a longer time to converge. To solve the problem, either more chip space must be allocated (and then returned if made available after optimization) or the compilation must be repeated with a different random number generator seed.

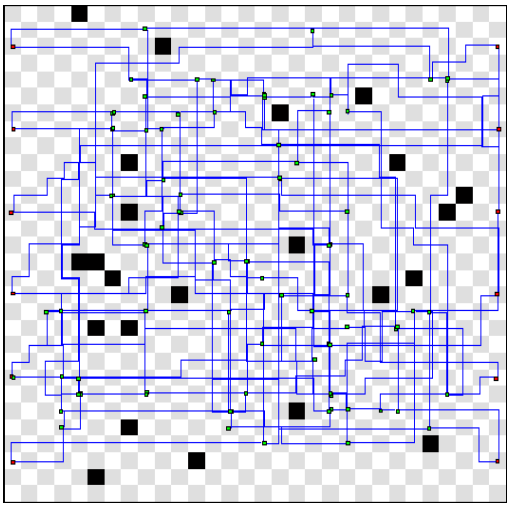


a)



b)





c)d)

Figure 7. Routing of a 100-gate 230-wire netlist on a 30x30 Cell Matrix with 3% bad cell rate. The dashed lines indicated unrouted wires. In (a), (b), and (c), the shades of gray indicate the neighborhood quality. Darker areas denote less desirable placement candidates. After three iterations (a), 48% of wires are routed. After 25 iterations (b), 91% of wires are routed. After 58 iterations (c), a first complete routing is achieved with the mean wire length of 10.4 After 90 iterations, the routing objectives are achieved. The mean wire length is 7.9 cells.

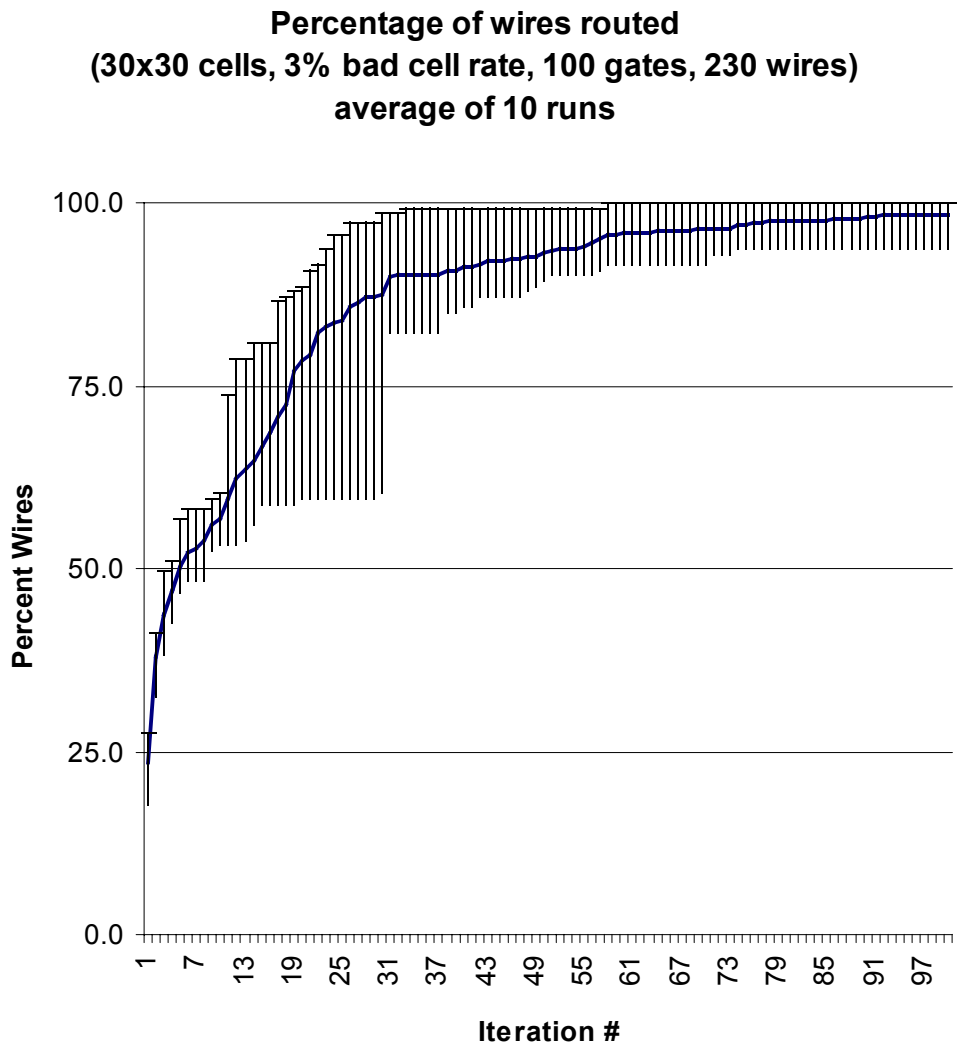


Figure 8. Percentage of wires routed successfully at each iteration. The error bars indicate the best and the worst cases, starting with ten random 100-gate netlists and defective cell positions on a 30x30 Cell Matrix.

Success Rate

A serious challenge for the Cell Matrix Compiler and other hardware compilers is the difficulty of predicting success in the general case. Various heuristics may be developed to predict whether the allocated chip area can accommodate a given schematic. However, the presence of

defective cells, complex matrix geometry, the degree of netlist connectivity, and the positions of terminal gates make this an intractable problem. A possible solution is to provide abundant chip resources, and let the compiler route and optimize the netlist. When the optimization is complete, the unutilized chip area can be reallocated for other use. In any case, a degree of user supervision may be required.

Another weakness of the Cell Matrix Compiler arises from premature stage transitions. In this case, suboptimal algorithm settings may result in unusually low convergence rates. This becomes a problem when the allocated Cell Matrix area approaches the minimal area sufficient for the implementation of the design. For example, if the general macroplacement is not optimal and microplacement is triggered, the adjustments of positions of groups of cells may take a longer time when the algorithm is performing the optimization of individual cells' placements (microplacement). The developer may greatly increase the effectiveness of the algorithm by detecting such premature transitions and adjusting the stage transition parameters.

FUTURE WORK

Parallel Implementation

Developing a parallel version of the Cell Matrix Compiler is a possible next step. The current Cell Matrix Compiler implements a highly parallelizable algorithm. Implemented in a parallel framework, its efficiency would increase almost proportionally to the number of processors dedicated to the problem until the number of processors approached the number of gates to be placed.

Self-Routing Cell Matrix

A more intriguing and perhaps highly challenging task, however, is to implement a self-routing Cell Matrix module, that is a Cell Matrix that will program itself according to logic design specification. Although this task is mentioned in this thesis, the problem is not addressed directly and extensive development may be required.

Extending into Alternative Topologies

This thesis outlines the implementation of a placement and routing algorithm for a single Cell Matrix topology in such a way that extending it to other Cell Matrix architectures proposed by Macias in [Macias, N. The PIG paradigm: The design and use of a massively parallel fine grained self-reconfigurable infinitely scalable architecture. In Proceedings of The First NASA/DOD Workshop on Evolvable Hardware, (1999), 175-80. Ed Stoica, A., Keymeulen D., Lohn J.] is conceptually straightforward. Thus, extending the algorithm to work with 6-connected hexagonal two-dimensional cells or 8-connected cubic three-dimensional cells may be another future project.

Control Mode Specification and Compilation

In control mode (C mode), Cell Matrix elements reprogram each other. This functionality

is new in Cell Matrix and is not supported by conventional logic design and schematic entry tools. A future project could develop symbolic or graphical representation of self-configurable logic and extend the Cell Matrix compiler to accept and process this new notation.

REFERENCES

15. Betz, V. and Rose, J. VPR: A new packing, placement, and routing tool for FPGA Research. In *Proceedings of International Workshop on Field Programmable Logic and Application,* 1997.

16. Blank, T. A survey of hardware accelerators used in computer-aided design. *IEEE Design and Test*, 1, 3 (1984) 21-39.

17. Brixius, N. and Anstreicher, K. *The Steinberg Wiring Problem*. University of Iowa, 2001.

18. Cai, H. *Gridless Routing System for Macro-Cell Design.* Delft University of Technology, Delft, The Netherlands. Published in Zobrist, G. *Routing, Placement, and Partitioning*, chapter 2. Alex Publishing Corporation, 1994.

19. Chen, W. and Wang, K. *Placement.* Department of Electrical Engineering and Computer Science, University of Illinois at Chicago. Published in Zobrist, G. *Routing, Placement, and Partitioning*, chapter 5. Alex Publishing Corporation, 1994.

20. Durbeck, L. and Macias, N. The Cell Matrix: An architecture for nanocomputing. *Nanotechnology,* 12 (2000), 217-230.

21. Durbeck, L. and Macias, N. Defect-tolerant, fine-grained parallel testing of a Cell Matrix. *SPIE ITCom,* Series 4867 (2002), 71-85. Ed Schewel, J., James-Roxby, P., Schmit, H., and McHenry, J.

22. Garey, M. and Johnson, D. The rectilinear Steiner tree problem is NP-complete. *SIAM Journal of Applied Mathematics*, 32 (1977) 826-834.

23. Hightower, D. A solution to line-routing problems in the continuous plane. In *Proceedings of the 6th Design Automation Workshop* (1969), 1-24.

24. Lee, C. Y. An algorithm for path connections and its applications. *IRE Transactions on Electronic Computers*, EC-10 (1961), 346-365.

25. Macias, N. The PIG paradigm: The design and use of a massively parallel fine grained self-reconfigurable infinitely scalable architecture. In *Proceedings of The First NASA/DOD Workshop on Evolvable Hardware,* (1999), 175-80. Ed Stoica, A., Keymeulen D., Lohn J.

26. Moore, E.F. *Shortest Path Through a Maze*. Harvard University Press (1959), 285-292.

27. Ritter, H. and Shulten, K. Kohonen's self-organizing maps: Exploring their computational

capabilities. In *Proceedings of the IEEE International Conference on Neural Networks* (1988), 109-116.

28. Ruben, S. *Computer Aids for VLSI Design*, 2nd edition, 1994. First printed as part of the Addison-Wesley VLSI Systems Series. Addison-Wesley Publishing Company, 1987. Out of print, 1993. Copyright returned to Steven M. Rubin. 1997. http://www.rulabinsky.com/cavd/. July, 2003.

29. Russell, S. and Norvig, P. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 2nd edition (2002).

30. Steinberg, L. The backboard wiring problem: A placement algorithm. *SIAM Review*, 3 (1961) 37-50.

31. Zhang, C. *VLSI Placement*, Institute of Theoretical Electrical Engineering, Unversity Karlsruhe, Kralsruhe, Germany. Published in Zobrist, G. *Routing, Placement, and Partitioning*, chapter 4. Alex Publishing Corporation, 1994.