Optimization of Transistor-Level Floorplans for Field-Programmable Gate Arrays

by

Ryan Fung

Supervisor: Jonathan Rose

April 2002

Optimization of Transistor-Level Floorplans for Field-Programmable Gate Arrays

by

Ryan Fung

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
BACHELOR OF APPLIED SCIENCE

DIVISION OF ENGINEERING SCIENCE

FACULTY OF APPLIED SCIENCE AND ENGINEERING
UNIVERSITY OF TORONTO

Supervisor: Jonathan Rose

April 2002

ABSTRACT

Optimization of Transistor-Level Floorplans for Field-Programmable Gate Arrays

Bachelor of Applied Science and Engineering, April 2002
Ryan Fung
Division of Engineering Science
Faculty of Applied Science and Engineering
University of Toronto

The design and custom hand-layout of FPGAs (Field-Programmable Gate Arrays) is a painstaking process that takes many person-years of effort to complete.

This research builds upon groundbreaking work done at the University of Toronto to work towards the construction of a tool that automatically generates physical layouts from FPGA architectural specifications. In particular, this research focuses on improving the performance of the placement phase of the layout generation engine of that tool.

Various heuristics, some of which make use of specific knowledge of FPGA circuitry, were developed to reduce the area and the amount of wiring resources needed to connect the functional cells within an FPGA layout. Comparisons with the original version, based on practical FPGA architectures, demonstrate the improved layout engine produces layouts about 40% smaller, on average, with about 30% less wiring demand.

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Jonathan Rose, for the guidance he offered and the general wisdom he shared. His enthusiasm is as remarkable as his vision.

I would also like to thank Ketan Padalia, the developer of the tools that serve the immediate foundation for this research, for providing me with the resources I needed to conduct this research. His assistance and support is greatly appreciated.

I would finally like to thank Vaughn Betz of Altera, whose pioneering research Ketan built upon, for the insights and knowledge he has offered me.

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION 1

1.1 MOTIVATION 1
1.2 SCOPE 3
1.3 ORGANIZATION OF THESIS 3

CHAPTER 2 BACKGROUND 5

2.1 OVERVIEW OF FPGA STRUCTURE 5
2.2 FPGA TILES 7
2.3 VPR (VERSATILE PLACE AND ROUTE) 7
2.4 TRANSISTOR AND FUNCTIONAL-CELL NETLIST GENERATION 9
2.4.1 Transistors, Cells, and Ports 9
2.4.2 SRAM Programming-Line Assignments 11
2.4.3 Cell-Level Netlist Details 12
2.4.4 Netlist Annotation 15
2.5 AUTOMATIC CELL-LEVEL PLACEMENT OF FPGA TILES 15
2.5.1 Netlist Reader 17
2.5.2 Optimization Engine 18
2.5.2.1 Initial Placement 18
2.5.2.2 Annealing Schedule 20
2.5.2.3 Layout Cost 20
2.5.2.4 Move Generation and Move Cost Arbitration 22
2.5.2.5 Advantages of Simulated Annealing 26
2.5.3 SRAM Regularization 27
2.6 AUTOMATIC INTERCONNECTED BLOCK PLACEMENT 28

CHAPTER 3 NEW OPTIMIZATION INFRASTRUCTURE 30

3.1 GOAL 30
3.2 RESTRUCTURING OF GENERAL OPTIMIZATION FLOW 31
3.2.1 Multi-Phase Optimization 31
3.2.2 Tile Compaction Between Reheat Anneals 32
3.3 MAJOR MODIFICATION OF EXISTING FEATURES IN SIMULATED-ANNEALING-BASED OPTIMIZATION ENGINE 34
3.3.1 Initial-Placement Modifications 34
3.3.2 Annealing-Schedule Modifications 34
3.3.3 Layout Cost Modifications 35
3.3.4 Move Generation and Move Cost Arbitration Modifications 37
3.4 OVERVIEW OF NEW HEURISTICS AND COSTS 39
3.4.1 Tile-Size Cost 40
3.4.2 Tile-Slope Cost 40
3.4.3 Wire-Overuse Cost 40
3.4.4 Block-Off-Edge Move 41
3.4.5 Compaction Move 42
3.4.6 Block Rotation and Flip 42
3.4.7 Block Equivalent-Pin Swap 43
3.4.8 Initial Large-Grid Placement 43
3.4.9 SRAM Reweave 44

CHAPTER 4 OPTIMIZATION TECHNIQUES AND COSTS 46

4.1 EXPERIMENTAL BENCHMARK SET 46
4.2 PSEUDO CODE 47
4.3 COST-BASED OPTIMIZATIONS 47

4.3.1 Tile-Size Cost *48*
4.3.2 Tile-Slope Cost *51*
4.3.3 Wire-Overuse Cost *54*
4.3.3.1 Congestion Model 55
4.3.3.2 Congestion Cost 57
4.4 MOVE-BASED OPTIMIZATIONS 64
4.4.1 Block-Off-Edge Move *64*
4.4.2 Compaction Move *69*
4.4.3 Block Rotation and Flip (Cell Template Swap) *77*
4.4.4 Block Equivalent-Pin Swap *81*
4.5 OTHER OPTIMIZATIONS 85
4.5.1 Initial Large-Grid Placement *85*
4.5.2 SRAM Reweave *90*
4.6 CUMULATIVE EFFECT OF OPTIMIZATION TECHNIQUES AND COSTS 95

CHAPTER 5 CONCLUSION 98

5.1 FINAL RESULTS 98
5.2 FUTURE WORK 99

APPENDIX A LAYOUT OPTIMIZER FLOW 104

APPENDIX B MOVE GENERATOR DETAILS 106

APPENDIX C GRAPHICAL ILLUSTRATION OF ATL CELL PLACEMENT RUNS 108

REFERENCES 121

LIST OF FIGURES

FIGURE 2-1 HIGH-LEVEL VIEW OF UPPER-LEFT CORNER OF A TYPICAL FPGA 6
FIGURE 2-2 ILLUSTRATING THE TILE-ABILITY OF AN FPGA TILE 8
FIGURE 2-3 VPR CAD FLOW 9
FIGURE 2-4 VPR_LAYOUT CAD FLOW 10
FIGURE 2-5 ILLUSTRATION OF PORT CONSTRAINTS 11
FIGURE 2-6 ILLUSTRATION OF THE PLACEMENT/ROUTING GRID (BASED ON ROUTING TRACKS) 14
FIGURE 2-7 ATL CAD FLOW 16
FIGURE 2-8 ILLUSTRATION OF AN FPGA TILE AND TILE-ABILITY BASED ON THE PORT ARRANGEMENTS 17
FIGURE 2-9 ILLUSTRATION OF A NET AND ITS BOUNDING BOX 21
FIGURE 2-10 LEGAL MOVE EXAMPLE 23
FIGURE 2-11 ILLEGAL MOVE EXAMPLE 24
FIGURE 2-12 ILLUSTRATING REGULAR ARRANGEMENT OF SRAM CELLS 27
FIGURE 3-1 ILLUSTRATING TILE COMPACTION BETWEEN REHEAT ANNEALS 33
FIGURE 3-2 ILLUSTRATION OF A NET AND ITS BOUNDING BOX 36
FIGURE 3-3 MOVE GENERATOR MODIFICATION: EXAMPLE 1 38
FIGURE 3-4 MOVE GENERATOR MODIFICATION: EXAMPLE 2 38
FIGURE 4-1 RINGS OF EQUAL TILE-SLOPE COST 53
FIGURE 4-2 RINGS OF EQUAL TILE-SLOPE COST INCONGRUENT WITH TILE BOUNDARY AND RESULTING PLACEMENT 53
FIGURE 4-3 COARSENESS OF CONGESTION GRID AS A FUNCTION OF TILE COMPACTNESS 58
FIGURE 4-4 EXAMPLE OF CELLS LIMITING COLLAPSE AND SPACE THAT CAN ACCOMMODATE THEM 66
FIGURE 4-5 EFFECT OF BLOCK-OFF-EDGE MOVE ON RANGE LIMIT 67
FIGURE 4-6 ALL BLOCKS MUST MOVE RIGHT TO PERMIT TILE SHRINKAGE 70
FIGURE 4-7 MOVE LIKELY REJECTED BECAUSE OF UNFAVOURABLE IMPACT ON OTHER OPTIMIZATION GOALS. 70
FIGURE 4-8 THE "CORRECT" MOVE SEQUENCE WILL SUCCESSFULLY MOVE THE BLOCKS OFF THE EDGE. 71
FIGURE 4-9 PROPOSED MOVES OF FIRST COMPACTION MOVE TYPE 72
FIGURE 4-10 PROPOSED MOVES OF SECOND COMPACTION MOVE TYPE 72
FIGURE 4-11 PROPOSED MOVES OF THIRD COMPACTION MOVE TYPE 73
FIGURE 4-12 PROPOSED MOVES OF FOURTH COMPACTION MOVE TYPE 73
FIGURE 4-13 T-SHAPED ARRANGEMENT PRODUCED BY COMPACTION MOVE TYPES (1) AND (2) 74
FIGURE 4-14 X-SHAPED ARRANGEMENT PRODUCED BY COMPACTION MOVE TYPES (3) AND (4) 74
FIGURE 4-15 INITIAL LARGE-GRID PLACEMENT 87
FIGURE 4-16 ILLUSTRATION OF BAD PROGRAMMING-LINE ASSIGNMENT 91
FIGURE 4-17 EXAMPLE OF AN SRAM REWEAVING 93
FIGURE 4-18 ILLUSTRATION OF GOOD PROGRAMMING LINE ASSIGNMENT ACHIEVED BY SRAM REWEAVE 94

LIST OF GRAPHS

GRAPH 4-1 FREE SPACE, RUN TIME, WIRELENGTH VS. TILE-SIZE COST FRACTION 51
GRAPH 4-2 WIRELENGTH, RUN-TIME, FREE-SPACE, OVERUSE VS. OVERUSE COST FACTOR 63
GRAPH 4-3 FREE SPACE, RUN-TIME, WIRELENGTH VS. COST-DIFFERENCE DIVISOR 69
GRAPH 4-4 FREE SPACE, RUN-TIME, WIRELENGTH VS. AVERAGE NUMBER OF TILE COMPACTION MOVES PER TEMPERATURE 76
GRAPH 4-5 RUN TIME, WIRELENGTH VS. TEMPLATE MOVE FREQUENCY 81
GRAPH 4-6 LEGAL MOVE PERCENTAGE, WIRELENGTH VS. AREA_FUDGE_FACTOR 86

LIST OF TABLES

TABLE 4-1 TECHNIQUE AND COST SUMMARY 46
TABLE 4-2 TILE-SIZE COST FRACTION COMPARISON 50
TABLE 4-3 EFFECT OF BLOCK-OFF-EDGE MOVE DIVISOR 69
TABLE 4-4 COMPACTION-MOVE EFFECT 75
TABLE 4-5 EFFECT OF TEMPLATE MOVES 80
TABLE 4-6 EFFECT OF INITIAL LARGE-GRID PLACEMENT WITHOUT RUN-TIME ADJUSTMENT 89
TABLE 4-7 EFFECT OF INITIAL LARGE-GRID PLACEMENT WITH RUN-TIME ADJUSTMENT 90
TABLE 4-8 EFFECT OF SRAM REWEAVING 94
TABLE 4-9 RESULTS OF THIS RESEARCH COMPARED WITH THOSE OF INITIAL ATL (TILE-AREA FACTOR: 1.4) 96
TABLE 4-10 RESULTS OF THIS RESEARCH COMPARED WITH THOSE OF INITIAL ATL (MINIMUM TILE AREA) 96

Chapter 1
INTRODUCTION

1.1 MOTIVATION

There are many competing options available for digital circuit implementation. Field-programmable gate arrays (FPGAs), mask-programmable gate arrays (MPGAs), standard-cell implementations, and custom layout are representative of the spectrum of possibilities available. The spectrum represents a continuous tradeoff between ease-of-engineering (rapid time to market), and area/speed/power/price performance. FPGAs offer ease and speed of development at the cost of sacrificing the area/speed/power/price performance achievable by the more involved and mass-produced alternatives. FPGAs can be programmed to implement different digital circuits and be re-programmed if necessary. This flexibility and ease of engineering contributes to lower non-recurring engineering costs and faster time-to-market, which have made FPGAs popular for development (prototyping) and low- to mid-volume production. [1 and 2] Better designed FPGAs, that reduce the area/speed/power performance gap and can be produced cheaply, can help increase the precise low- to mid-volume point where FPGAs remain a viable implementation alternative.

With good FPGA design being of paramount importance, the primary factors that contribute to this goal should be considered. The overall logic and routing architecture, the transistor-level design, the physical layout, and the computer-aided-design (CAD) tools, which program the FPGAs, are all important factors of FPGA design. [1]

Of course, these factors are highly interrelated. Transistor-level design decisions will affect layout alternatives, and logic and routing architecture changes will impact the heuristics used in the CAD tools which program FPGAs to operate at high speeds. Ideally, all these interrelated factors can be considered at every stage in the design process. For example, when determining logic and routing architecture, ideally CAD tool capability, and circuit and layout considerations could be evaluated as each of the design choices are made.

Over the past decade, research has been conducted at the University of Toronto that has tried to link three of the four primary factors indicated above. [3] This research has taken the approach of creating a CAD tool that automates the exploration of FPGA architectures. This research has shown the area/speed performance benefits realized by considering transistor-level circuit design and CAD tools, while making architectural decisions.

Even more recently, additional research at the University of Toronto has attempted to incorporate the consideration of physical layout in the context of automated FPGA-architecture evaluation. [1]

Currently, considerable FPGA design and engineering effort is expended considering the implications of layout, as well as actually laying out an FPGA. Since key structures in the FPGA are replicated over-and-over, custom-design hand-tuning is very important and is often required to lay out the respective structures for good area/speed performance. Automated FPGA architecture evaluation that considers physical layout can greatly reduce the initial design effort. Furthermore, the tool constructed in [1], while evaluating architectures, automatically considers transistor-level circuit designs and

physical layouts; these designs and layouts can serve as starting points, reducing the cost and time spent in the FPGA design cycle. Clearly, the realization of both these advantages rely on the tool properly considering layout constraints and effectively producing adequate-quality FPGA layouts based on the considered architecture.

This work is an attempt to move closer to the goals of considering physical layout when exploring architectures and providing good-quality physical layouts for given architectural specifications.

1.2 SCOPE

This research sought to improve the automated layout mechanism used by the expanded tool created at the University of Toronto [1] that attempts to consider physical layout when evaluating FPGA architectures.

In this research, it is shown that various heuristics, some of which make use of knowledge of the circuitry within an FPGA, can be used to improve the layout performance of the tool.

The scope of this research was limited to creating mechanisms for optimizing placements of the functional cells (multiplexers, buffers, switches, etc.) within an FPGA. This specifies a detailed floorplan for the transistor-level layout. Exact transistor-level layout and routing between functional cells is left to concurrent and future work. At the time of this writing, work on both these fronts was being conducted at the University of Toronto.

1.3 ORGANIZATION OF THESIS

Chapter 2 presents background information and details about previous work that

are relevant to this research. Chapter 3 describes important infrastructure changes and modifications made to the FPGA tile layout tool (which serves as the foundation for this research); it also introduces the major heuristics developed to improve layout quality that rely on these general enhancements. Chapter 4 describes the major heuristics in more detail; experimental results are presented to illustrate the effects of each of the relevant techniques. Chapter 5 summarizes the effect the developments of this research have on the quality of the layouts produced by the FPGA tile layout tool. Appendix A and B provide pseudo-code details of important aspects of the implementation. Appendix C presents illustrations of layouts produced by the tool at various stages during layout optimization.

Chapter 2
BACKGROUND

The first section of this chapter briefly describes the structure of FPGAs. The second section describes the "tiles" of logic and routing that are replicated to produce an FPGA. The third section provides information regarding the generation of transistor-level and functional cell-level netlists of the FPGA tiles considered in this research. The fourth section provides information regarding the initial state of the cell-level layout engine that serves as a foundation for this research. The final section is a brief outline of some of the previous work that is relevant to the placement and compaction problems central to this work.

2.1 OVERVIEW OF FPGA STRUCTURE

An FPGA is a re-programmable digital device that can be configured and re-configured to implement different digital circuits. Three components make up the FPGAs considered in this work: logic blocks, programmable routing, and I/O pads. The I/O pads lie in a ring surrounding the FPGA core (which contains the logic and programmable routing). The core is often created by replicating a "tile" of logic (the logic block) and its surrounding routing, many times, both horizontally and vertically across the core of the chip. This general structure is illustrated in Figure 2-1.

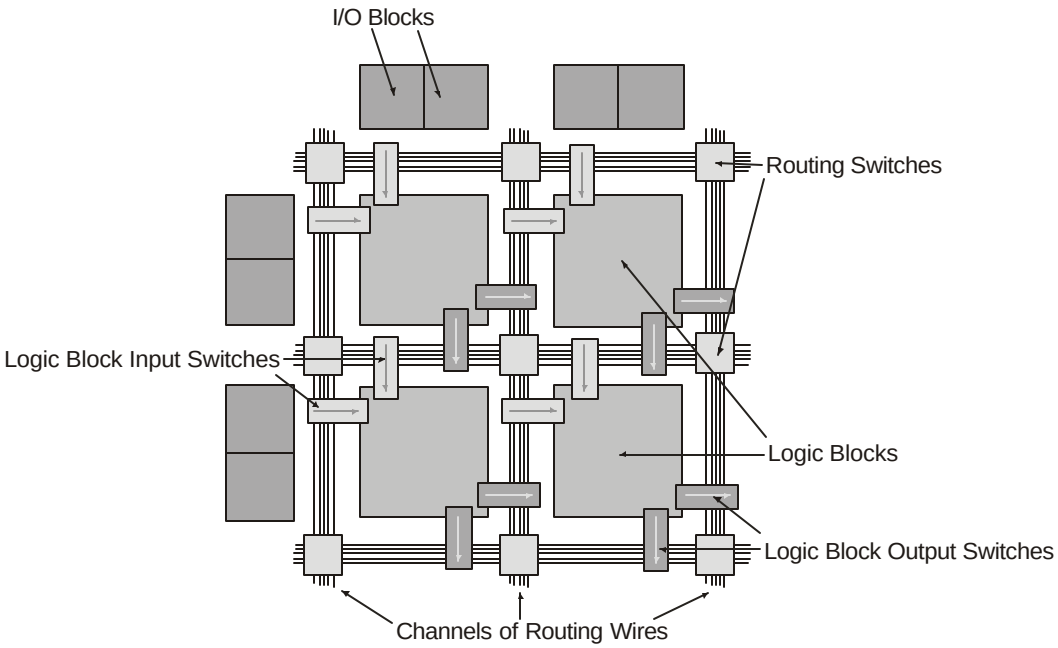


Figure 2-1 High-level View of Upper-Left Corner of a Typical FPGA

Bundles of staggered wires, spanning different distances, run in channels between the logic blocks. Programmable switches are used to transfer signals from one wire to another and to connect the logic blocks to wires. Static-RAM (SRAM) cells are the entities programmed during FPGA programming. These, in turn, control, amongst other things, the programmable switches – which may be simple pass transistors, or tri-state buffers for greater drive strength.

A logic block consists of one or more basic logic elements (BLEs). These BLEs generally consist of a programmable look-up table (LUT) and the flip-flop it optionally drives. The look-up tables are essentially multiplexers with the respective input lines driven by SRAM cells. By hooking the BLE inputs to the control lines of the multiplexer, any N-input Boolean logic function can be achieved (with the appropriate SRAM programming). Logic blocks offer additional levels of flexibility such as optional feedback of BLE outputs to inputs and the optional registering of the BLE output.

The architecture of an FPGA is determined by parameters that affect the building blocks presented above.

2.2 FPGA TILES

As mentioned earlier, the core of an FPGA can be realized by replicating tiles – each tile consists of a single logic block and with its surrounding routing. The careful design and layout of these tiles directly affect the speed/power performance of the FPGA. The replication of this tile many times to form the core of the FPGA means that layout area savings in the fundamental tile largely map to chip area savings; hence, smaller tile area leads to lower overall fabrication cost or, alternatively, more logic in the FPGA for a given fabrication cost (allowing the FPGA to implement larger digital circuits). The automatic layout mechanisms studied in this research are interested in the layout of the FPGA tile. Figure 2-2 (adapted from [1]) illustrates how a core can be constructed by replicating an FPGA tile. Notice how the respective ports on the tile must align to permit the replication. This tile-ability consideration will be discussed later.

2.3 VPR (VERSATILE PLACE AND ROUTE)

VPR is a flexible CAD tool designed at the University of Toronto [3]. VPR performs clustering, placement, and routing of digital circuit netlists for a variety of FPGA architectures (each specified with an architectural description provided to VPR). A simplified view of a VPR CAD flow is illustrated in Figure 2-3.

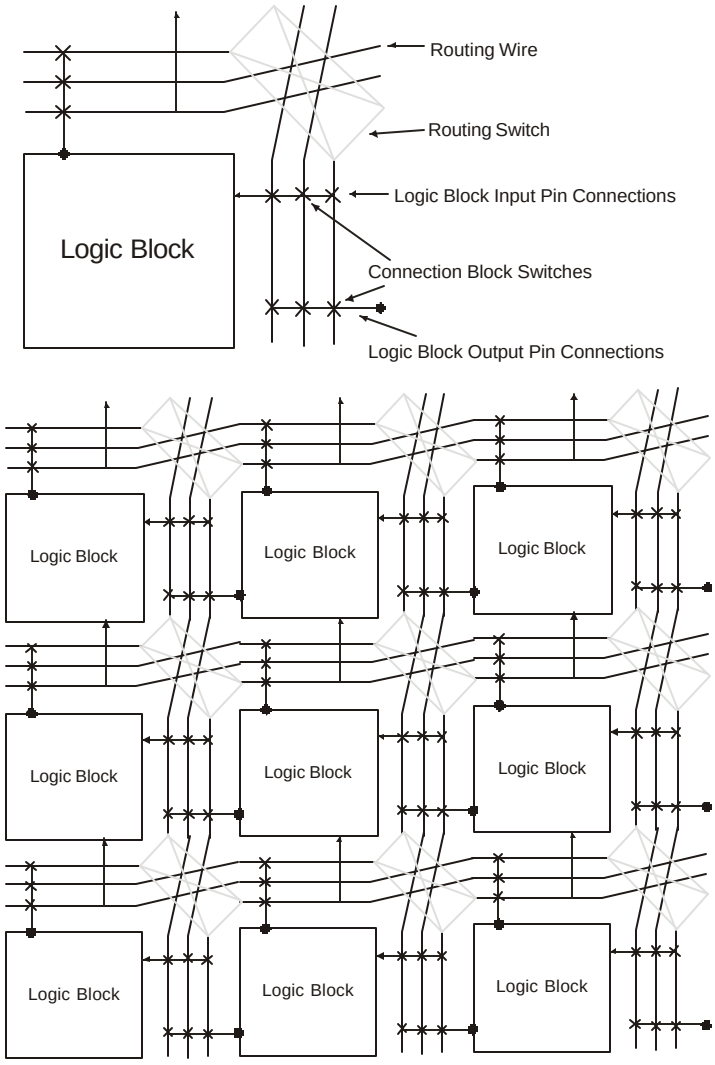


Figure 2-2 Illustrating the Tile-ability of an FPGA Tile

VPR gives designers the ability to observe how architecture changes affect software performance and transistor-level circuit design. Using the architectural description, VPR synthesizes an FPGA out of a pre-determined set of transistor-level structures. It creates area and delay estimates based on the types, sizes, and delay characterization of these structures. VPR does not consider the physical layout of these structures, but it does consider the final route-ability and potential timing performance of

the FPGAs it creates based on its digital-circuit place-and-route engine.

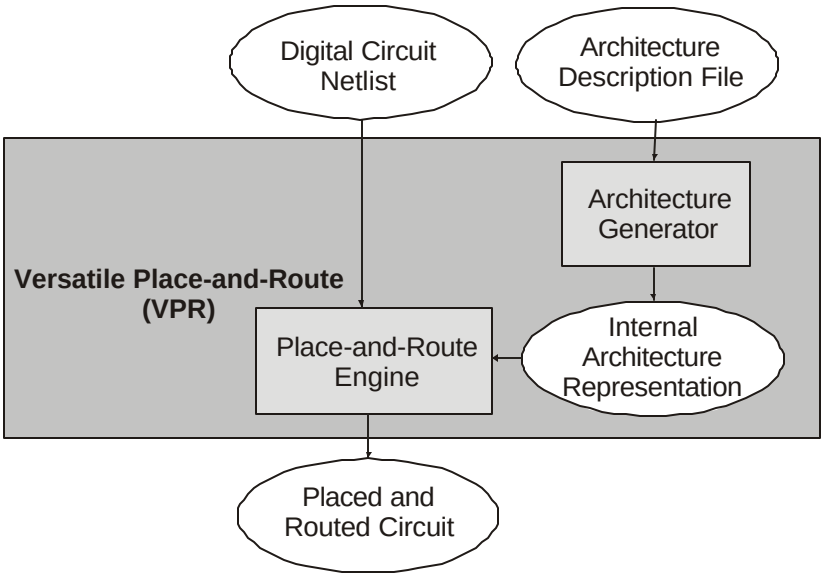


Figure 2-3 VPR CAD Flow

2.4 TRANSISTOR AND FUNCTIONAL-CELL NETLIST GENERATION

2.4.1 TRANSISTORS, CELLS, AND PORTS

VPR_LAYOUT designed at the University of Toronto [1] extends VPR by generating a transistor-level and cell-level circuit specification of an FPGA tile based on an architectural description read into VPR. VPR_LAYOUT primarily operates on VPR's internal architecture representation, which is created based on an architecture description. This internal representation contains a description of all the programmable routing and the connectivity to the logic blocks. VPR_LAYOUT also assumes certain transistor-level implementations of the transistor-level structures (functional cells), considered by VPR, when building the transistor-level and cell-level netlists of an FPGA tile. The functional cells include: SRAMs, buffers, multiplexers, flip-flops, LUTs, pass-transistor routing

switches, and tri-state buffers. These are described in [1]. The cell-level netlist is based on these cell primitives. The VPR_LAYOUT CAD flow is illustrated in Figure 2-4.

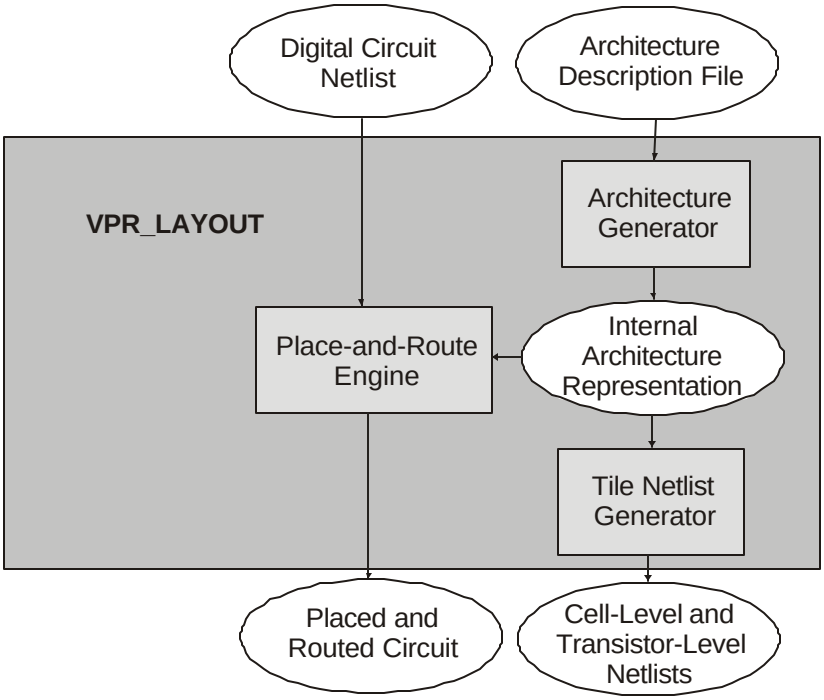


Figure 2-4 VPR_LAYOUT CAD Flow

Besides functional cells and transistors, the netlists generated by VPR_LAYOUT also include ports. Ports are the end-points of the wire segments at the edge of an FPGA tile that cross into an adjacent tile. Since tiles must be tile-able vertically and horizontally, port placement and constraints are important considerations for tile legality. For example, a port on the left side of a tile must have a matching port at the same vertical location on the right in order for the respective signal to flow between the tiles. Each port is assigned a pairing number and a tile-side character to facilitate this. The same unique pairing number is assigned to each member of a corresponding pair of ports which must line up vertically or horizontally on opposite sides of the tile. The tile-side character specifies whether the port should be placed on the top ('T'), bottom ('B'), left

('L'), or right ('R') side of the tile. Necessarily, if one member of a matching port pair is constrained to the top of the tile, the other member of the pair will be constrained to the bottom. This is illustrated in Figure 2-5. Notice how ports with the same pairing number are placed opposite one another so that, when the tile is tiled, they connect. Notice also how the tile-side characters are respected.

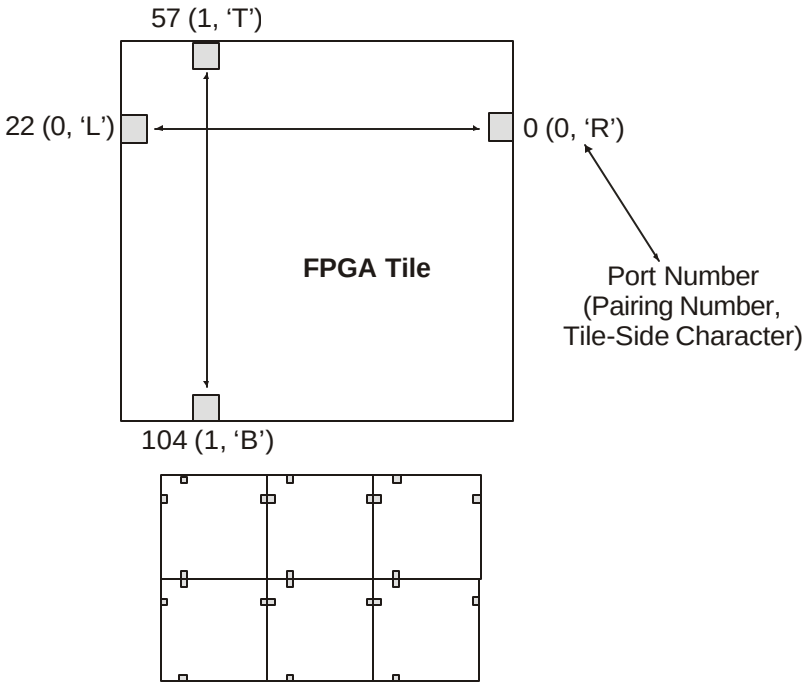


Figure 2-5 Illustration of Port Constraints

2.4.2 SRAM PROGRAMMING-LINE ASSIGNMENTS

VPR_LAYOUT is also responsible for making SRAM programming line assignments to the SRAM cells. It creates the same number of word and bit lines (to achieve a square memory array for the tile).

It turns out that word and bit line assignments are arbitrary. To recognize this, consider that it is a CAD tool which generates the streams of data fed into the SRAM cells. Different data streams program the SRAM cells to different values and

consequently different digital circuits are implemented in the FPGA. Since the CAD tool can be adjusted to produce data streams suitable for any arbitrary programming line assignment, it does not matter which SRAMs connect to which word and bit lines. Acknowledging this fact, the netlist generator assigns word and bit lines randomly to the SRAMs to avoid embedding netlist information implicitly in the SRAM programming-line connectivity. For example, when a programmable multiplexer is generated, its respective SRAM cells will be generated at the same time. If all these cells were tied together with the same word line, implicit association of the SRAMs would result. By randomly making SRAM assignments, such implicit embedded knowledge is removed, reflecting the true arbitrariness of the word and bit line assignments. This leaves room for the layout optimizer to independently leverage the arbitrariness of word and bit line assignments for its own purposes – with no "hints" skewing experimental results.

2.4.3 CELL-LEVEL NETLIST DETAILS

The cell-level netlist is the one placed during the layout creation studied in this research; therefore, details of its generation are especially important for this research. Cell area was calculated by VPR_LAYOUT based on the total transistor area used to construct the cell. An extra 50% of that area was added to allow for intra-cell routing, and to account for spacing around the edges necessary to achieve adequate well spacing (and accommodate other design rules). This would allow cells to directly abut one another. Cell height and width was set to make the cell as square as possible and to upper bound the estimated cell area. Precisely, VPR_LAYOUT calculated cell height and width by starting with a cell area specified in units of minimum-width transistor areas. Therefore, the cell heights and widths were in units of (minimum-width transistor areas)$^{1/2}$.

For this research, this specification of cell heights and widths based on units of (minimum-width transistor areas)$^{1/2}$ was abandoned. Instead, cell heights and widths are specified in terms of the number of routing tracks spanned. Basing cell heights and widths (and the corresponding placement grid) on units of routing tracks is more meaningful in the sense that cell pins can be precisely positioned at locations where routing contacts will have to be made. Furthermore, the size of ports (which are implicitly defined by VPR_LAYOUT as being 1x1) make much more sense when a routing track grid is used because ports essentially represent a single routing segment (one routing track by one routing track in size). To achieve this "more realistic" specification of cell heights and widths, the total transistor area of the cell (specified in units of minimum-width transistor areas) is multiplied by 2.25 (routing tracks)^2/(minimum-width transistor area). The resulting value is an estimate of the number of routing grid (placement grid) squares occupied by the cell. Refer to Figure 2-6 for more details and further explanation of the terminology. The 2.25 multiplier is designed to account for layout-bloat due to intra-cell routing and design rule accommodation. It is a rough estimate based on observation of several custom layouts of digital logic. The accuracy of this estimate can be examined in future work by conducting more detailed studies. Nevertheless, this estimate is not extremely important because once actual layouts of the cells are produced (work on this front is being conducted at the University of Toronto), the related area information can be directly embedded in the netlist without resorting to area estimates. Furthermore, the heuristics developed in this research do not depend on the specific heights and widths of cells but on the general problem of placing FPGA-tile cells and ports on a placement/routing grid.

There are a few details that should be noted when considering the new placement grid, depicted in Figure 2-6. The placement/routing grid is sized based on the minimum distance between contacts (because of the metal halo around them). That way, the router can route different signals in adjacent grid squares – placing contacts to hop between metal layers whenever necessary. When different design rules are present in different metal layers, the most conservative rules (those that produce routing grid squares of the largest size, in square microns) are used so that the same grid can be applied to all layers.

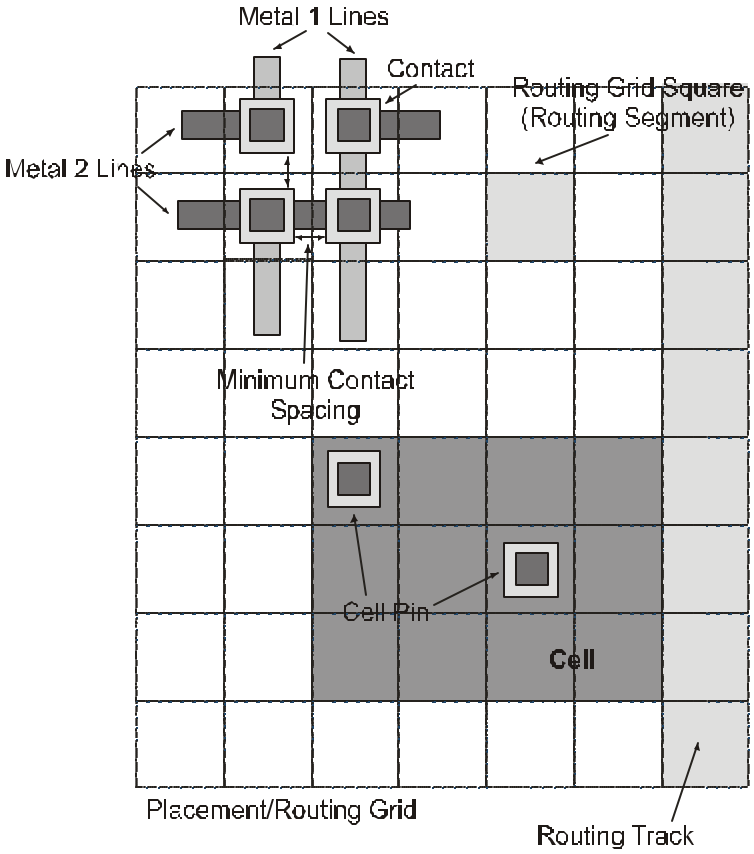


Figure 2-6 Illustration of the Placement/Routing Grid (Based on Routing Tracks)

Each cell has pins distributed throughout its area that represent the cell "ports" to which connections can be made. VPR_LAYOUT originally positioned these pins around the perimeter of the cell. Since connections to the cell can be made through vias to points

within the cell, this research modified VPR_LAYOUT to randomly spread the pins throughout the cells instead of sticking to the perimeter. This avoids routing congestion around the cell perimeter (especially in the form of "via towers") and is more representative of the fact that signals within cells are not needed and produced only at the perimeter.

2.4.4 NETLIST ANNOTATION

VPR_LAYOUT is also responsible for annotating its produced netlists with information that guides placement. It does this by providing details of the roles of various cells, ports, transistors, and connections in terms of the overall tile structure. For example, SRAM cell pins responsible for SRAM programming (attached to word and bit lines) are annotated with pin class values that indicates their connection to word or bit lines. That way, information regarding which nets correspond to SRAM programming can be extracted from the netlist.

2.5 AUTOMATIC CELL-LEVEL PLACEMENT OF FPGA TILES

ATL (Automatic Tile Layout) designed at the University of Toronto reads in the netlists generated by VPR_LAYOUT and lays out of the corresponding FPGA tile.

To simplify the placement problem, the functional cell-level netlist is placed instead of the transistor-level netlist. By placing functional cells (consisting of a few transistors) instead of individual transistors, a detailed floorplan rather than a transistor-level placement is determined. Nevertheless, because the functional cells consist of

highly related logic, this level of placement should produce results very close to a “flat” placement while avoiding the complexity of individual transistor placement. Refer to Figure 2-7 for the ATL CAD flow.

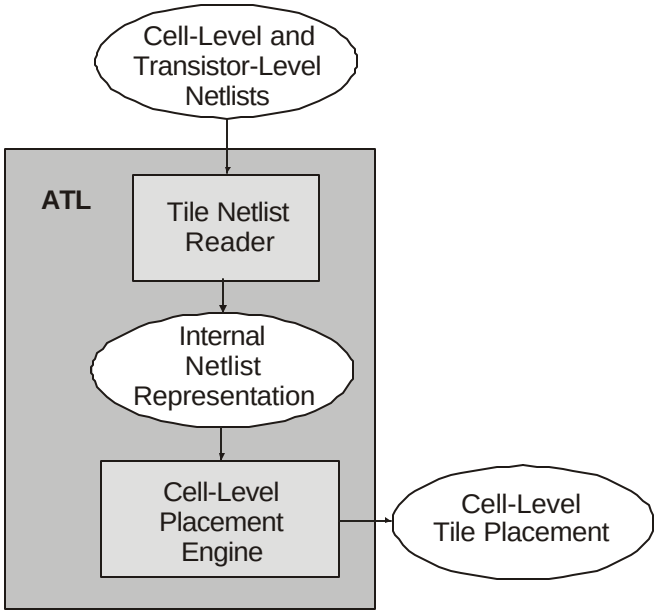


Figure 2-7 ATL CAD Flow

The placement problem involves placement of cells and ports to form a rectangular FPGA tile. The ports are arranged on the perimeter of the tile; they indicate the points of routing connection with adjacent tiles. To permit tile-ability, the respective port constraints must be obeyed. Each port must reside on a particular side of the tile and must be matched with its partner, which is placed opposite it on the tile. That way when the tile is replicated, horizontally and vertically, the port will connect with its partner, connecting the appropriate signals, as described in 2.4.1. Cells are placed so that they do not overlap inside the port perimeter. Cells can be placed anywhere within the port perimeter. Refer to Figure 2-8 for a general illustration of the placement problem.

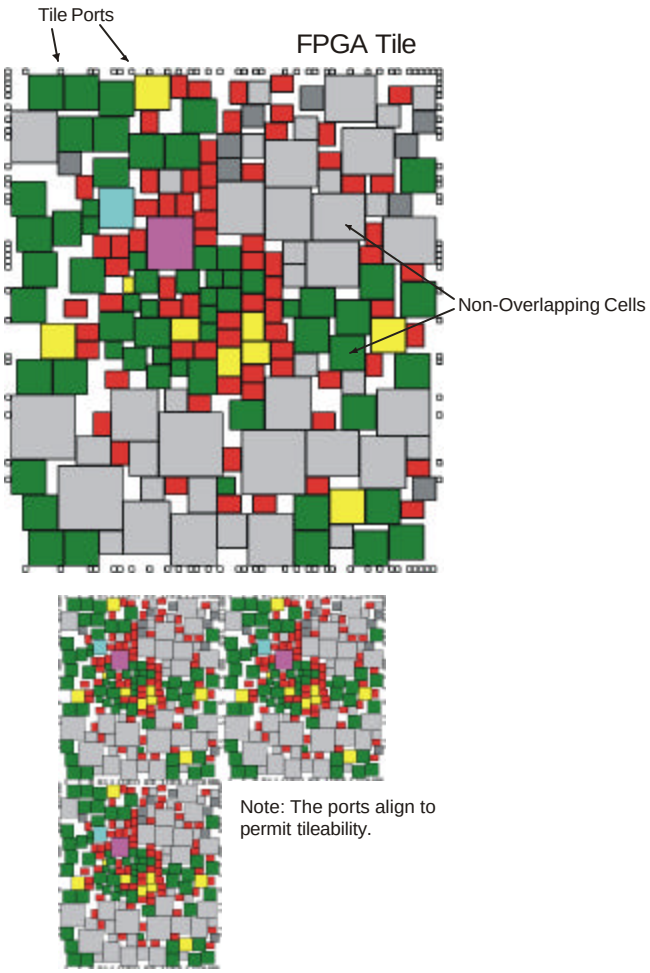


Figure 2-8 Illustration of an FPGA Tile and Tile-ability Based on the Port Arrangements

What follows is a description of the key components of the ATL flow.

2.5.1 NETLIST READER

The netlist reader parses the transistor-level and cell-level netlists, setting up internal netlist representations appropriately. The internal representations are designed to permit the layout optimizer's heuristics to operate on them quickly.

2.5.2 OPTIMIZATION ENGINE

The layout optimization engine is based on the simulated-annealing algorithm. This is a popular optimization technique, a detailed description of which can be found in [2].

The following pseudo-code describes the general annealing algorithm used:

```
BEGIN PLACEMENT ANNEAL
        FOR EACH PLACEMENT TEMPERATURE
                PERFORM A STANDARD RANGE-LIMIT MOVE AND UPDATE LAYOUT COST
                REPEAT FOR DESIRED NUMBER OF MOVES IN TEMPERATURE

                REDUCE TEMPERATURE
                EXIT WHEN EXIT CRITERION MET
        END PLACEMENT TEMPERATURE
END PLACEMENT ANNEAL
```

There are four major components in a simulated annealer adapated to create FPGA tile layouts: (1) initial placement; (2) annealing schedule (initial temperature computation, a temperature update schedule, and exit criterion); (3) layout cost; and (4) move generation and move cost arbitration. Each of these major components will be described next, as they were implemented in ATL, by [1].

2.5.2.1 INITIAL PLACEMENT

The simulated-annealing optimization engine in ATL operates by continually making small changes to the current legal layout – illegal layouts are not considered by the optimization engine. Over time, these incrementally small changes serve to greatly transform the layout and improve the cell and port placement. To begin this process, however, an initial placer is required to create a cell and port layout that can be gradually mutated over the anneal.

The first step in creating a cell and port layout is to determine initial tile

dimensions. ATL calculated the tile area based on the total cell area. The tile width and height was calculated to make the tile as square as possible.

The following equations were used to calculate the tile area:

```
WIDTH(SIDE) = MAX(SQRT(TOTAL_CELL_AREA) * AREA_FUDGE_FACTOR, NUM_PORTS(SIDE))
WIDTH = MAX(WIDTH(LEFT), WIDTH(RIGHT), WIDTH(TOP), WIDTH(BOTTOM))
HEIGHT = WIDTH
```

The AREA_FUDGE_FACTOR is used to bloat the tile area to make space for routing and to give cells more room, primarily for initial placement. A value of 1.4 was selected because it permitted initial placement of all netlists experimented with at the time.

The placement grid ATL was based on, initially, was a minimum-width transistor area grid. Each grid square had the area of a minimum-width transistor (which includes the area of the drain contact and source contact to the first metal layer, but not the gate contact). As mentioned earlier, instead of this, for this research, a routing grid, illustrated in Figure 2-6, is used during placement.

Ports are placed around the perimeter of the tile. The tile is sized large enough to place all the respective ports on the relevant sides ensuring paired ports are opposite one another. ATL, produced by [1], ensured that ports can not overlap one another. Cells are placed randomly in the interior of the tile.

Initial placement is carried out through successive iterations until it succeeds, or the iteration count exceeds a set threshold. In the first iteration, a cell ordering is created so that larger cells are placed first. This heuristic helps initial placement because smaller cells can often fit in the gaps between the larger cells. If an initial placement attempt fails – a legal placement (without cell overlap) can not be created – it is repeated, but the

previous ordering of cell placement is modified so that cells that could not be placed (without causing overlap) in the previous attempt are placed first. Initial placement fails if the iteration count exceeds a threshold value, and the layout generator aborts.

2.5.2.2 ANNEALING SCHEDULE

The simulated annealing algorithm operates by making continuous changes to the current layout at lower and lower "temperatures". The current temperature affects the types of moves which are performed and which moves are rejected. Moves which severely degrade placement quality are more likely to be rejected at lower temperatures. More details will be provided in 2.5.2.4.

An annealing schedule is required to compute an initial temperature, control the temperature updates throughout the anneal, and to finally terminate the anneal based on some criteria. In this sense, the annealing schedule governs the annealing process; it controls the initial state (initial temperature computation), transitions between states (temperature updates), and when the optimizer should terminate (exit criteria). The adaptive annealing schedule presented in [2] is used.

2.5.2.3 LAYOUT COST

The simulated annealing algorithm requires a cost function that is used to compute placement costs that indicate the quality of the current placement. When a move is proposed, the placement cost difference, which would result from the move, is calculated based on the cost function. Based on the cost difference and the current

placement temperature, the move (placement mutation) is either accepted or rejected.

The cost function in ATL was entirely based on wire length. Wirelength cost is based on bounding boxes enclosing net terminals. Each net (electrically connected cell pins, wires, contacts, etc.) has an associated bounding box which tightly encloses all things connected by the net. ATL ignored cell pins and used a bounding box estimate that enclosed the respective connected cells and ports completely (a worst-case estimate). Refer to Figure 2-9 for an illustration of a net with its respective worst-case bounding box. Notice how the bounding box includes the entire 2x2 cell on the left and the entire 4x3 cell on the bottom; hence the box does not tightly enclose the electrically connected contacts – it is expanded to fully encompass all the relevant cells (hence, it assumes worst-case cell pin positions).

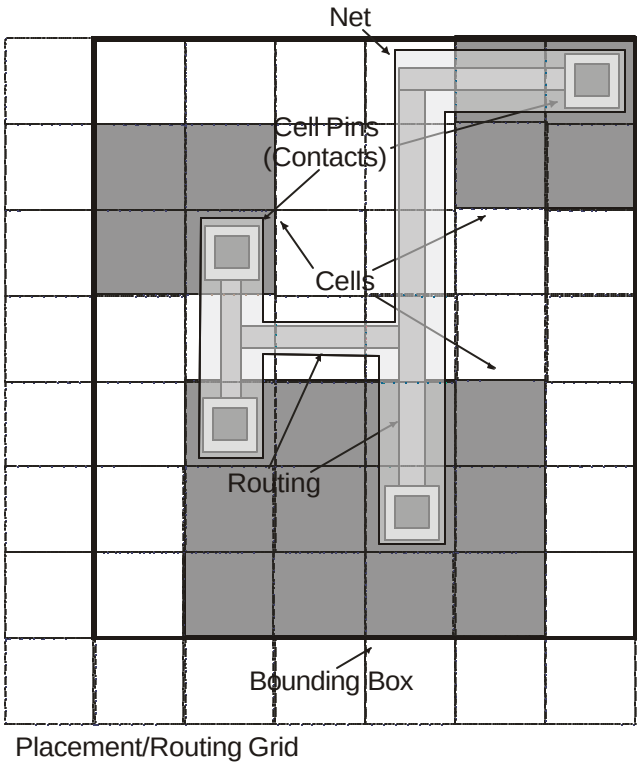


Figure 2-9 Illustration of a Net and Its Bounding Box

The following formula is used to compute wirelength (or bounding-box cost):

$$cost = \sum_{inet=1}^{num\_nets} q(inet) * (bbx(inet) + bby(inet))$$

`bbx` and `bby` are the width and height, respectively, of the bounding-box associated with the respective net. `q` is a multiplication factor that is used to create a wirelength estimate from a bounding-box `height+width`. The value of `q(inet)` is a function of the number of terminals (cell pins and ports) connected by the net. It is 1 for nets with 3 or fewer terminals, and its value increases to 2.79 for nets with 50 or more terminals. [4]

2.5.2.4 MOVE GENERATION AND MOVE COST ARBITRATION

A mechanism is needed to propose and perform the types of mutations and transformations which can be applied to the current placement. This mechanism is called move generation. Move cost arbitration controls whether the moves proposed by the move generator are accepted or rejected (whether the corresponding placement mutation is allowed or prevented), based on the cost change that would result and the current temperature.

The following pseudo-code describes the algorithm used to select a set of cells for moving:

```
PICK PRIMARY CELL TO MOVE AND DESTINATION LOCATION FOR THE CELL BASED ON THE RANGE LIMIT
GATHER OTHER CELLS (GROUP B) THAT THIS PRIMARY CELL DISPLACES
CONSIDER MOVING THESE CELLS INTO THE REGION THAT WOULD BE ABANDONED BY THE PRIMARY CELL
GATHER THE CELLS AROUND THE PRIMARY CELL DISPLACED BY GROUP B CELLS; THESE CELLS, ALONG WITH THE PRIMARY CELL,
        FORM GROUP A
TRY TO SWAP GROUP A AND B WITHOUT CREATING CELL OVERLAP
IF THE MOVE WOULD CREATE AN ILLEGAL PLACEMENT (CELL OVERLAP)
        REJECT PROPOSED MOVE
ELSE IF PROPOSED MOVE IS REJECTED FOR COST REASONS
        DO NOT PERFORM MOVE
ELSE
        PERFORM PROPOSED MOVE
```

It should be noted that if the translation proposed for the primary cell is (? x, ? y), the translation for all the group A cells will be (? x, ? y); that is, they are moved as a unit. Also, the translation (inverse move) proposed for the group B cells will be (-? x, -? y) to ensure they move as a unit to the location abandoned by the group A cells.

Refer to Figure 2-10 for an illustration of a legal move.

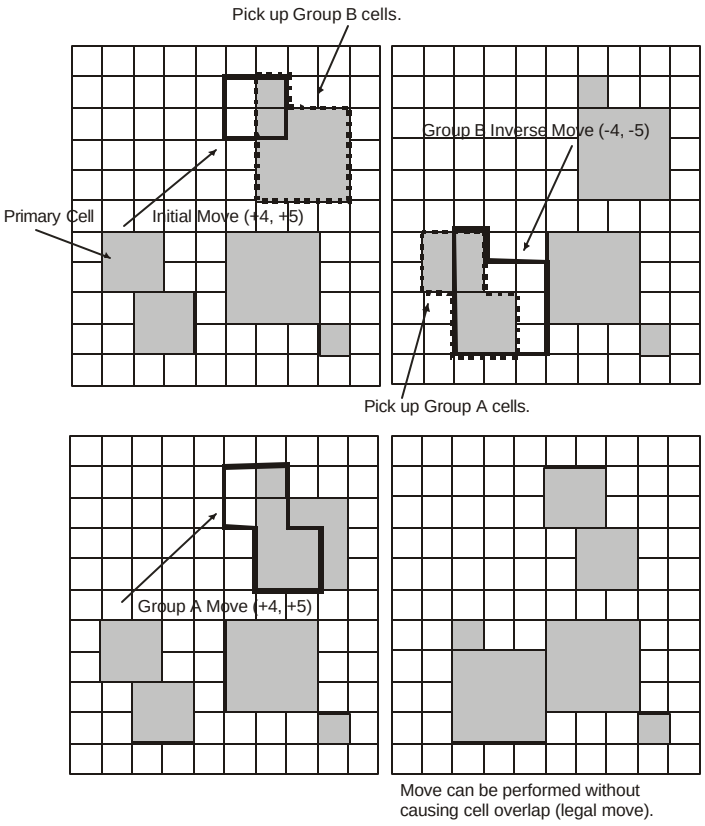


Figure 2-10 Legal Move Example

Refer to Figure 2-11 for an illustration of an illegal move.

A legal proposed move will be accepted or rejected depending on the placement cost difference it produces. A move is accepted if the subsequent placement cost would be less than or equal to the current placement cost. If the subsequent placement cost is greater than the current placement cost, the move is accepted probabilistically. A move

this increases the placement cost by a given amount is more likely to be rejected at lower placement temperatures. Also, during the same placement temperature, a move that increases the placement cost by a greater amount is more likely to be rejected. For precise details regarding this cost arbitration, refer to [1] and [3].

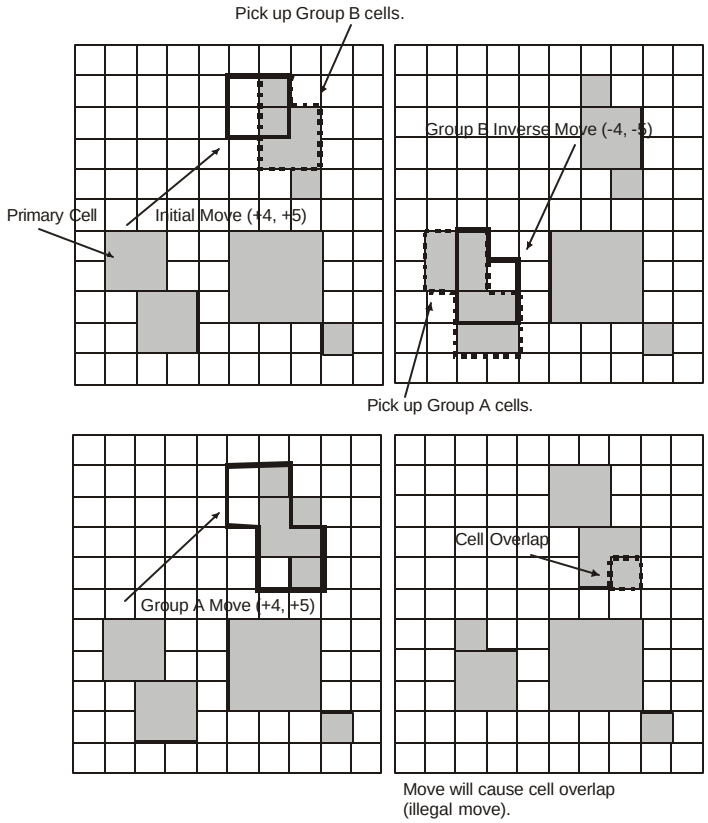


Figure 2-11 Illegal Move Example

There are two levels of move rejection. A proposed move can be rejected because of legality reasons (it would create an illegal placement). Legal proposed moves can also be rejected for cost reasons. Acceptance ratio is defined as the percentage of legal proposed moves which are accepted because of cost reasons.

Port moves are subject to the same cost arbitration. Nevertheless, port swaps are generally simpler because they only have to consider port placement legality (paired ports and side constraints); other than that, they can be swapped freely between port locations

because there are no partial overlap considerations.

A range limit is used to limit the distance a single move can transport a cell or port; it is kept constant during a placement temperature and is gradually reduced over the course of an anneal based on measures of proposed move acceptance and the temperature update schedule [1]. A range limit is used to restrict the search space of move possibilities that are considered when proposing a move for a cell or port. As an anneal proceeds, the placement will first globally improve to the point where cells will lie in relatively good positions with respect to one another; subtler improvements are made at lower and lower temperatures. A proposed move that moves a cell a large distance is more likely to be rejected as the anneal proceeds because the block already lies in a relatively good position. The practical distances cells can be successfully moved continually shrinks. Proposing an increasing number of moves which are likely to be rejected is counter-productive. Therefore, the range limit is designed to collapse the size of the proposed move search space around the moving cell as the anneal proceeds – so more productive local moves are considered; initially, the range limit is set to the size of the tile. The fruitfulness of the move search space is measured by the fraction of legal proposed moves accepted for cost reasons. Therefore, the feedback mechanism used to control the shrinkage of the range limit is based on acceptance ratio. [2] For example, as move acceptance decreases, the range limit is shrunk to focus the move generator's efforts on more productive local (likely acceptable) perturbations.

ATL establishes a minimum range limit of 20x20 to ensure that blocks can still effectively jump around their local area (effectively exploring the cost space) without resorting to multiple proposed moves to take them places.

2.5.2.5 ADVANTAGES OF SIMULATED ANNEALING

Simulated annealing offers several advantages that make it a well suited optimization engine for layout improvement.

Simulated annealing offers easy accommodation of new, differing optimization goals. Accommodation of new optimization goals is achieved by adding new component costs (with relative weightings) to the cost function and by making changes to the move generator (to expand or refine the move space).

Simulated-annealing offers cost "hill climbing". Individual legal placement moves are accepted and rejected depending on the cost change the move causes and the temperature. Moves that greatly increase the placement cost will be accepted at higher temperatures, but cost increase will be more and more discouraged as the temperature drops. In that way, simulated-annealing optimization allows "hill climbing" in the sense of exploring different parts of the cost space even if it means moving out of a local minimum (a cost increase) when trying to locate a more global (absolutely deeper) minimum. As the temperature is decreased, the area of search is reduced, and the algorithm tries to find areas of lower cost in the "general" region settled upon. To explore the cost space, the annealer move generator selects moves from a move space. The move space is the range of possible moves that the placer can perform to mutate the placement during cost space exploration. For a more detailed discussion of simulated-annealing optimization, refer to [5].

2.5.3 SRAM REGULARIZATION

Initially, ATL leveraged the arbitrariness of word and bit line assignments by creating a regular array of SRAM cells. SRAMs connected to the same word line were lined up and SRAMs connected to the same bit line were also lined up. This regular array was preserved throughout placement (Figure 2-12). A regular array of SRAM cells produces straight programming lines and consequently a low programming line cost. By ignoring the programming line net costs throughout placement, SRAMs were swapped with each other within the regular array structure to achieve good positions relative to the cells with which they were connected. Since word and bit line assignments are arbitrary, once placement was done, the length of the programming lines did not change from the initial low value, because the SRAMs stayed within the regular array, and the respective programming lines could be connected to the SRAMs they intersected. A 4% improvement in wirelength (measured through bounding-box wirelength cost) was achieved, by [1], using SRAM regularization.

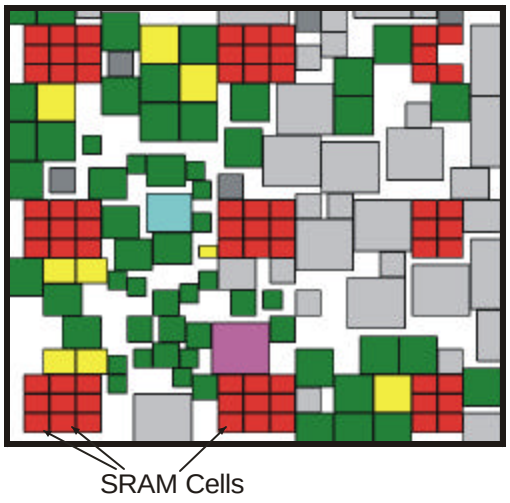


Figure 2-12 Illustrating Regular Arrangement of SRAM Cells

2.6 AUTOMATIC INTERCONNECTED BLOCK PLACEMENT

The placement problem explored in this work requires the placement of blocks of various size within a grid, without overlap; while fulfilling this requirement, wiring optimization and block compaction represent optimization objectives.

As mentioned in [1], utilizing a simulated-annealing-based optimization engine to solve this placement problem is a popular approach.

The Timberwolf tool [6] uses such an approach when performing "macrocell placement". This tool initially allows overlap in the placement of differently sized blocks. A penalty is used to gradually enforce separation of these blocks to reduce overlap. A clean-up phase at the end of placement is required to remove all overlap to make the placement legal. This is different than the approach used in ATL which forbids overlap at all stages of placement.

The placement algorithm used in VPR [3] also uses simulated annealing, and avoids block overlap, but it operates on blocks of the same size placed in a grid that accommodates one block per grid square. Therefore, the placement problem tackled by VPR is very regular in nature – involving block swaps and movement between grid squares.

There are also many algorithms available to perform one-dimensional and two-dimensional compaction. Layout compaction, in particular, is a popular topic. A layout compaction algorithm discussed in [7] is based on constraint graphs. A constraint graph is used to represent blocks that one-dimensional compaction is to be performed on, along with minimum and maximum distance constraints. Constraint-graph compaction involves finding the longest path in a directed graph. This algorithm does not operate directly on a

layout, it requires the construction and analysis of a constraint graph. Consequently, this compaction-focused optimization does not simultaneously consider other optimization goals while performing compaction and it is difficult to model other optimization goals within the framework of this algorithm

Another algorithm discussed in [7] tries to generate a layout with small area by examining equations expressing block area as a function of block aspect ratio (shape functions). By considering the shape functions of lower-level blocks, composite blocks made from those lower-level blocks can be characterized by their own shape functions. In a bottom-up fashion, higher and higher levels of block composition can be considered and, ultimately, a low-area layout aspect ratio can be determined for the overall layout. The consequences of this aspect ratio choice for the layout can be propagated down the composite block hierarchy to define the aspect ratios of all the lower-level blocks. This floorplanning does not deal with actual block positioning (an actual layout/placement) and, consequently, it does not simultaneously deal with other optimization objectives while determining the constraints necessary to produce a small layout. Another limitation of this approach (in the context of ATL and this research) is that cells first have to be characterized (shape functions determined) and then laid out to satisfy their imposed aspect ratios. As mentioned earlier, cell layouts are a subject of concurrent and future work, and, consequently, such layout characterization was not available. The idea of using ATL to determine cell aspect ratios and pin positions was briefly investigated and it is discussed in 5.2 as a possible avenue of future investigation.

Chapter 3
NEW OPTIMIZATION INFRASTRUCTURE

3.1 GOAL

The goal of this research was to develop placement heuristics that could be incorporated into the Automatic-FPGA Tile Layout (ATL) tool that attempt to: (1) minimize the area of laid out tiles, (2) minimize the wire length needed to interconnect the cells and ports within the tile, and (3) balance the wiring requirements over the tile area to prevent localized over-demand of wiring resources and, hence, avoid routing congestion.

The focus of this research was to achieve objectives (1) and (2). A congestion model, along with heuristics and costs designed to reduce congestion measured by the model, were developed to address objective (3). Nevertheless, techniques used to achieve objective (3) are considered tentative because an FPGA tile router was not available at the time of this research; therefore, verification of the assumptions used to address objective (3) is left to future work that can examine the behaviour and performance of a tile router. Details regarding these assumptions are provided in 4.3.3.

To meet the objectives stated above, several changes were made to ATL. These changes can be categorized into three major classes: (1) restructuring of the general optimization flow; (2) modification of existing algorithms in the simulated-annealing-based optimization engine; and (3) addition of new heuristics and costs. The first two classes do not notably improve placement results (layout quality); they provide the

foundations and infrastructure needed for the heuristics and costs to operate effectively and efficiently. It is changes belonging to the third class that directly attempt to satisfy the research objectives.

The first two classes are the primary subjects of this chapter. The next section describes the first class of changes in detail and the following section describes the second class of changes in detail. The last section briefly introduces the new heuristics and costs so the reader can be familiar with all of them before they are described in detail in the next chapter.

3.2 RESTRUCTURING OF GENERAL OPTIMIZATION FLOW

3.2.1 MULTI-PHASE OPTIMIZATION

The single-anneal optimization of ATL was replaced by a single initial anneal optimization phase followed by successive iterations of tile-shrinkage and annealing (these anneals are called reheat anneals). This multi-phase mechanism was introduced to permit the optimizer to decrease tile area while performing placement optimization. The initial incarnation of ATL did not shrink tile area during optimization. An initial tile area was calculated and the optimizer tried to work within that constraint. If the optimizer could not create an initial placement within the tile area specified, it would give up. This research showed that smaller tile areas could be achieved (balanced with adequate fulfillment of other placement objectives), if tile shrinkage is performed during optimization.

Here is pseudo-code describing this multi-phase optimization:

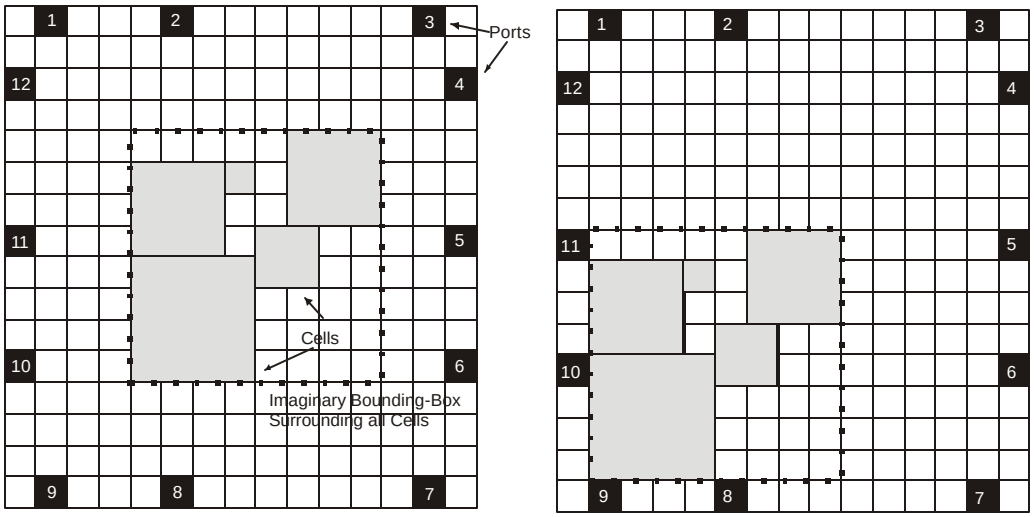
```
PERFORM INITIAL ANNEAL, SIMILAR TO THAT PERFORMED BY THE ORIGINAL ATL, RECORDING HIGHEST TEMPERATURE (?) THAT
        PRODUCES AN PROPOSED MOVE ACCEPTANCE RATIO BELOW ?
BEGIN REHEAT AND TILE-SHRINK ITERATIONS
        SHRINK TILE
        RECORD TILE AREA IMPROVEMENT ?
        REHEAT ANNEAL TEMPERATURE TO ?
        PERFORM REHEAT ANNEAL
        EXIT WHEN TILE-AREA IMPROVEMENT (?) IS BELOW THRESHOLD d, FOR e SUCCESSIVE ANNEALS
END REHEAT AND TILE-SHRINK ITERATIONS
```

It should be noted that there is an optimization tradeoff present in the selection of the reheat anneal temperature. If a higher temperature is selected, the reheat anneals can perform better cost "hill climbing" – perhaps, better exploring the cost space to find better layouts. If a lower temperature is selected, the reheat anneals will better preserve the decisions made during the initial anneal and previous reheat anneals; therefore, past optimization effort can be better leveraged. This issue is discussed more thoroughly when the detailed optimization heuristics are described.

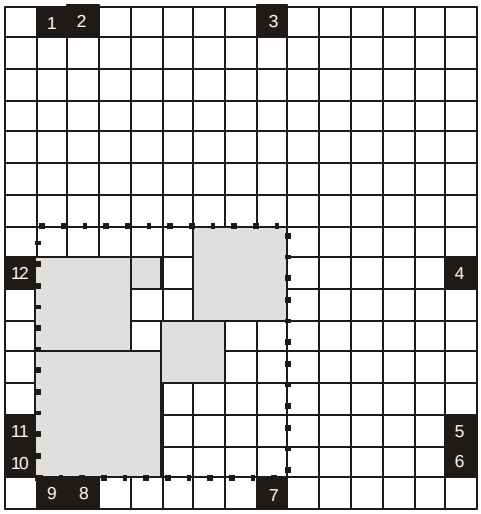
3.2.2 TILE COMPACTION BETWEEN REHEAT ANNEALS

Tile compaction between anneals is performed to permit tile shrinkage during optimization. The tile is compacted by moving all cells as a unit, until the lower-left corner of the imaginary bounding box tightly enclosing them touches the lower-left corner of the port perimeter. The ports are then collapsed around these cells completing the shrinkage. Figure 3-1 illustrates this. Notice how the relative cell positions and the port ordering are maintained so that the optimization effort of previous anneals is not lost during compaction and subsequent reheat anneals have a good global starting point to work from. That is why ports are also not moved more than necessary to achieve tile collapse. This is not as important, however, because ports tend to sort themselves out quickly during optimization because they can swap freely with one another – no partial

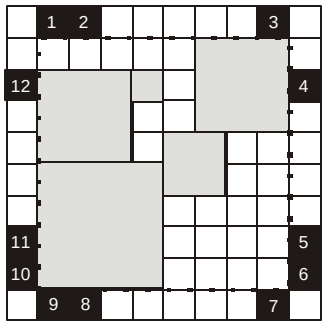
overlap problems like cells have to contend with.



Step One:
Translate all cells, maintaining relative positions, to lower-left corner.



Step Two:
Move ports on left and right downwards.
Move ports on top and bottom leftwards.
Preserve relative ordering of ports and do not move ports a greater distance than the cell will shrink, in this case, 5 horizontally and 6 vertically.



Step Three:
Collapse ports and shrink tile.

Figure 3-1 Illustrating Tile Compaction Between Reheat Anneals

3.3 MAJOR MODIFICATION OF EXISTING FEATURES IN SIMULATED-ANNEALING-BASED OPTIMIZATION ENGINE

As mentioned in 2.5.2, there are four major components in a simulated annealer adapted to create FPGA tile layouts: (1) initial placement; (2) annealing schedule; (3) layout cost; and (4) move generation and move cost arbitration.

The major modifications made to each of these components are described in turn.

3.3.1 INITIAL-PLACEMENT MODIFICATIONS

This research extended the tool so that more than one port can be placed at a port location. This extension required not only changes to initial placement, but also to move generation. The extension is necessary because the compaction of some tiles is not limited by cell area, it is limited by the size of the port perimeter. It turns out (according to this research) that an overlap of two ports per port location is sufficient to prevent compaction of all tiles experimented with from being constrained by ports. It is physically possible to have more than one port at a location because of the existence of more than one metal layer. Future work on the tile router has to be careful, however, that matching or paired ports, on opposite ends of the tile, end up on the same metal layer.

3.3.2 ANNEALING-SCHEDULE MODIFICATIONS

Instead of using a single exit criterion to determine when an anneal is complete, different exit criteria are used for the initial anneal and the subsequent reheat anneals.

The exit criteria are based on a temperature threshold and a cost-improvement percentage threshold. If the current temperature is below the temperature threshold and the cost improvement over the last temperature is less than the cost-improvement percentage threshold, the exit criterion is satisfied. The use of these two parameters permits flexible tuning of how hard the annealer tries to thoroughly explore the final local cost minima before exiting an anneal. That way, effort can be diverted from over-optimizing local minima of earlier anneals in favour of deeply exploring the local minima of the final anneal. This change was motivated for the sake of run-time; experiments showed, if tuned appropriately, the run-time can be cut, at least, in half, without sacrificing quality.

3.3.3 LAYOUT COST MODIFICATIONS

The wirelength cost, originally used in ATL, was maintained because of its fruitfulness in leading towards many optimization goals. This cost is designed to penalize moves that increase the total estimated bounding-box wirelength used to make all the cell connections. This encourages the general movement of cells closer to cells with which they are connected. This minimizes the total amount of routing (metal) needed by the router to make all the connections and it should decrease the length of any given connection. Hence, this cost directly tries to satisfy objective (2), which is the reduction in overall wirelength. In fact, the experimental measure used, to determine the extent that objective (2) is satisfied, is the value of the wirelength cost. As a side benefit, minimization of wirelength should also decrease general routing delay and improve the speed of the circuit; of course, more targeted optimization for timing, based on timing analyses, would produce better (more focused) timing results; timing-oriented

optimization is left to future work. Another side benefit of wirelength reduction is the overall reduction in routing also reduces overall routing capacitance and total resistance; this should also reduce power consumption to a certain degree. Detailed study and focused power optimization is left to future work, however.

The precise formulation of the wirelength cost was revised. Wirelength is still measured based on bounding boxes, however, the bounding boxes are now precise instead of pessimistic (as originally implemented in ATL). That is, the bounding box corresponding to a net is stretched around all the attached cell pins and ports exactly. The formulation of the cost initially in ATL meant that the corresponding bounding box stretched around all the respective cells and ports (not considering actual cell pin positions). Figure 3-2 illustrates the new notion of a bounding box with respect to a net (compare this to Figure 2-9).

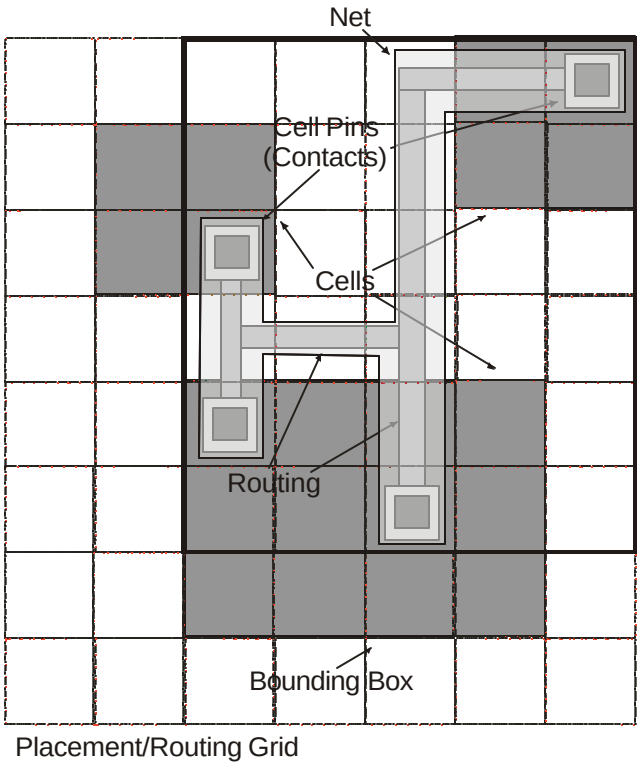


Figure 3-2 Illustration of a Net and Its Bounding Box

3.3.4 MOVE GENERATION AND MOVE COST ARBITRATION MODIFICATIONS

The move generator was changed to permit better movement of larger cells. It turns out that larger cells that lie near the perimeter of the tile are the bottleneck to tile compaction (achievement of objective (2) – small tile areas). The generally large number of connections attached to larger cells and their bulk size make them difficult to move; if they lie near the edge of the tile, they will inhibit compaction. Consider Figure 3-3, the move generator proposes to move the larger block (primary block) upwards one unit away from the tile perimeter. This would cause overlap with the smaller block above it. The original move generator would propose an inverse move of one unit downwards for the smaller block. This would result in an illegal placement and so the move is aborted. As mentioned earlier, if the proposed move of the primary cell is (?x, ?y), the original move generator proposes the inverse move (-? x, -? y) for group B cells (refer to 2.5.2.4 for more details). The modified move generator ensures an inverse move covers distances at least equal to the respective dimensions of the primary block. The modified move generator proposes the inverse move:

$$(SGN(-?x) \cdot MAX\ (ABS(?x), WIDTH_{PRIMARY\ BLOCK}), SGN(-?y) \cdot MAX\ (ABS(?y), HEIGHT_{PRIMARY\ BLOCK}))$$

where $sgn(?)$ is 1 if $? > 0$, -1 if $? < 0$, and 0 if $? = 0$.

The modified move generator would produce an inverse move, in Figure 3-3, of three units downwards. This results in a successful move.

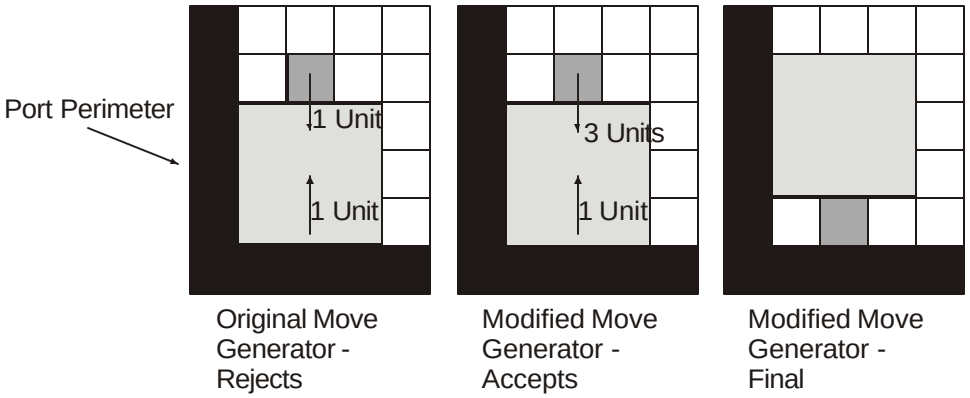


Figure 3-3 Move Generator Modification: Example 1

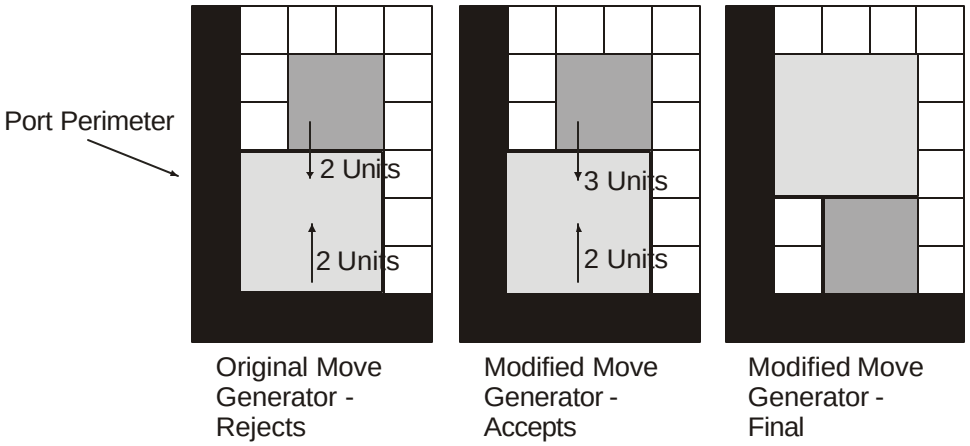


Figure 3-4 Move Generator Modification: Example 2

Consider another example in Figure 3-4. The proposed move is to move the larger cell two units upwards. The original move generator would create an inverse move for the smaller cell of two units downwards. This would result in an illegal placement. The modified move generator proposes an inverse move of three units downwards resulting in a legal placement.

Even though this mechanism is a heuristic that does not solve all proposed move feasibility problems, it represents a slight tweak targeted to address the fundamental problem of nudging cells off edges. It turns out that despite the fact that it changes the notion of what constitutes an inverse move, it otherwise does not affect placement results

(other than indirectly helping facilitate tile compaction); it solely helps swap larger cells away from the tile perimeter. Finally, notice that the inverse move is a function of the primary cell dimensions and displacement alone. This is necessary to ensure that the same inverse move is applied to all the respective cells (group B cells, referred to in 2.5.2.4); otherwise, if differing inverse moves are applied, additional overlaps may be created.

This research found that a minimum range-limit a small multiple (1 or 2) of the largest cell's longest dimension further facilitates this nudging of cells off edges. Notice that in the examples presented (Figure 3-3 and Figure 3-4), the proposed move must displace the larger cell by a distance at least equal to the smaller cell's relevant dimension for the move to be successful. Otherwise, not enough room is evacuated to accommodate the smaller cell. By basing the range limit on the largest cell's dimensions, the move space is adapted somewhat to the size of cells in the netlist. Consider this adaptability opposed to the static minimum range limit originally defined in ATL that was a fixed size of 20x20. It turns out that because the largest cell dimensions in the netlists experimented with are approximately 20 units, this modification has little affect on any placement optimization results other than tile compaction and ensuring adequate nudging can always be performed to permit tile shrinkage.

3.4 Overview of New Heuristics and Costs

This section provides short descriptions of the mechanisms added to ATL to address the various optimization objectives. The various mechanisms are briefly described to provide general context before each, in turn, is described and examined in

detailed, in the next chapter.

3.4.1 TILE-SIZE COST

This cost has two components. One component is designed to penalize moves that increase the area of an imaginary bounding box enclosing all the cells (not ports) in the tile. This encourages "crunching" of the cells together away from the tile edge to promote and facilitate tile shrinkage during the reheat and tile-shrink iterations. The other cost component is designed to penalize moves that increase the number of cells on the edge of the imaginary bounding box enclosing all of the cells. This encourages evacuation of the imaginary bounding box perimeter, hopefully, leading to a collapse of the box.

3.4.2 TILE-SLOPE COST

This cost is designed to penalize moves that increase the distance of a block from the geometric center of the tile. This encourages "bunching" of the cells together towards the center of the tile to promote and facilitate tile shrinkage. By constantly tugging cells inwards, this cost works with the tile-size cost by moving "central cells" further from the edge, so cells on the edge have space to move off the edge of the imaginary bounding box. Its name suggests the effect the cost has on the placement; the cost creates an effective slope inclined towards the edge of the tile that cells tend to tumble down.

3.4.3 WIRE-OVERUSE COST

This cost is designed to penalize moves that create routing hotspots in the

placement or increase the "hotness" of a hotspot. Routing hotspots are areas of the placement identified as being problematic routing areas; these areas are identified as locations the placer thinks the router will route many distinct nets (distinct electrical nodes) over. These nets will have to occupy different routing layers and consequently hotspots indicate areas of the chip containing nets the router may fail to route because it does not have enough routing resources in the local area. The more nets the placer deems will have to be routed over a given spot, the "hotter" the hotspot and the greater the anticipated routing difficulty. By encouraging the reduction in the "hotness" and number of hotspots, the routing difficulty should be decreased. This cost, in particular, may fight tile compaction and the minimization of wirelength; for example, increasing the area required to route nets (over hotspots) decreases the magnitude of the hotspots by giving the router more options (metal) to complete the respective routes. This cost represents work in progress because its formulation relies on a placement model of routing congestion that must be confirmed and tuned (in future work) to that encountered by an actual FPGA tile router.

3.4.4 BLOCK-OFF-EDGE MOVE

This move is designed to encourage cells to move off the edge of the imaginary bounding box enclosing all cells. This encourages the collapse of the imaginary bounding box, hopefully, leading to tile compaction. The need for this type of move originates from the following. Once the general locations of blocks settle during placement, the move acceptance function tends to reject moves that span a great distance and the range-limit move generator tends to propose moves in a local region. To shrink the tile, however,

sometimes moves have to be accepted that transport cells across a large distance because cells can only move to locations that can accept them (possibly with a bit of cell juggling) without producing an overlap. This move type addresses both cost arbitration and move generation issues that would otherwise prevent the type of long-distance move often needed for the sake of tile compaction. The block-off-edge move reduces the magnitude of cost increase associated with moves that would successfully move cells off the edge; it also removes the standard range-limit, imposing its own range-limit geared to explore moving a cell off an edge of the imaginary bounding-box to a location somewhere just inside that edge. This move type works in conjunction with the tile-size compaction cost.

3.4.5 COMPACTION MOVE

This "compound" move is designed to encourage cells to converge towards the center of the tile, making room for the outer cells to move off the edge of the imaginary bounding box surrounding all cells. This move type works in conjunction with the tile-slope compaction cost. This move actually involves all cells (hence, it is a "compound" move) in a focused effort to take immediate advantage of gaps and spacings for cells to move closer to the center of the tile. Each compaction move tries to move all cells. Cells closer to the center are moved inwards first, hopefully, opening gaps that outer cells can, in turn, fill. Each individual cell move is arbitrated with the cost function like a normal move. It is the move sequencing which makes compaction moves useful.

3.4.6 BLOCK ROTATION AND FLIP

This move expands the move space by exploring cell re-orientation possibilities. Cells can be flipped (horizontally and vertically) and rotated (90 degree angles) because the routing grid and the cell layouts are orientation independent to a certain extent; that is, in terms of design rules, layouts can be flipped and rotated as indicated.

3.4.7 BLOCK EQUIVALENT-PIN SWAP

Certain cells have groups of input pins and output pins whose respective connections can be swapped because the functionality of the circuit is independent of exactly which signals are routed to the various pins on each cell. This move expands the move space by exploring these pin swap possibilities. During cell topology generation (during netlist generation), care is taken to distribute cell input and output pins in positions across the cell that reflect their actual positions in the layout. These pin positions are not motivated by netlist and placement considerations and, hence, from a placement and netlist perspective, the actual positions are random. Furthermore, netlist generation does not take into account placement and cell topology, and, hence, from a placement perspective, connections are made to essentially random cell locations. The netlist generator actually attempts to ensure that there is no systematic bias tying related connections to adjacent cell pin locations, etc.. This move type, and the previous move type (3.4.7), attempt to explore better positions for those connections than those randomly assigned by the netlist generator.

3.4.8 INITIAL LARGE-GRID PLACEMENT

Cells are placed on a large tile initially to allow globally good placements for cells to be determined without cell overlap inhibiting move generation freedom. Placements with little free space tend to have a lot of proposed moves rejected for legality reasons solely (not cost reasons). Therefore, if placement optimization begins with a dense placement, cells will have a hard time achieving globally good positions relative to one another because many proposed moves will be rejected. This technique inflates the initial tile size and modifies the move generator so cells can be swapped freely with one another without overlap considerations. Cells are spaced out on the large initial tile, so a given cell move involves at most two cells in a swap. Once initially good global placements for cells are found, normal cell movement ensues and compaction techniques are used to reduce the tile size from this initial large tile configuration.

3.4.9 SRAM REWEAVE

SRAM word and bit lines are assigned randomly to SRAMs by the netlist generator to avoid systematically associating certain SRAMs together by connecting them to the same programming lines. As discussed in 2.4.2, word and bit line assignments are arbitrary because FPGA SRAM programming can be adjusted to program the target SRAMs to any values for any fixed word and bit line assignment. SRAM Reweaving gives the layout optimizer the freedom to initially place SRAMs without considering word and bit line costs. After all the SRAMs are assigned good global positions (close to logic they are attached to), word and bit lines can be assigned to SRAMs (rewoven) based on SRAM placement. That way, the arbitrariness of word and bit line assignments can be leveraged to minimize word and bit line length and to

minimize consideration of programming line length when determining globally good positions for SRAMs.

Chapter 4
OPTIMIZATION TECHNIQUES AND COSTS

Each of the major optimization techniques (heuristics) and costs (summarized in Table 4-1) that were developed for ATL to help meet the objectives stated in 3.1 are described, in detail, throughout this chapter. The techniques and costs are organized by type and the goals they address.

The first section of this chapter discusses the FPGA tiles used during experimentation (the benchmark set). The second section points the reader to pseudo-code descriptions of the overall optimization flow and the move generation process. The third section examines the cost-based optimizations. The fourth section examines the move-based optimizations. The fifth section examines the optimizations which do not completely fall under either of these classifications. The last section presents an overall performance comparison between the version of ATL produced by this research and the initial version of ATL, produced by [1].

Table 4-1 Technique and Cost Summary

		Techniques and Costs		
		Cost-Based	**Move-Based**	**Other**
Goal	**Minimum Area**	Tile-Size Cost Tile-Slope Cost	Block-Off-Edge Move Compaction Move	
	Minimum Wire Length		Block Rotation and Flip Block Equivalent-Pin Swap	Initial Large-Grid Placement SRAM Reweave
	Balanced Wiring Requirements	Wire-Overuse Cost		

4.1 EXPERIMENTAL BENCHMARK SET

As each of the techniques and costs are examined, results are presented to

quantify their significant effects and examine observable trends. These results are based on experiments conducted during this research. Most of the experiments conducted (and all of the results presented in 4.3, 4.4, 4.5) were based on the set of ten "benchmark tiles" used in [1]. These benchmark tiles were created by [1], based on architecture files that were found to result in high-quality FPGAs [3]. The electrical parameters used to size buffers and switches, which were specified in the architecture files, were based on TSMC's 0.18 µm technology and were taken from work done in [8]. The ten benchmark tiles created by [1], and used in this research, all had 40 wires per FPGA routing channel and a differing number of LUTs per tile (1 to 10).

As mentioned earlier, the results of this research, to a certain extent are independent of the exact cell sizes and net connectivity of FPGA tiles. In fact, the generality of the results make the heuristics useful for the exploration of diverse FPGA architectures, from which cells and ports can be abstracted.

4.2 PSEUDO CODE

If a better understanding of the interrelationship between and the integration of the various techniques is desired, the reader is encouraged to examine the pseudo code presented in Appendix A and B, as each of the techniques and costs are described. Appendix A contains the pseudo code that summarizes the overall placement flow. Appendix B contains pseudo code that illustrates the details of move generation.

4.3 COST-BASED OPTIMIZATIONS

First cost-based optimizations are presented which leverage the cost-based move arbitration of simulated annealing to achieve layout goals.

4.3.1 TILE-SIZE COST

Actual tile compaction only occurs between reheat anneals; therefore, the reheat anneals need to monitor and encourage congregation of cells towards the center of the tile, so compaction can actually occur between the anneals. If the annealer tries to optimize wirelength and other measures of placement merit without directly considering eventual tile compaction, active re-arrangement of cells will have to be performed during the tile compaction phase. This is because the annealer will make no direct attempt to group cells (compact cells together); consequently, they may spread out, attempting to satisfy other constraints, over the entire tile area. If cells are actively re-arranged, during the tile compaction phase, either only tile compaction concerns are considered (reducing overall placement quality) or extra complexity has to be added to that phase to monitor all the quality aspects the annealer is designed to monitor.

To allow tile compaction efforts to consider general placement quality, a tile-size cost is added to monitor progress towards tile shrinkage during the anneal (that can be balanced with other placement costs). That way, tile compaction can be balanced with wirelength improvement, for example, and moves that greatly benefit eventual tile compaction can be immediately recognized and accepted.

This cost is based on an "imaginary" bounding box that encloses all the tile cells (not ports) (described in 3.2.2 and Figure 3-1). This bounding box is incrementally maintained as the cells are moved. The cost has two components, as follows:

$$COST = (WIDTH_{IMAGINARY\ BOUNDING\ BOX} ? HEIGHT_{IMAGINARY\ BOUNDING\ BOX}) + NUM\_BLOCKS_{LEFT\ SIDE} + NUM\_BLOCKS_{RIGHT\ SIDE} + NUM\_BLOCKS_{TOP\ SIDE} + NUM\_BLOCKS_{BOTTOM\ SIDE}$$

One component of the cost is the "imaginary" bounding box area. Moves that decrease the area of the bounding box improve this component of the tile-size cost. The second component of the cost is the number of cells lying on the edge of the "imaginary" bounding box. This component of the cost is designed to encourage movement of cells off the edge of the "imaginary" bounding box so that collapse of box area eventually occurs. A tile-size cost multiplier is applied to the calculated tile-size cost so it can be weighted with respect to the bounding-box wirelength cost.

The area of the imaginary bounding box is used instead of the perimeter, for example, because of the interaction with the second component of the cost. The area component of the cost is the only meaningful aspect of the cost. The cell count on the edge affects the second component of the cost but it does not affect the tile-compaction potential which is strictly a function of the imaginary bounding-box area. Therefore, the second component is designed to be a tie-breaking cost in the sense that given the same area, the cell count on the edge is the deciding factor. This means that if a side of the imaginary bounding-box collapses by one unit, even if the cell count on the edge increases, the overall cost should show the merit of the respective move. For example, by monitoring the imaginary bounding-box area, shrinking a WxH bounding box to Wx(H-1) results in a cost improvement of the first component by W units. As few as one cell can be on the edge that collapsed, and, following the collapse, as many as W cells can be on that edge (maximum one cell per grid square). Therefore, in general, despite the maximum cost increase of the second component (W – 1), the cost decrease of the first

component is sufficient for the tile-size cost to recognize the merit in shrinking the imaginary bounding-box area.

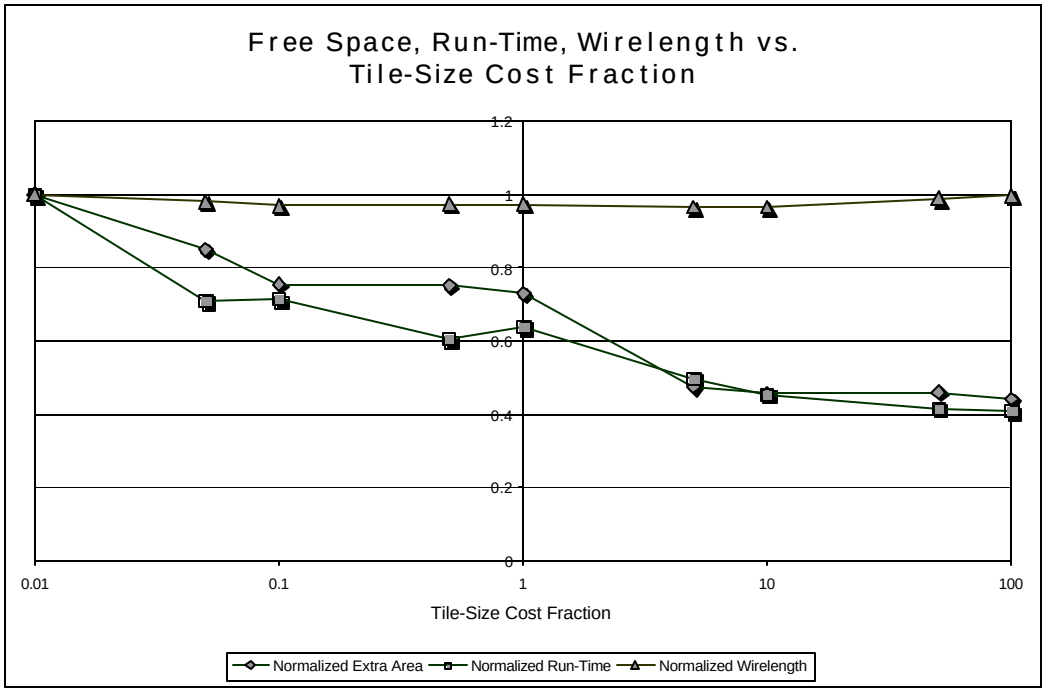
The cost factor that is used to weight this cost relative to the bounding-box wirelength is re-determined between reheat anneals. It is calculated so that the weighted tile-size cost is a certain multiple of the bounding-box wirelength cost. The precise multiple (fraction) is increased over the course of the anneal up to a certain value. This upper-bound value is adjusted during experimentation. It turns out that one can not completely "turn off" this cost because the placement run times grow unreasonably large as tile compaction occurs slowly and randomly over a long period of time. Therefore, experiments adjust the maximum tile-size cost fraction, but do not set it to 0.

Table 4-2 Tile-Size Cost Fraction Comparison

	Tile-Size Cost Fraction: 0.01 (?)			**Tile-Size Cost Fraction: 5 (?)**			**Improvement (? Result/? Result)**		
Number of LUTs	*Free Space / Cell Area*	*Run Time (s)*	*Wirelength*	*Free Space / Cell Area*	*Run Time (s)*	*Wirelength*	*Free Space / Cell Area*	*Run Time*	*Wirelength*
1	0.67	317	38363.2	0.13	193	34525.5	0.19	0.61	0.90
2	0.28	721	40991.8	0.16	267	41394.2	0.57	0.37	1.01
3	0.46	802	50532.2	0.12	437	48248.8	0.26	0.54	0.95
4	0.28	1120	48698.3	0.13	399	46896.6	0.47	0.36	0.96
5	0.34	1415	62175.8	0.12	598	60563	0.35	0.42	0.97
6	0.23	1834	70322.9	0.10	1019	68170.9	0.46	0.56	0.97
7	0.19	1975	74099.4	0.11	858	72614.3	0.62	0.43	0.98
8	0.20	2152	81387.9	0.10	1150	78154.4	0.53	0.53	0.96
9	0.22	2024	91470.5	0.10	1550	87457.8	0.45	0.77	0.96
10	0.12	4333	103764	0.10	1741	102187	0.85	0.40	0.98
Average							**0.47**	**0.50**	**0.97**

It turns out that the average percentage free space in the tile is cut in half (reduced by a factor of 2.11) by increasing the tile-size cost fraction from 0.01 to 5. The run-time is similarly cut in half and the wirelength is improved by 3.5%. Higher cost fractions start to increase the overall wirelength because tile-size cost reduction is prioritized over wirelength optimization. Therefore, a tile-size cost fraction of 5 was settled upon. It should be noted that smaller tile sizes reduce the total amount of wire used up to a certain

point, however, there is a wirelength penalty incurred by incessantly compacting cells within the rectangular tile – wrenching them away from their globally good positions. Table 4-2 illustrates the experimental results obtained from the benchmark set for two cost fractions. Graph 4-1 illustrates the change of free space, run time, and wirelength as the tile-size cost fraction is adjusted; note the logarithmic horizontal scale.



Graph 4-1 Free Space, Run Time, Wirelength vs. Tile-Size Cost Fraction

4.3.2 TILE-SLOPE COST

The tile-size cost is designed to decrease the size of the imaginary bounding box enclosing all cells. The cost encourages movement of the cells inwards just off the edge of the bounding box so the bounding box can collapse. Nevertheless, unless there is a general evacuation of cells away from the edge towards the tile center, there will be a buildup of cells just inside the edge. This is because other costs (optimization goals) may motivate cell movement towards the bounding-box edge (closer to the ports, for

example). After successive imaginary bounding-box shrinks, a larger collection of cells will build up just inside the box perimeter; this general trend will result in an overall placement of cells, which inhibit further "imaginary" bounding-box collapse.

To prevent the cell buildup that inhibits imaginary bounding-box collapse, the tile-slope cost was created. This cost refers to a tile slope because it creates a virtual slope inclined towards the edge of the tile that encourages cells to "tumble" towards the center. It is a "weak" cost in the sense that it is designed to tug slightly on all cells without disrupting general placement quality (measured through other costs). It is designed to break cost "ties" and generally encourage movement of cells away from the tile edge so that the annealer, with its tile-size cost, can effectively achieve imaginary bounding-box collapse.

This cost (for each cell) is formulated as follows:

```
COST = MAX(HORIZONTAL DISTANCE OF CELL CENTER FROM VERTICAL LINE THROUGH TILE CENTER / TILE WIDTH, VERTICAL
        DISTANCE OF CELL CENTER FROM HORIZONTAL LINE THROUGH TILE CENTER / TILE HEIGHT)
```

The effect of this cost formulation can be visualized by envisioning the concentric rings of equal cost that it creates (Figure 4-1). The center of a cell can move along a rectangular ring, without affecting the respective cost. A move which translates the center of a cell to a rectangular ring which is outside the current rectangular ring results in a tile-slope cost increase. By using the max of the two dimensions rather than the sum, for example, is motivated by the tile shape and orientation. The sides of the rectangular rings are parallel to the tile sides, so that cells do not conglomerate to form shapes distinctly different from that of the tile (like in Figure 4-2). This would be unbeneficial and would waste area; it is better that the cell conglomeration "match" the tile boundary so the latter can collapse snugly over the former without any area wastage.

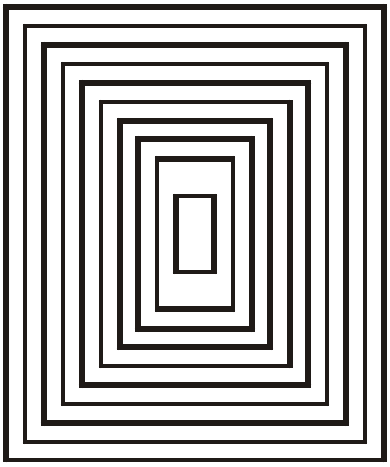


Figure 4-1 Rings of Equal Tile-Slope Cost

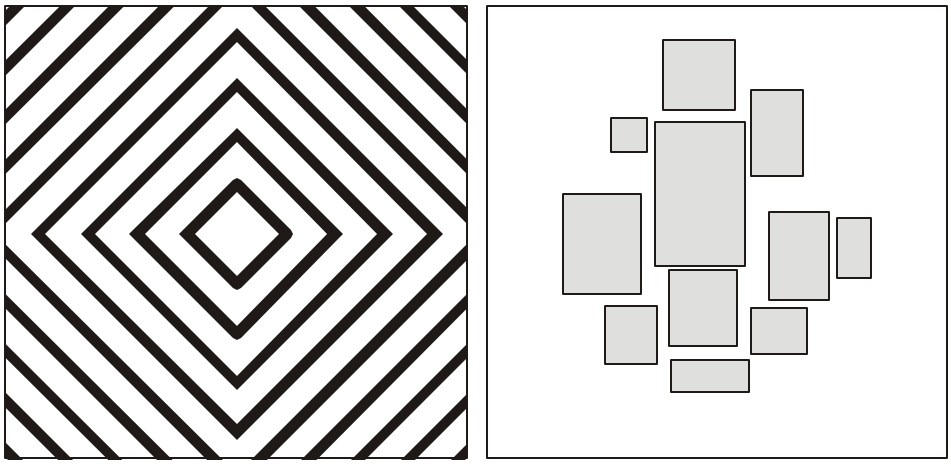


Figure 4-2 Rings of Equal Tile-Slope Cost Incongruent with Tile Boundary and Resulting Placement

The normalization of horizontal and vertical distance with respect to tile width and height has the effect of biasing which direction of cell migration the cost prefers. Imagine the tile width is twice the magnitude of the tile height. A move which decreases the horizontal distance by one unit has half the cost benefit of a move which decreases the vertical distance by one unit. Therefore, in general, vertical collapse is promoted. The cost structure has the effect that collapse is promoted in the direction that corresponds with the smaller dimension. Smaller height than width encourages vertical collapse and smaller width than height encourages horizontal collapse. By promoting skewing of tile

aspect ratio, better compaction (final tile area) results were observed (7% reduction in tile free space). One can understand this better by considering that a one unit collapse in one dimension of a 400x300 tile results in a 400 square-unit area reduction while a one unit collapse in the other results in a 300 square-unit area reduction. Therefore, the experimental result seems to indicate that it is equally easy to achieve collapse in both dimensions, so biasing collapse in the already smaller dimension is beneficial.

The tile-slope cost factor is set just like the tile-size cost factor to normalize it between reheat anneals to be a certain multiple of the bounding-box wirelength cost. It turns out that as long as the fraction (multiple) is kept to a reasonably low value, there is no change in wirelength results or final tile free space as the precise fraction is varied. It turns out that this cost is merely a time saver. By pulling all cells inwards before they inhibit further compaction, overall run-time is decreased. A total run-time savings of 16% was achieved without sacrificing placement quality.

4.3.3 WIRE-OVERUSE COST

Monitoring just overall wire utilization is not sufficient to ensure route-ability. If too many different connections have to be routed in a given area, the router will either be unable to route all the connections because there are not enough metal layers in the given region (available in the given technology), or the router will have to try extending the routes outside of the current region, if it is possible. If there are enough non-optimal routes, even if there is "enough wire" in the chip theoretically to route all the connections, the router will use up all available wiring leaving some connections not routed. Non-optimal routes also increase the actual length of connections beyond their

expected length (defined within the placer's net bounding boxes) leading to increased path delays. A timing-driven router can avoid extending connections that are "critical" for timing; nevertheless, forcing the router to take non-optimal routes is generally unbeneficial because the increased wire usage increases the overall pressure on routing resources – making the router's job more difficult. If the tile can not be routed, the router may also have to resort to growing the tile in certain areas to make room for the respective routing. This is disadvantageous because it increases the overall tile area.

If the severity of the routing "hotspots" can be reduced (less severe local wiring demand), and the overall wire utilization is sufficiently low, the router may also complete its job with fewer layers of metal. This ultimately could reduce the expense of producing the FPGA and would consequently be beneficial. Reducing the amount of non-optimal routing the router has to resort to also reduces the total routing capacitance and resistance, reducing the overall power consumed due to the routing.

4.3.3.1 CONGESTION MODEL

This research was conducted in the absence of a router. Therefore, to measure routing congestion, a router congestion metric was developed. This congestion metric was adapted to form a placement cost to monitor congestion. If future work shows a correlation between this congestion metric and router congestion, this congestion model and metric can be tuned to reduce final congestion.

The congestion model used considers overlapping bounding boxes. The congestion over each placement grid square is calculated. It is calculated by adding contributions from each of the wirelength bounding boxes overlapping the grid square.

The contribution of a bounding box is equal to the total wirelength of the corresponding net divided by the number of routing grid squares covered by the bounding box (net routing density). This estimate of congestion is based on several assumptions. It assumes the router can, typically, route all the connections of a net without violating the respective bounding box perimeter. It is also assumes the amount of extra routing square segments needed for metal hops (vias) is relatively small; note that when a via is used, two routing square segments at the respective placement grid square is used instead of one. It is also assumed the router can chose any number of possible paths to route all the net connections within the bounding-box; hence, probabilistically, the larger the area of the bounding box, generally, the smaller the probability the router will use a given grid square – actually, it is the area to perimeter ratio that matters because wire-use increases with perimeter. By basing the contribution of a net on the probability the router will use the respective placement/routing grid square, it is hoped that potential router choice can be incorporated in the congestion assessment.

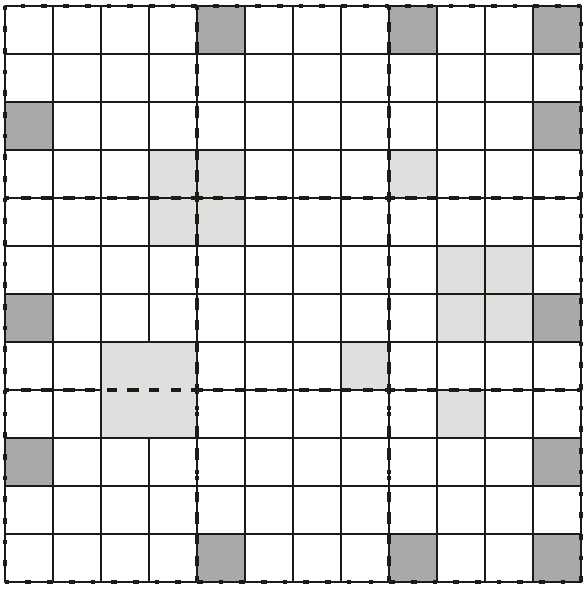
It should be noted that this model's view of routing flexibility is overly optimistic. A better modeling of router options would consider that near points of connection (to cell pins and ports), the net must establish the appropriate connections. Consequently, near net sinks and sources, the probability of the router using a routing square segment increases. This affect can be approximately modeled by carefully considering the area around sinks and sources (adding the relevant high probability bonuses to the respective placement grid squares); this modification to the routing model was experimented with briefly and should be reconsidered when future work assesses overuse modeling and costing with reference to a tile router. Infrastructure was put in place to bonus routing

grid squares around relevant cell pins and ports, however, the cost described next and the experimental results do not incorporate this cost or model feature since the respective bonus magnitudes are highly speculative without considering a tile router carefully.

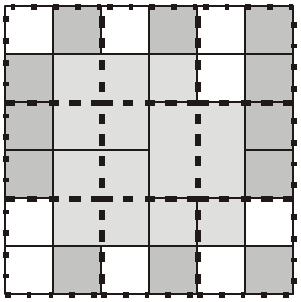
4.3.3.2 CONGESTION COST

To avoid introducing routing hotspots in the placement and to reduce overall routing congestion, a placement wire-overuse cost was created. This wire-overuse cost is based on the congestion model proposed. Monitoring the overuse with the preciseness of the proposed model is too expensive (in terms of run time). To make the run-time practical, a few approximations are made that do not impede the placer's ability to reduce placement congestion as measured by the proposed model.

Instead of considering each placement grid square individually, a coarser congestion grid is monitored. The coarseness of the grid is specified in terms of the number of routing grid squares each congestion grid square should enclose when the tile is highly compacted – most of the tile area filled with cells. If the tile is loosely compacted, the coarseness of the grid is artificially increased so the height and width of the congestion grid matches the height and width of the grid if the tile was highly compacted. Refer to Figure 4-3 for an illustration of this. As will soon become evident, the run-time of the algorithm and the "realism" of the cost are both a function of the coarseness of the grid. Therefore, this decrease in coarseness as the tile is shrunk was motivated by several reasons, which will be discussed later.



4x4 Placement Grid Squares per
Congestion Grid Square when
tile is loosely compacted.
Congestion Grid: 3x3



2x2 Placement Grid Squares per
Congestion Grid Square when
tile is highly compacted.
Congestion Grid: 3x3

Figure 4-3 Coarseness of Congestion Grid as a Function of Tile Compactness

The congestion cost of a congestion grid square is:

```
AREA OF CONGESTION GRID SQUARE (NUMBER OF PLACEMENT/ROUTING GRID SQUARES ENCLOSED) ? MAX(0, CONGESTION
        MEASURED – (CONGESTION THRESHOLD – 1.0))^2
```

The measured congestion is a function of the net routing density of all nets that intersect the congestion grid square. The net routing density of all nets which intersect the square are added together to get the worst-case congestion value (as if all the nets actually intersected over one placement grid square within the congestion grid square). By

considering the worst-case congestion value, a coarse congestion grid will report congestion even if there is not any, and, it will never fail to recognize congestion. Currently, a congestion threshold of 5.0 is used, because it seems to work well to reduce the congestion reported by the congestion model. A value of 5.0 also seems reasonable because the congestion reported at a placement grid square is equal to the expected amount (expected value) of metal lines needed over that square. Consequently, there should be a correlation between the maximum congestion measure and the number of layers of metal needed for routing. A threshold of 5.0 would imply that congestion should be limited to five layers of metal and overuse should be reported above that amount. Assuming an eight-layer metal process (achievable by state-of-the-art processes, now, or in the near future [9]), five layers of metal for inter-cell routing seems reasonable; this would leave 3 layers of metal for intra-cell routing and routing of power and ground (which are not in the tile netlists). Nevertheless, this threshold value should be experimented with in future work.

The amount the measured congestion exceeds one less than the threshold value is squared so that congestion can be balanced throughout the tile. That is, an excess congestion value of 6.0 at one location is penalized more than two excess congestion values of 3.0 at two locations. 1 is subtracted from the threshold value so that at the threshold value, 1 unit of congestion cost is measured. If this is not done, the optimizer will almost never reduce the congestion below the threshold value because values slightly above the threshold would have negligible cost.

Finally, by multiplying $\text{MAX}(0, \text{CONGESTION MEASURED} - (\text{CONGESTION THRESHOLD} - 1.0))^2$ by the congestion-grid "placement square area", a cost is determined that assumes the worst-

possible congestion in the congestion grid square occurs at all positions within the region. Of course, because of this the greater the coarseness of the congestion grid, the greater the worst-case analysis is overly pessimistic. Therefore, finer grids tend to estimate congestion more realistically.

Generally, the coarser the congestion grid, in general, the fewer the number of congestion grid squares ("bins") crossed by a given net. That is why, increasing the grid coarseness reduces the run-time; the number of bins that have to be updated for a given net is reduced.

It turns out a better balance between precision and run-time can be achieved, without sacrificing placement quality, as measured by the overuse model. By only monitoring certain nets with respect to wire-overuse cost, the optimizer can avoid having to make as many updates during an anneal. Nets that have a low net routing density do not greatly affect congestion measures. Furthermore, they tend to be large-area nets that span a large number of "congestion bins". Therefore, not considering these nets is beneficial because of their low affect on congestion measures and the fact that the expense in updating bins is greatly reduced if they are ignored. The threshold density below which nets are ignored is based on the congestion threshold and the maximum number of nets which cross a placement/routing grid square in the tile. By dividing the congestion threshold by the maximum number of nets which cross a placement/routing grid square, the threshold density is obtained. The contribution of nets below this threshold is minimal because even if many of those nets overlap (up to the maximum number of overlapping nets) only a maximum density equal to the overuse threshold will be contributed.

All the nets are considered when re-computing the overuse cost between anneal temperatures, but during a temperature only non-ignored nets are updated when performing overuse updates. It turns out that using this technique with the density threshold described earlier, only 13% percent of the nets are typically ignored, the placement results are not affected, and the run time is improved by approximately an order of magnitude.

Taking a step back, by reducing the coarseness of the congestion grid in step with tile shrinkage, several advantages are obtained. The preciseness of the congestion measurement is increased as more local placement improvements are made near the end of optimization. When the tile is larger, nets tend to be spread out over a larger distance; in fact, all features of the tile tend to be spread out, so a coarser congestion grid at this time makes sense; coarse features and regions are coarsely monitored. This also keeps the relative number of bins crossed by a set of nets grossly at the same value throughout the anneal. When the regions and features shrink, the coarseness grid shrinks with them, keeping the same relative areas of the tile under supervision by the same squares of the congestion grid.

The specification of grid coarseness is made in terms of the number of routing grid squares enclosed by a congestion grid square (when the tile is highly compacted). Therefore, tiles with more cell area will have a greater number of congestion grid squares. Consider this opposed to the increased number of routing grid squares in a congestion grid square, for tiles with larger cell area, if the width and height of the congestion grid was specified instead. Assuming that local regions in tiles with a larger total cell area are similar to local regions in tiles with a smaller total cell area (with

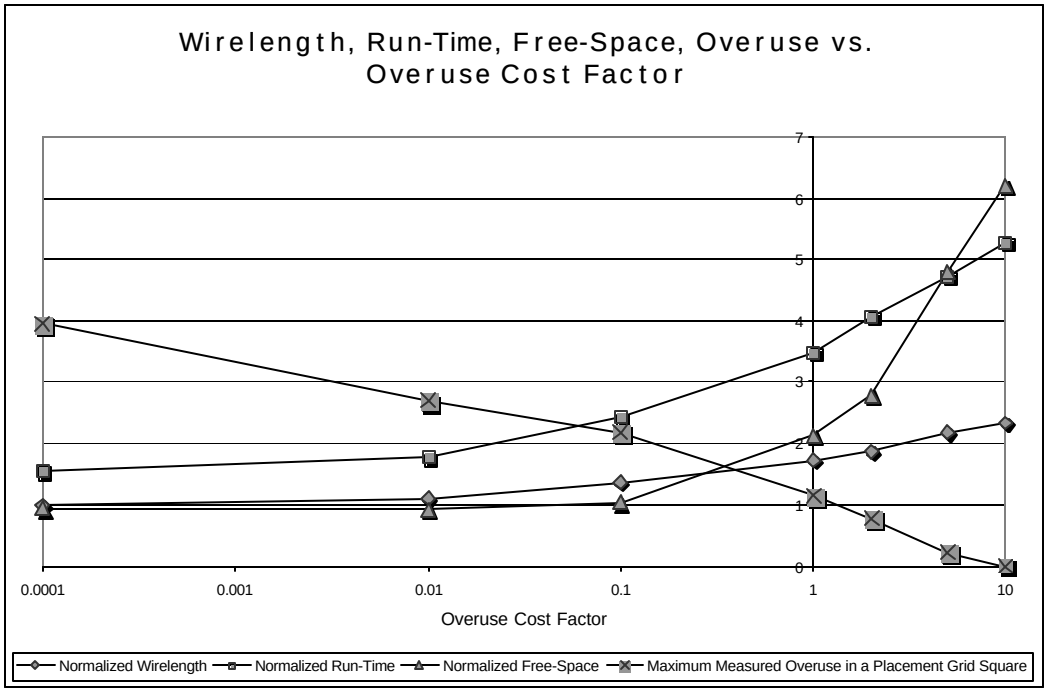
respect to routing congestion), a congestion grid square should have the same number of constituent routing grid squares to monitor the congestion appropriately for differing tile sizes. Experiments show that this assumption is reasonably accurate (allows for better optimization of the congestion model). One would expect, however, that the longer average distance of cell interconnections within larger tiles would tend to result in extra routing passing over cells that is connecting more remote parts of the tile. Nevertheless, this affect would tend to suggest more metal layers are needed to route such tiles, changing the congestion threshold but not the congestion grid coarseness or precision.

The coarseness of the congestion grid was specified so that 15x15 placement grid squares lie within a congestion grid square when the tile is highly compacted. This degree of coarseness seemed to produce the best results in terms of run-time and placement quality.

Of course, all these considerations and assumptions, and the underlying congestion model, which motivated the various decisions, should be tested when a router is available. That is why, other than the results presented next, the experimental results presented in this paper do not consider overuse cost and the congestion model.

To control how strongly the layout optimizer prioritizes overuse reduction, an overuse cost factor is multiplied by the raw overuse cost before adding it to the total cost. Graph 4-2 illustrates the effects of different overuse cost factors. The overuse threshold used is 5.0; as indicated earlier, this value produced good results as measured by the overuse model. Values higher than this did not restrain hotspot magnitude as much and values lower than this could not be satisfied, even with considerable placement degradation.

The maximum overuse measured in a placement grid square (Graph 4-2) is a measure of the overuse in the hottest hot-spot in the tile (usage above the overuse threshold). Wirelength, free space, and run time are all normalized with respect to a run which does not monitor overuse and does not suffer from any of the overhead of the respective computations. The number of hot-spots in the tile was measured, at the end of optimization, by counting how many placement grid squares had a predicted quantity of routing above them greater than the overuse threshold. The percentage of placement squares, which were "hot", was below 4% for all (> 0) overuse cost factor values tested. An overuse cost factor of 10 reduced this count to 0 for all tiles in the benchmark set.



Graph 4-2 Wirelength, Run-Time, Free-Space, Overuse vs. Overuse Cost Factor

Notice how the run-time increases as the overuse cost factor increases. This effect is due to the fact that congestion monitoring inhibits tile compaction because the respective moves that lead to tile compaction tend to increase congestion. Therefore, tile

compaction occurs over a longer period of time as the optimizer struggles to balance all the optimization goals. Therefore, the final optimizer exit criteria (based on tile compactness) are not satisfied as quickly when the optimizer focuses on reducing congestion. This effect is minimized by lower overuse cost factor values. The actual overhead of computations performed to monitor congestion can be more readily observed at those lower values. An overhead of about 60% run time is incurred due to the monitoring of congestion. This relatively small overhead is due to the carefully chosen algorithmic tradeoffs discussed above.

Of course, the maximum measured overuse is decreased as the overuse cost factor is increased. This comes at the expense of increased overall wirelength and tile free-space as these optimization goals are sacrificed in favour of limiting overuse.

4.4 MOVE-BASED OPTIMIZATIONS

Secondly, moved-based optimizations are presented, which expand, direct, skew, or modify the annealer's move space to facilitate the achievement of optimization goals.

4.4.1 BLOCK-OFF-EDGE MOVE

At lower placement temperatures, large cost increases are likely to be rejected. Therefore, cell movements over great distances that destroy good global placement choices are likely to be rejected. The starting temperature of the reheat anneals is chosen to be high enough that useful progress (tile shrinkage and wirelength improvement, for example) through cost "hill climbing" can be achieved; nevertheless to prevent the destruction of globally good placement decisions, a lower temperature is beneficial. In

general, to preserve placement decisions, the temperature is selected to be relatively low. To achieve adequate tile compaction, however, some cells have to be moved great distances – in order to get off the edge of the "imaginary" bounding box – despite the fact that competing costs resist such a move. The reason for long-range moves is generally because near the end of placement optimization, when the center of the tile is highly compacted, large clusters of cells often congregate in one section near the edge of the tile. There is no room to move the cells towards the tile center and no short sequence of local moves (likely to be proposed) can move the respective cells off the tile edge. The tile-slope and tile-size compaction cost factors can be increased to the point where they dominate so that such long-distance moves are accepted, but such increased focus on tile compaction generates lower-quality placements. Furthermore, there is a more fundamental problem that such long-distance moves are generally not proposed during the reheat anneals because the relatively low temperatures motivate small range limits. Consider Figure 4-4 which was captured from an actual placement run without block-off-edge moves. Notice how there are cells near the edge of the tile preventing tile collapse, even though there is free space available to accommodate them. A combination of the range-limit imposed on them and the cost penalty in moving them limits tile compaction.

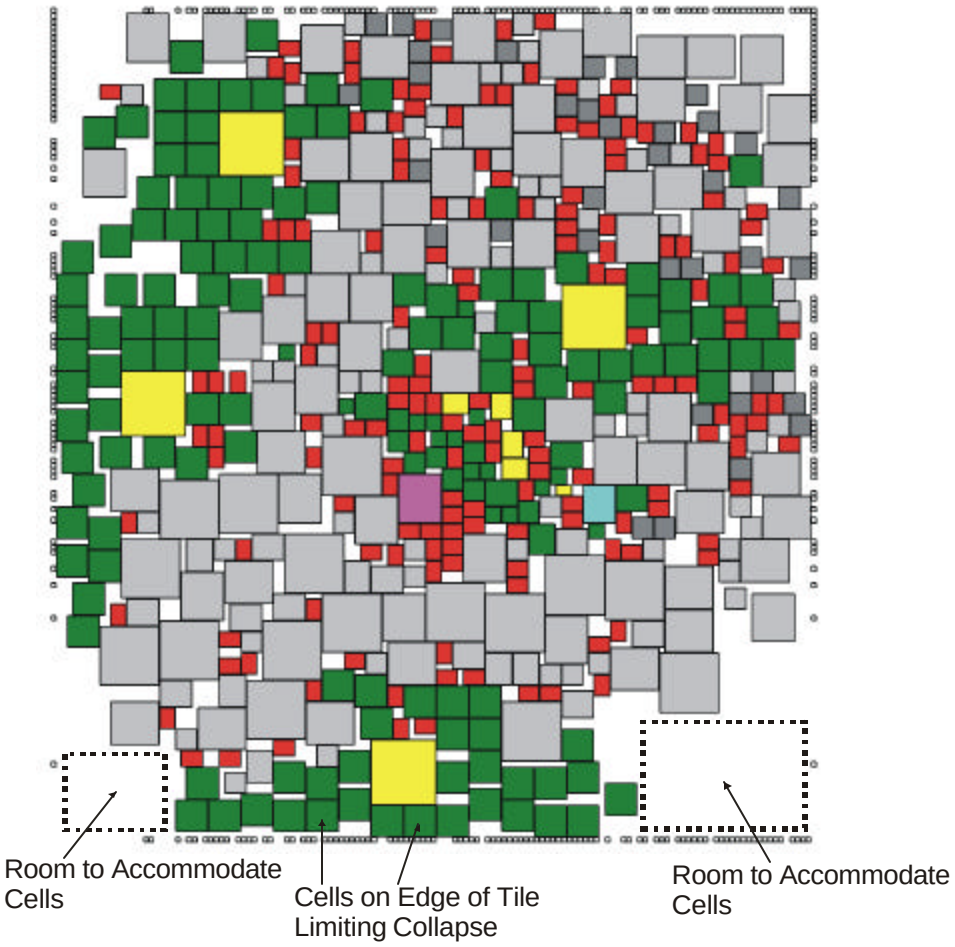


Figure 4-4 Example of Cells Limiting Collapse and Space that can Accommodate Them

Block-off-edge moves are strictly designed to propose the long-distance moves necessary to permit tile compaction and to provide the cost bonuses needed to ensure those specific moves are accepted by the annealer. These moves ignore particular range limits. Whenever a cell on the edge of the "imaginary" bounding-box is selected for moving, the range limit in the direction parallel to the edge is removed, so that the proposed move destination can be anywhere along the edge (but just inside the edge). Refer to Figure 4-5 for an illustration of how the range-limit is modified for this move. This solves the problem of beneficial move proposal by allowing long-range moves specifically for cells that need to be moved to facilitate compaction. If the proposed move

is to a location that can accept the cell without disturbing other cells and that location is inside the edge of the "imaginary" bounding-box, a cost bonus is applied. This solves the problem of ensuring such "necessary" moves are accepted. The cost bonus is simply performed by dividing the resulting cost difference by a cost-bonus divisor before the difference is used to determine proposed move acceptance or rejection. If the cost difference indicates a cost decrease, the bonus has no effect; the move is accepted no matter what. If the cost difference indicates a cost increase, the perceived magnitude of the cost increase will be reduced, so the move is more likely to be accepted. Of course, large cost increases are still likely to be rejected (maintaining some degree of cost arbitration). Nevertheless, if the cost-bonus factor is adjusted appropriately, "necessary" moves can be more frequently proposed and accepted without seriously harming the overall placement quality; such a decrease in overall placement quality manifests if the alternative of increasing the compaction cost factors is employed instead.

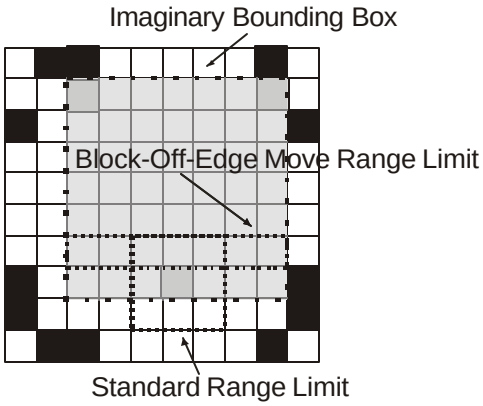
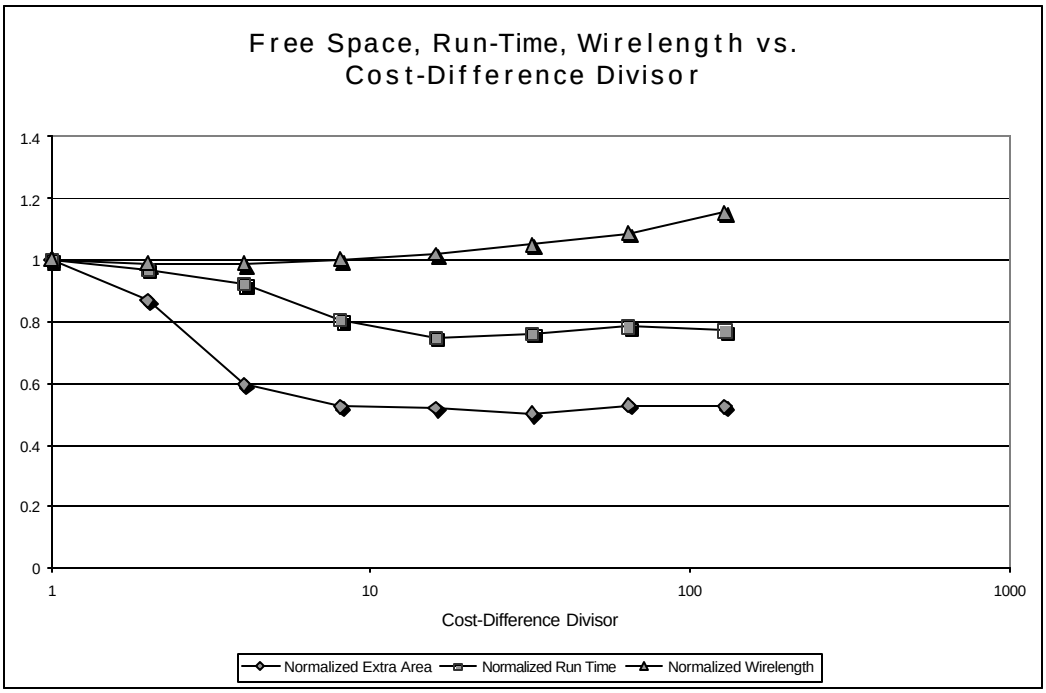


Figure 4-5 Effect of Block-Off-Edge Move on Range Limit

It turns out that even if the range-limit is extended, the cost division (cost bonus) is necessary to achieve any sort of difference in results. Furthermore, if the cost bonus is implemented in isolation, placement results degrade without compaction benefits. Graph

4-3 illustrates the affect on placement results of changing the cost divisor for Block-Off-Edge moves. The results are presented assuming that Block-Off-Edge moves are executed half of the time when it is possible to execute them (the move involves a block on the imaginary-bounding box edge); it turns out that this value produces the best results. It should be noted that as the divisor is increased, the final free space area is almost immediately reduced to a good value. The run-time decrease is similarly immediate; run time decrease makes sense because tile compaction occurs more effectively, and, the optimizer's exit criteria based on tile compaction is satisfied sooner (refer to Appendix A for more details). The wirelength increase is steady as the divisor is increased, which makes sense, because any increase in wirelength cost is diluted by the divisor. Notice how the wirelength dips for small divisor values, however; this is probably because the decrease in area benefits overall wirelength mitigating the impact of the cost divisor. A cost divisor of 8 is selected because it achieved the free space and run-time benefit without affecting wirelength. Table 4-3 indicates the detailed results of a comparison between a cost divisor of 1 and a cost divisor of 8 for the netlists in the benchmark set. Notice the general improvement in free space and run time without the large impact on wirelength.



Graph 4-3 Free Space, Run-Time, Wirelength vs. Cost-Difference Divisor

Table 4-3 Effect of Block-Off-Edge Move Divisor

	Cost Difference Divisor: 1 (?)			**Cost Difference Divisor: 8 (?)**			**Improvement (? Result/? Result)**		
Number of LUTs	*Free Space / Cell Area*	*Run Time (s)*	*Wirelength*	*Free Space / Cell Area*	*Run Time (s)*	*Wirelength*	*Free Space / Cell Area*	*Run Time*	*Wirelength*
1	0.14	332	36712.5	0.12	229	34934.9	0.85	0.69	0.95
2	0.36	341	41867.1	0.11	313	40676.6	0.31	0.92	0.97
3	0.24	548	47611.4	0.12	368	49379.9	0.48	0.67	1.04
4	0.33	400	47963.2	0.13	362	50119.5	0.38	0.91	1.04
5	0.35	685	62061.7	0.12	603	59356.1	0.34	0.88	0.96
6	0.18	969	67861.3	0.12	804	69523.7	0.63	0.83	1.02
7	0.25	1140	74078.3	0.11	900	73966	0.43	0.79	1.00
8	0.19	1382	80286.5	0.13	1002	80581.8	0.67	0.73	1.00
9	0.21	1302	87013.5	0.11	1087	88724.5	0.53	0.83	1.02
10	0.19	1687	98628.2	0.12	1393	100638	0.62	0.83	1.02
Average							**0.52**	**0.81**	**1.00**

4.4.2 COMPACTION MOVE

Even with tile-compaction cost bonuses designed to benefit moves that lead to smaller tiles, combinations and sequences of those moves have to be proposed by the move generator for tile compaction to result. It is unlikely that the move generator is able

to generate the proper sequences randomly and frequently. For example, consider the following sets of blocks, with the indicated connections, that have to move to the right to permit tile-area shrinkage (Figure 4-6):

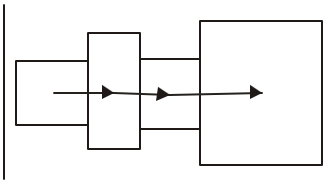


Figure 4-6 All blocks must move right to permit tile shrinkage.

Assuming the blocks are “tightly” held in their respective relative positions by the connections that run between them, any “out-of-order” moves which drastically change the relative positions of the blocks are likely to be rejected (Figure 4-7):

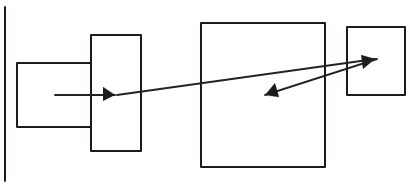


Figure 4-7 Move likely rejected because of unfavourable impact on other optimization goals.

Consequently, each block can move to the right only after the block ahead of it is moved (Figure 4-8):

The tile-slope compaction cost will bonus moves to the right so that moves which result in connection cost tie-breaks will tend to be resolved in favour of rightward movement (over time). Nevertheless, the correct succession of moves have to be proposed for this to happen efficiently (reasonable run-time) and effectively (for example, taking advantage of gaps which temporarily open between blocks by squeezing a block between them).

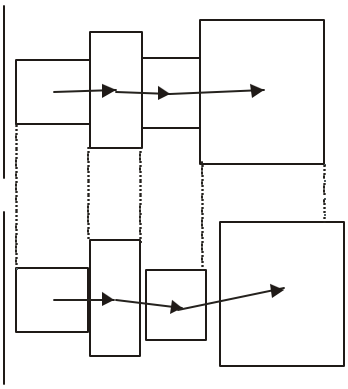


Figure 4-8 The "correct" move sequence will successfully move the blocks off the edge.

The solution is to introduce a type of multiple-block move, which attempts to successively move all blocks to the center of the tile one after another. By making compaction a priority of the move generator, more efficient, and effective tile compaction can take place.

On average the move generator selects this multiple-block move a particular number of times per temperature. The move considers all blocks, one after another. Blocks closer to the tile center (Manhattan-wise) are considered first. The following pseudo-code summarizes the compaction move process:

```
COMPACTION_MOVE_TYPE := SELECT ONE OF FOUR COMPACTION MOVE TYPES
FOR EACH CELL, STARTING WITH CELLS CLOSER TO TILE CENTER
        PICK A DIRECTION TO NUDGE THE CELL CLOSER TO THE CENTER (BASED ON THE COMPACTION MOVE TYPE AND THE
                CELL POSITION)

        IF THE CELL MOVE WOULD CREATE AN ILLEGAL PLACEMENT
                REJECT PROPOSED MOVE
        ELSE IF PROPOSED MOVE IS REJECTED FOR COST REASONS
                DO NOT PERFORM MOVE
        ELSE
                PERFORM PROPOSED MOVE AND UPDATE COST
```

Notice that all cells may be moved by a compaction move. In that sense, this move is an all-block or all-cell move. There are four types of compaction moves: (1) move blocks above center downwards and move blocks below center upwards (Figure 4-9); (2) move blocks left of center rightwards and move blocks right of center leftwards

(Figure 4-10); (3) move blocks above center downwards and move blocks below center upwards, only if they are not horizontally (as opposed to vertically) more distant from the center (Figure 4-11); (4) move blocks left of center rightwards and move blocks right of center leftwards, only if they are not vertically (as opposed to horizontally) more distant from the center (Figure 4-12).

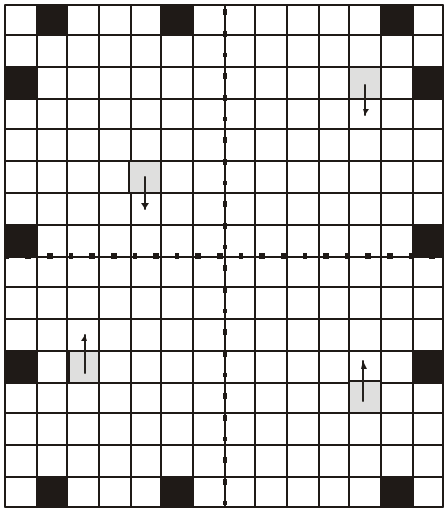


Figure 4-9 Proposed Moves of First Compaction Move Type

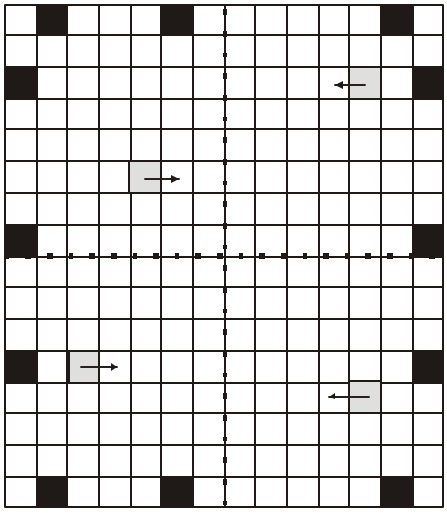


Figure 4-10 Proposed Moves of Second Compaction Move Type

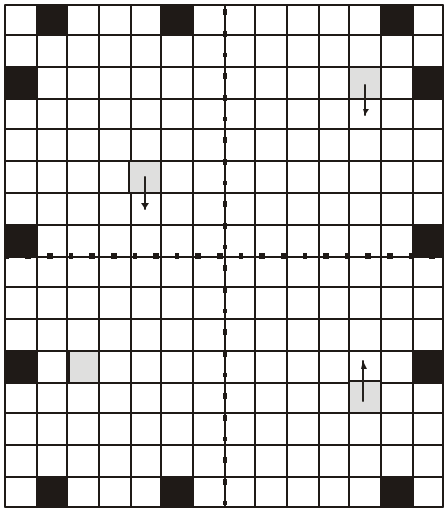


Figure 4-11 Proposed Moves of Third Compaction Move Type

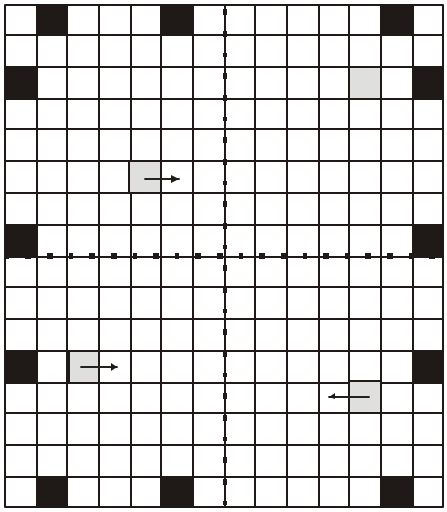


Figure 4-12 Proposed Moves of Fourth Compaction Move Type

Move types (1) and (2) tend to produce t-shaped arrangements as blocks are squashed along vertical and horizontal central axes (Figure 4-13). Move types (3) and (4) tend to produce x-shaped arrangements (Figure 4-14). Therefore, if either (1) and (2), or (3) and (4) are used in isolation, large portions of free space are left at the edges of the tile. By utilizing all four types of moves, overall compaction takes place to fit the rectangular shape of the tile.

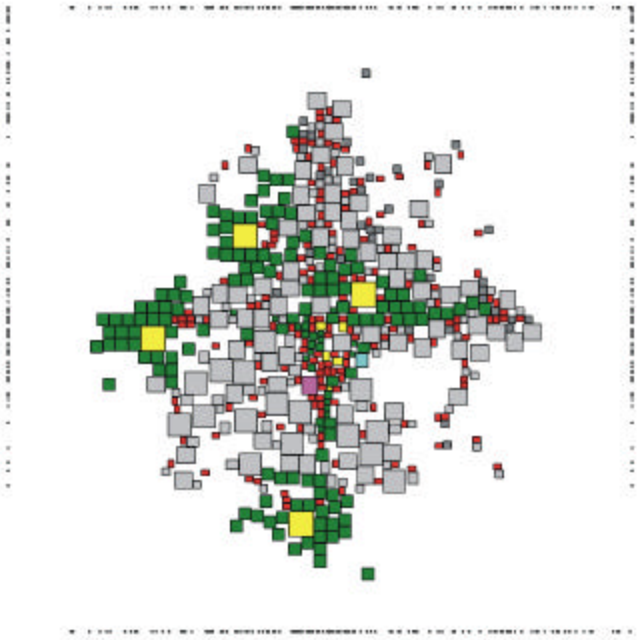


Figure 4-13 t-Shaped Arrangement Produced by Compaction Move Types (1) and (2)

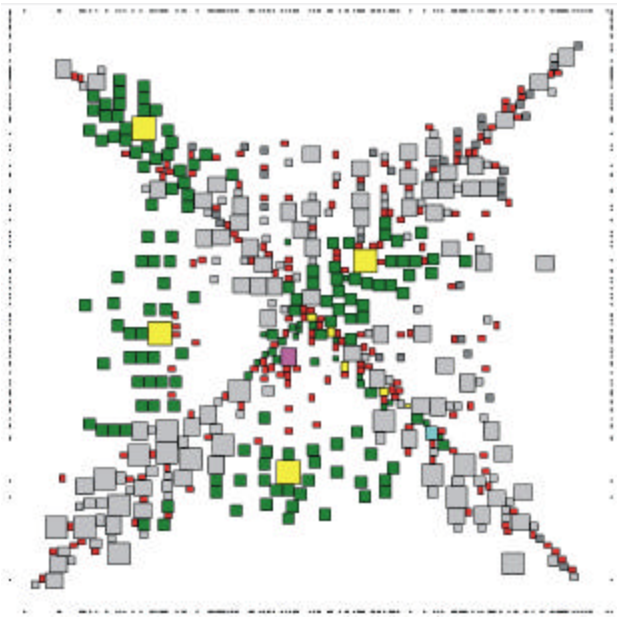


Figure 4-14 x-Shaped Arrangement Produced by Compaction Move Types (3) and (4)

One unit (distance) moves are suggested (proposed) for each of the blocks. These proposed "shift" moves are still subject to the same cost arbitration as a normal move.

Therefore, all the specialized move generation really does is sequence and promote "beneficial" moves instead of relying on fortune and long run times.

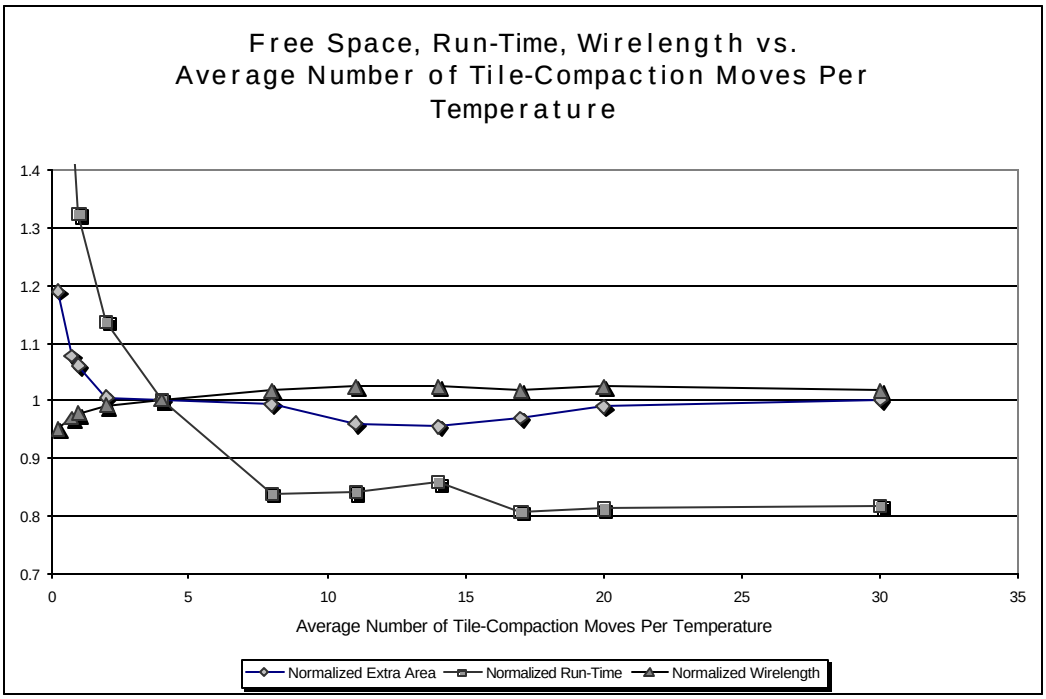
Multiple-unit (distance) moves were experimented with and showed no advantage over single-unit moves. The compaction move types all propose one-dimensional moves. Two-dimensional moves were experimented with; however, there was no perceivable advantage over one-dimensional moves. In fact, blocks tended to get in the way of each other, obstructing efficient compaction.

Since compaction moves affect all cells, they are performed relatively infrequently. The frequency they are performed with is specified as the average number of times they are performed per temperature. If compaction moves are not performed, the run time of the tool increases greatly because compaction occurs very slowly. Therefore, when the performance of 0 compaction moves per temperature was compared with the performance of 4 compaction moves per temperature (in Table 4-4), the tool was terminated in the former case once the respective run times were over twice as long as in the latter case; therefore, the reported free space percentages for the former case are not final values. Nevertheless, notice the dramatic improvement in free space and run time.

Table 4-4 Compaction-Move Effect

	Average of 0 Compaction Moves per Temperature (?)		**Average of 4 Compaction Moves per Temperature (?)**		**Improvement (? Result/? Result)**	
Number of LUTs	*Free Space / Cell Area*	*Run Time (s)*	*Free Space / Cell Area*	*Run Time (s)*	*Free Space / Cell Area*	*Run Time*
1	2.60	1148	0.12	236	0.047	0.206
2	2.83	907	0.12	372	0.044	0.410
3	3.61	1886	0.11	595	0.031	0.315
4	1.81	2341	0.10	509	0.057	0.217
5	4.14	5821	0.13	852	0.030	0.146
6	2.71	15501	0.13	1116	0.049	0.072
7	1.37	> 2798	0.11	1399	N/A	N/A
8	5.05	> 3146	0.12	1573	N/A	N/A
9	4.79	> 4040	0.11	2020	N/A	N/A
10	5.34	> 4460	0.11	2230	N/A	N/A
Average					**0.043**	**0.228**

By changing the average number of multiple-block compaction moves per temperature, the following trends can be observed (Graph 4-4).



Graph 4-4 Free Space, Run-Time, Wirelength vs. Average Number of Tile Compaction Moves per Temperature

Notice the run time improves as the tile-compaction frequency is increased. This only happens up to a point because the tile-compaction moves are relatively expensive (each move involves all blocks). The reason an upwards trend in run-time is not observed is because once rapid compaction takes place, the optimizer is likely to experience a series of reheat anneals without additional compaction; hence, the optimizer's exit criterion is satisfied and the optimizer exits (terminating execution and keeping the run time small).

Overall wirelength begins to increase somewhat. This relatively "weak increase" is probably due to the fact that rapid tile compression results in compacted tiles sooner. Compacted tiles inhibit the ability of blocks to achieve adequately "good positions"

relative to one another because a large portion of proposed moves are infeasible due to block overlap.

Extra area (free space) is reduced up to a point. As the more effective tile-compaction move frequency is increased, surplus tile area is initially reduced because compaction opportunities can be better and more frequently leveraged (for example, squeezing between blocks temporarily separated). This trend continues until the tile-compaction rate is increased to the point where the essentially greedy multiple-block compaction takes place almost immediately after compaction opportunities present themselves. If rapid tile compaction takes place, followed by a series of reheat anneals without apparent forward progress, the optimizer's exit criterion will be satisfied and the optimizer will terminate before the other compaction mechanisms can effectively engage and explore compaction possibilities.

4.4.3 BLOCK ROTATION AND FLIP (CELL TEMPLATE SWAP)

The cell specification in the netlist assumes particular layout orientations for the cells. Cell layout orientation determines cell pin placement and, consequently, the particular layout orientation affects placement quality. There is nothing that necessitates a particular layout orientation because cell layouts can be flipped and rotated to a certain extent re-arranging the relative cell pin positions. The netlist builder does not consider cell placements when determining the orientation, therefore, the orientation it chooses may be highly sub-optimal.

By expanding the placer move space to explore cell layout possibilities, cell orientations can be considered and explored during placement optimization.

Nevertheless, not only does this move potentially rotate cells and change relative pin placements, it also potentially affects cell aspect ratios allowing cells to better abut and squeeze between other cells. Nevertheless, currently the netlist builder chooses aspect ratios for all of the cells, which are close to square, so such benefits are not readily realized during actual placement.

Block rotation and flip moves are facilitated by the creation of cell templates in the placer. These templates are loaded from the initial netlist specification to represent all possible flips and rotations the cell can undergo without requiring re-layout. During placement, each cell has a set of templates that can be imposed to re-arrange pin positions and resize the cell to represent flip and/or rotation possibilities. When the block rotation and flip move imposes a new cell template, it maintains the position of the lower-left corner of the cell. The placement change this move results in is measured with the standard cost arbitration scheme and the move can be accepted or rejected (undone) based on the resulting cost difference and the current placement temperature.

Experiments were formed to determine if cell repositioning, at the same time as template swapping, was beneficial. It turns out that there is no advantage in coupling those two types of moves. Therefore, for the sake of move generation simplicity, both types of moves are proposed independently (and do not occur together). Perhaps, there is an added benefit in that this encourages smaller overall placement changes. Annealers appear to function best when the cost space is uniform and the moves selected from the move space result in small placement perturbations. If the move space is such that drastically different placements result from every move, the placer randomly explores wider scattered regions of the cost space rather than carefully considering one local

region before moving on to another.

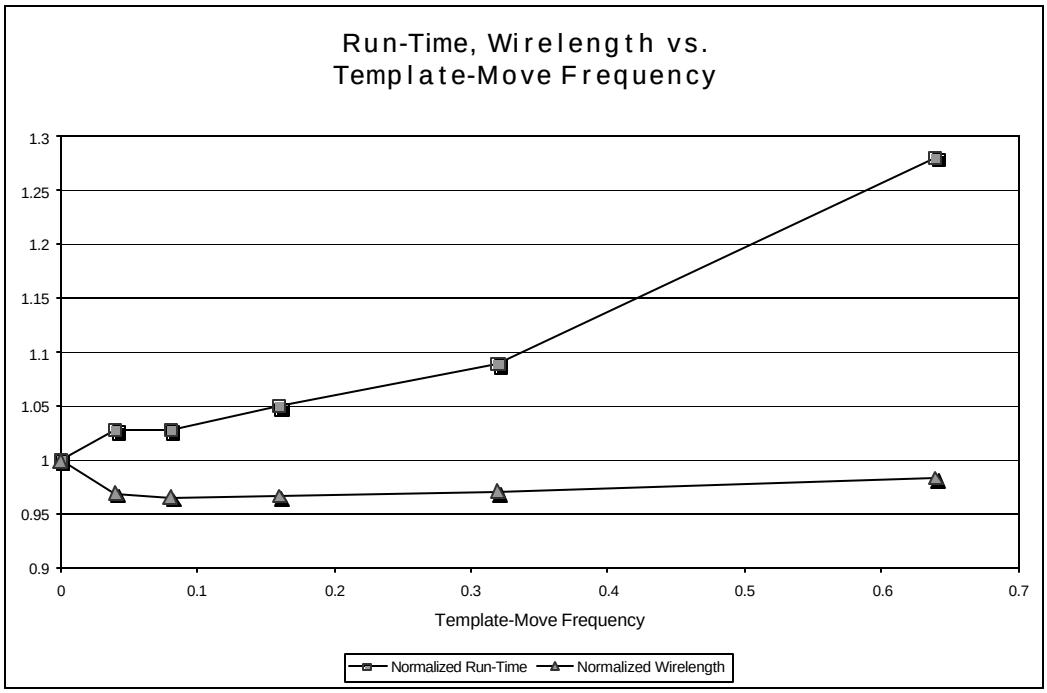
While design rules permit all cell re-orientations, transistor characteristics, due to uncertain mask alignment and patterning in orthogonal directions in modern semiconductor processes, might be largely affected by 90-degree rotations. [9] Assuming CMOS-technology is used, speed would only be affected, circuit functionality should not. Therefore, the degrees of freedom offered to this type of move should be reconsidered when speed (performance) considerations are integrated into the tool or the cells are laid out using a technology other than CMOS whose functionality is affected by transistor characteristics.

Furthermore, currently the tool does not consider cell layouts that are beyond simple re-orientations of the cell specification in the netlist. One natural extension is to permit the tool to read in several different layout possibilities for several cells allowing it to explore highly different cell aspect ratios and pin possibilities when placing cells. At that point, the decoupling between cell translations and template changes might be reconsidered, because a compound move might be able to explore areas of the cost space not readily reachable by a series of independent translations and template changes. That said, as discussed in 5.2, experimentation indicates that highly non-square aspect ratios are generally detrimental; hence, exploring drastically different cell aspect ratios might not be beneficial. Regardless, adding consideration of many cell templates, to explore an expanded range of layout possibilities, should be relatively easy and should not require changes to the optimizer, only changes to the optimizer's data structure loaders (informing it of the layout possibilities); that is, the template swapping code is currently general enough to handle such swaps.

The frequency of template moves (percentage of moves which are template moves) is varied to produce Graph 4-5. There is no noticeable change is free space percentage as the frequency of this move is changed within reasonable values. Notice the increase in run time as the template move frequency is increased. This increase in run time is not due to the fact that template moves are more expensive than regular moves. The increase in run time is due to the fact that by performing template moves, the optimizer is distracted from performing other moves, especially those that contribute to tile compaction. Therefore, the optimizer does not complete (with a highly compacted tile) until it has run for a longer time. Nevertheless, notice the improvement in wirelength due to this type of move. The trend of improving wirelength reverses itself when this move is performed very frequently because this biased emphasis on template moves detracts from overall placement quality. Table 4-5 indicates that template moves reduce wirelength by about 3%.

Table 4-5 Effect of Template Moves

	Template Move Frequency: 0 (?)		**Template Move Frequency: 0.08 (?)**		**Improvement (? Result / ? Result)**	
Number of LUTs	*Run Time (s)*	*Wirelength*	*Run Time (s)*	*Wirelength*	*Run Time*	*Wirelength*
1	539	35826	552	34250.3	1.02	0.96
2	808	44826.5	784	41743.4	0.97	0.93
3	1134	50149.1	1176	49894.2	1.04	0.99
4	1077	48382.8	976	47729.6	0.91	0.99
5	1860	62605.7	1956	60973.1	1.05	0.97
6	2468	71493.2	2578	69401.9	1.04	0.97
7	3107	79121.9	3225	75749.3	1.04	0.96
8	2773	84448.8	3043	79296.3	1.10	0.94
9	3520	91003.4	3517	89498.3	1.00	0.98
10	5400	104721	5979	100888	1.11	0.96
Average					**1.03**	**0.97**



Graph 4-5 Run Time, Wirelength vs. Template Move Frequency

4.4.4 BLOCK EQUIVALENT-PIN SWAP

Particular cells have equivalent pins in the sense that a given set of connections attached to those pins can be swapped without affecting the functionality of the circuit. For example, a programmable multiplexer's inputs are equivalent in the sense that connections to its inputs can be swapped freely because the SRAM cells which control the multiplexer can be programmed to any values. Netlist generation makes the various connections to equivalent pins without considering placement, therefore, it makes no effort to optimize the relevant connections for placement considerations.

A mechanism is created (block equivalent-pin swap) to allow the placer to swap connections to equivalent pins. That way, as the placer places cells, the connectivity can be swapped (arbitrated by the cost function) to achieve greater placement improvement through an expansion of the proposed move space. During netlist loading, atomic pin

groups and compatible pin groups are formed based on the cells read in. Each compatible group contains a set of two or more atomic pin groups. An atomic pin group defines a set of pins whose connections must be swapped with those of another atomic pin group (in the same compatible group). All the connections corresponding to an atomic pin group must be swapped with the corresponding connections of the other atomic pin group; that is, the atomic pin grouping must be maintained (hence, the term atomic).

A few examples can help illustrate this. Consider the control pins of a multiplexer. To facilitate swapping of control pins the netlist loader would create the following compatible group with the corresponding atomic pin groups:

```
COMPATIBLE_GROUP_1 {
        ATOMIC_PIN_GROUP_1(CONTROL_A, NOT_CONTROL_A)
        ATOMIC_PIN_GROUP_2(CONTROL_B, NOT_CONTROL_B)
        ATOMIC_PIN_GROUP_3(CONTROL_C, NOT_CONTROL_C)
}
```

During the anneal, if a pin swap move is selected, and compatible_group_1 of the respective block is selected (a connection swap between control pins), atomic_pin_group_1, atomic_pin_group_2, and atomic_pin_group_3 would be available for swapping. If atomic_pin_group_1 and atomic_pin_group_2 are selected for swapping, the connections attached to CONTROL_A and CONTROL_B will be swapped and the connections attached to NOT_CONTROL_A and NOT_CONTROL_B will be swapped. Therefore, the respective atomic pin groupings are maintained; that is, swapping CONTROL_A and CONTROL_B can not occur in isolation because the atomic pin groupings of compatible_group_1 guarantee both the CONTROL pins and the NOT_CONTROL pins will be swapped together. This is important because the functionality of the multiplexer could not be preserved if SRAM ? was connected to CONTROL_A and NOT_CONTROL_B, for example.

Of course, single pin swaps are also supported by this scheme by including only a single pin in each atomic pin group. Consider the three additional compatible pin groups:

```
COMPATIBLE_GROUP_2 {
        ATOMIC_PIN_GROUP_1(CONTROL_A)
        ATOMIC_PIN_GROUP_2(NOT_CONTROL_A)
}
COMPATIBLE_GROUP_3 {
        ATOMIC_PIN_GROUP_1(CONTROL_B)
        ATOMIC_PIN_GROUP_2(NOT_CONTROL_B)
}
COMPATIBLE_GROUP_4 {
        ATOMIC_PIN_GROUP_1(CONTROL_C)
        ATOMIC_PIN_GROUP_2(NOT_CONTROL_C)
}
```

These groups would facilitate swapping of the connections attached to the CONTROL and NOT_CONTROL pins; logically, this is permitted because of the flexibility offered by the arbitrariness of SRAM programming – simply programming an SRAM bit to its complement value would make the resulting circuit function equivalently.

Another potential application of the connection-swapping mechanism is in the utilization of layout options. Often cell layouts provide a choice of several locations where a particular signal can be connected. These options may take the form of several exposed ports that a signal can be connected to or a location where a via can be placed to get a signal out. In any case, these options allow flexible use of the cell in that connections can be made to the closest convenient option rather than having to target a single pin. By creating compatibility groups to represent these options, a given connection can be swapped between these various possibilities with the placer cost function arbitrating the choice. Consider the following compatible group as an example:

```
COMPATIBLE_GROUP_5 {
        ATOMIC_PIN_GROUP_1(CLOCK_OPTION_1)
        ATOMIC_PIN_GROUP_2(CLOCK_OPTION_2)
        ATOMIC_PIN_GROUP_3(CLOCK_OPTION_3)
}
```

This compatible group can be used to represent the various pins to which a clock signal can be routed. The netlist generator will randomly select a given option (such as CLOCK_OPTION_1) to connect the clock to. The pin-swapping mechanism in the placer will then automatically consider swapping the clock connection to the other options. The placer currently supports option optimization, however, the netlists explored do not specify options and consequently their introduction is left to future work.

For the cells considered in the explored netlists, multiplexers, SRAMs, and LUTs are viable candidates for equivalent-pin swapping. All the input pins of a multiplexer are candidates for connection swapping, because the programming of the SRAM control bits of the multiplexer can be adjusted. The connections attached to a control pin and its complement can be swapped as discussed above. Pairs of control/control_complement pins can be swapped, as long as the pairings are maintained through atomic pin groups as discussed above. Connections attached to the output pins of an SRAM can be swapped, again because of programming flexibility. Finally, LUT inputs can be swapped (in pairs) because the SRAM programming bits programming the LUT can be adjusted in response; the connections attached to a LUT-input pin and its complement pin can be similarly swapped; also, the connections programming the LUT from SRAM cells can be swapped freely as well.

It turns out that a 1.4% improvement in wirelength can be achieved without sacrificing tile area compaction or run-time speed. This modest improvement is over that achieved by block rotation and flips which sort out inappropriate cell connectivity (for a given placement) to a certain extent. Nevertheless, the power of this move type will be uncovered primarily through the exploration of options in the cell layouts that should be

explored in future work.

4.5 OTHER OPTIMIZATIONS

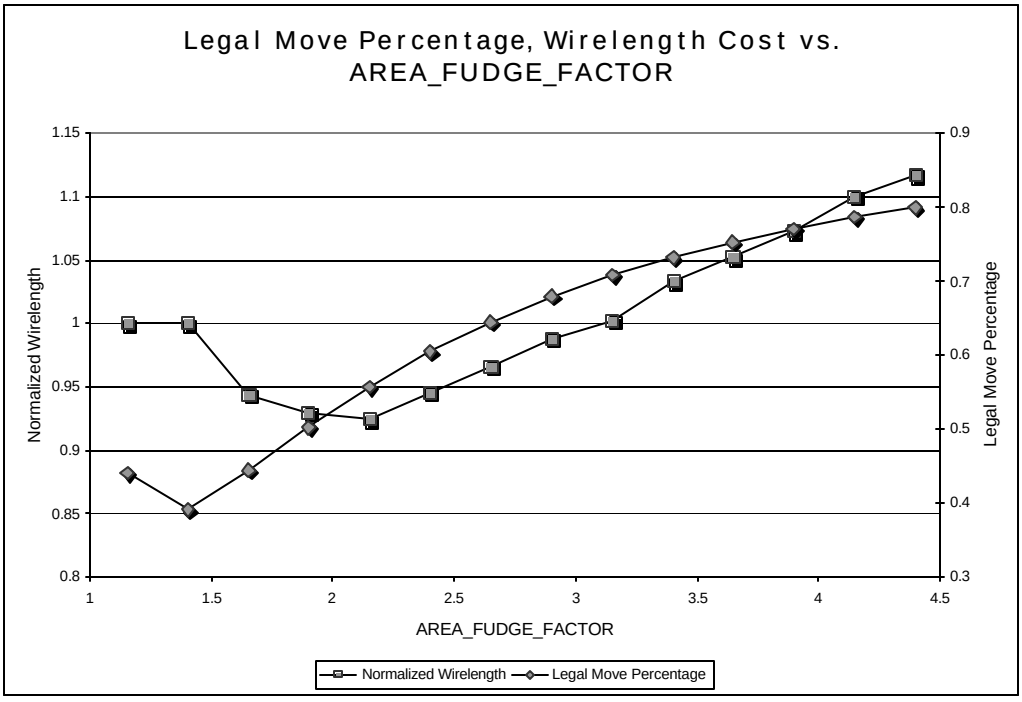
Thirdly, optimizations which are not entirely move-based or cost-based are presented.

4.5.1 INITIAL LARGE-GRID PLACEMENT

Achieving small area tiles is an optimization goal. Earlier work on ATL started off with a relatively small tile area and tried to optimize wirelength within that area (there was no mechanism to shrink the tile throughout the anneal). The initial tile area was 1.4 times the total area occupied by the cells, and this value was empirically determined as a good value because initial placement would fail for lower values. Besides the limitation of having tile area determined by initial placement capability, there is a fundamental optimization problem/tradeoff created by this approach. The smaller the initial tile area, the better the final tile area and, hence, the greater the satisfaction of a low-area goal. Nevertheless, the smaller the tile area, the less freedom of movement available to the move generator. More compact placements are inherently crowded with cells. The placement move generator operates by picking a random cell to move and a random location to move the cell to; as described in 2.5.2.4, the placer move generator gives up if it can not quickly select a set of cells to move which will result in a post-move legal placement. Therefore, the more crowded the cells, the more difficult it is to find legal moves to propose. This lack of move freedom reduces the optimization effectiveness of

the placer because the annealer can not explore the cost space effectively. Therefore, there is a trade-off, between smaller tile areas and lower utilization of routing resources (effective optimization of wirelength cost), present in the earlier work on ATL.

This tradeoff in the initial version of ATL can be readily observed. Notice in Graph 4-6 how the legal move percentage increases as the AREA_FUDGE_FACTOR is increased (increasing the tile area). Also notice how the normalized wirelength improves steadily as the area is initially increased. This is despite the fact that larger tile areas mean larger port/pin distances and more spread out nets (more wiring). Therefore, the ability of the optimizer to perform effectively is greatly inhibited by small initial tile sizes because of cell crowding.



Graph 4-6 Legal Move Percentage, Wirelength vs. AREA_FUDGE_FACTOR

This research modified ATL so it currently optimizes for tile area by alternating between arranging the cells to permit tile shrinkage (balanced with other optimization

objectives) and collapsing the tile, until further tile collapse can not “easily” be achieved. Therefore, there is no advantage in selecting a small initial tile size because the tile area will be optimized throughout the placement (balanced dynamically with other optimization goals). Considering the impact of cell crowding on wirelength and other optimization goals, it is advantageous to select a large initial tile size to permit cells to achieve globally good positions, with local optimizations occurring as the tile is shrunk. An initial large-grid placement (refer to Figure 4-15 for an illustration of the spacing between cells that characterizes this large-grid placement) anneal serves this purpose. Subsequent reheat anneals between tile shrinks serve to make the local optimizations necessary to fix changes due to the tile shrink and to further optimize the placement as cells find good positions around each other in the context of the smaller tile.

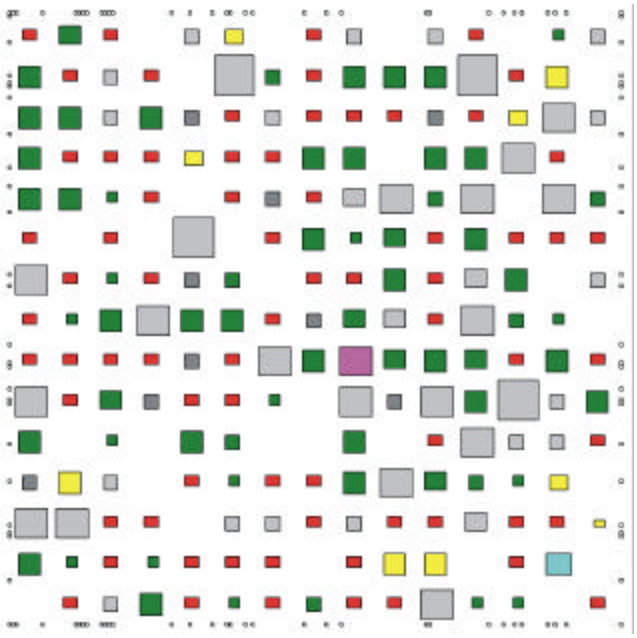


Figure 4-15 Initial Large-Grid Placement

The goal of the large-grid initial placement is to give cells the movement freedom and flexibility to achieve good global positions relative to one another. Therefore, the

initial grid is sized large enough so there is enough space available to place all cells and cells are separated at regular distances (determined by the largest cell dimensions) so they can swap freely with one another. That way, illegal moves are no longer a consideration. This freedom of movement tends to produce good global placement results despite the fact that cells are artificially kept apart, and are prevented from bunching. Global placement seems to be at least as important, if not more important, than locally realistic placement – perhaps, because the sorting out of local positions occurs in the later phases. The ports are still placed in "standard" port slots because they can always swap freely with one another. The large initial placement grid also has an added bonus of making initial placement trivial (the tile is sized to explicitly fit every cell in a separate non-overlapping section).

There is a natural reduction of cell movement freedom (a gradual solidifying of position) as the tile collapses. Cells can only jiggle around because substantial moves are likely to cause irreconcilable overlap with other cells (illegal proposed moves). Nevertheless, the good cell arrangements determined by the initial large-grid placement anneal and earlier reheat anneals should be more explicitly preserved. Consequently, the reheat anneals start with a low enough placement temperature to preserve placement quality for the most part, while leaving room for some hill-climbing capability.

This research considered acceptance ratio a good measure of anneal progress independent of the circuit netlist – precise temperature values are a certain function of cost and, hence, are netlist dependent. Therefore, the reheat anneal temperature is selected to be the temperature that achieves a particular acceptance ratio during the initial large-grid placement. Experiments show an acceptance ratio of 0.2875 yields good

results. The minimum range-limit used for the initial large-grid placement permits cells to swap with cells up to a distance equal to two times the "largest-cell dimension" away. It was found that this value gave cells adequate freedom to move while restricting the move space sufficiently (at lower temperatures) to adequately narrow the search space (propose fruitful moves).

Consider the difference this technique has on final wirelength (Table 4-6). The technique reduces wirelength by 37%. Both runs produced the same final fee space results (within experimental noise). Notice how the run with no initial large-grid placement, that starts with a smaller initial tile size, terminates sooner because tile compaction occurs sooner and consequently the final optimizer exit criterion is satisfied earlier.

Table 4-6 Effect of Initial Large-Grid Placement without Run-Time Adjustment

	No Initial Large-Grid Placement (Initial Tile Size 1.4 Times Cell Area) (?)		**Initial Large-Grid Placement (?)**		**Improvement (? Result / ? Result)**	
Number of LUTs	*Run Time (s)*	*Wirelength*	*Run Time (s)*	*Wirelength*	*Run Time*	*Wirelength*
1	29	46300.2	40	33187.9	1.38	0.72
2	41	61070.1	56	41846.9	1.37	0.69
3	55	76340.6	74	50587.1	1.35	0.66
4	38	79032.1	55	48972.8	1.45	0.62
5	64	100415	137	59091.6	2.14	0.59
6	90	119091	172	69541.4	1.91	0.58
7	112	126008	206	74605.5	1.84	0.59
8	119	128286	266	76651.6	2.24	0.60
9	146	142934	249	88005.3	1.71	0.62
10	183	158069	275	97323.9	1.50	0.62
Average					**1.69**	**0.63**

To ensure that both runs could be evaluated more fairly, the no initial large-grid placement run was allowed to spend more time on each anneal and reheat anneal temperature to balance the run times (Table 4-7). Again, essentially, the same final free space results were produced by both runs (within experimental noise). Notice how the technique still produces better wirelength results. In fact, the extra run time does very

little to improve the no initial large-grid placement results because of the inability to adequately explore the cost space.

Table 4-7 Effect of Initial Large-Grid Placement with Run-Time Adjustment

	No Initial Large-Grid Placement (Initial Tile Size 1.4 Times Cell Area) (?)	**Initial Large-Grid Placement (?)**	**Improvement (? Result / ? Result)**
Number of LUTs	*Wirelength*	*Wirelength*	*Wirelength*
1	43304.7	33187.9	0.77
2	61466.3	41846.9	0.68
3	72534.2	50587.1	0.70
4	68205.2	48972.8	0.72
5	93268	59091.6	0.63
6	109340	69541.4	0.64
7	122972	74605.5	0.61
8	118497	76651.6	0.65
9	133225	88005.3	0.66
10	154299	97323.9	0.63
Average			**0.67**

4.5.2 SRAM REWEAVE

The netlist generator does not consider the placement of the cells when it assigns SRAMs to word and bit lines. Therefore, for the most part, the word and bit line assignments are random in that cells ultimately placed in two remote locations of the tile may be attached to SRAMs driven by the same programming lines. Since the word and bit lines run the length of the tile, reducing the amount of zigzagging necessary to route them can result in a large savings of routing resources. Another way of looking at this is the placer will try hard to minimize the routing resources needed to route the SRAM word and bit lines, consequently tugging them away from logic they are attached to (or tugging that logic away with them). Either way, these two effects produce conflicting optimization goals (which the placer can not satisfy). Furthermore, the tradeoff created is artificial in the sense that it is a function of the arbitrary word and bit line assignments made by the netlist generator.

Consider Figure 4-16 for an illustration of a bad programming line assignment. The lines emanating from the SRAM, enclosed by the solid oval close to the center, connect to SRAMs (enclosed in dotted ovals) attached to this one by programming lines. Notice how the programming line assignments are such that the respective SRAM cell is attached to SRAMs throughout the tile because those SRAMs gravitate towards the logic they are attached to.

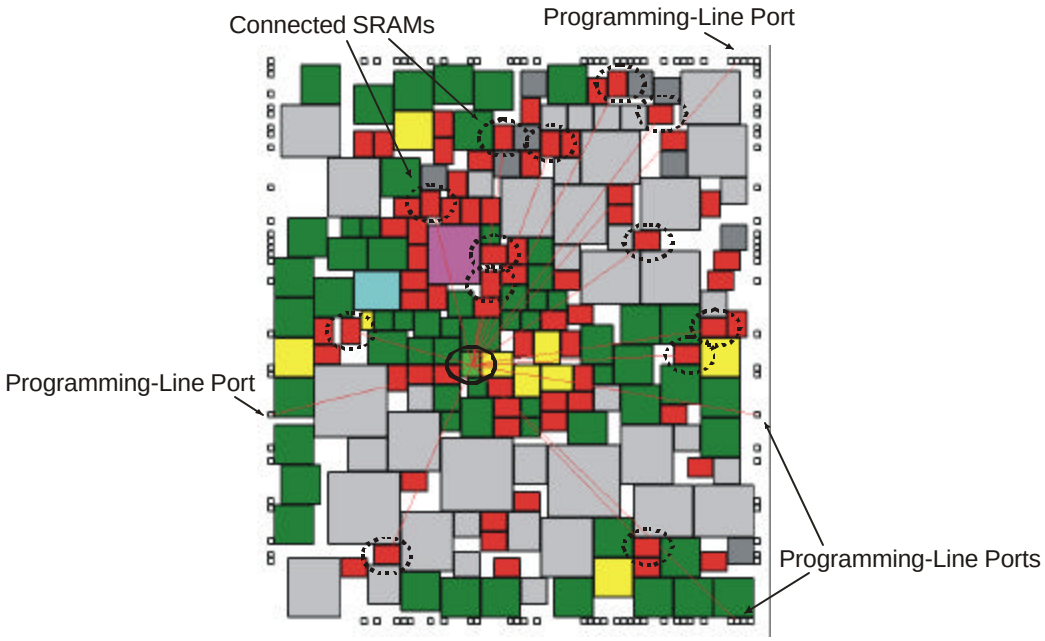


Figure 4-16 Illustration of Bad Programming-Line Assignment

A mechanism was created that is invoked between the reheat anneals to reweave the SRAM word and bit lines depending on the current placement. This decouples placement consideration of SRAM programming lines from the arbitrary assignments made by the netlist generator. In fact, the initial large-grid placement does not consider the cost of the programming lines at all. That way, the SRAMs can be tugged by the logic they are attached to without the arbitrary SRAM programming lines impairing movement. During the reheat anneals, the SRAM programming lines are monitored,

however, because they eventually have to be routed and should be considered when optimizing for wirelength and congestion.

The SRAM reweaving mechanism simply rips up all currently connected SRAM programming lines and then re-connects the SRAMs to their closest bit and word lines in order; it is performed between reheat anneals. To preserve the tile-ability of the tile, the ports associated with a given programming line are placed opposite one another at opposite ends of the tile. Therefore, the programming lines would ideally be one unit in width. The placement of their respective ports determine a natural ordering of the programming lines from left to right and from bottom to top. This ordering determines the reweaving. The bottom-most line is woven through the bottom-most SRAMs until the line is full. The next-bottom-most line is woven through the next-bottom-most SRAMs until that line is full. This process repeats until all the SRAMs are rewoven. Refer to Figure 4-17 for an illustration of the result of reweaving. In fact, it is the association of the SRAMs more than which ports they are associated with that matters, because the ports tend to move more freely to match the position of the SRAMs they are tied to rather than the other way around. This is probably due to the fact that the ports can move and swap with other ports without experiencing the overlap problems faced by the cells; also, the ports are not tied down like the SRAMs are to associated logic.

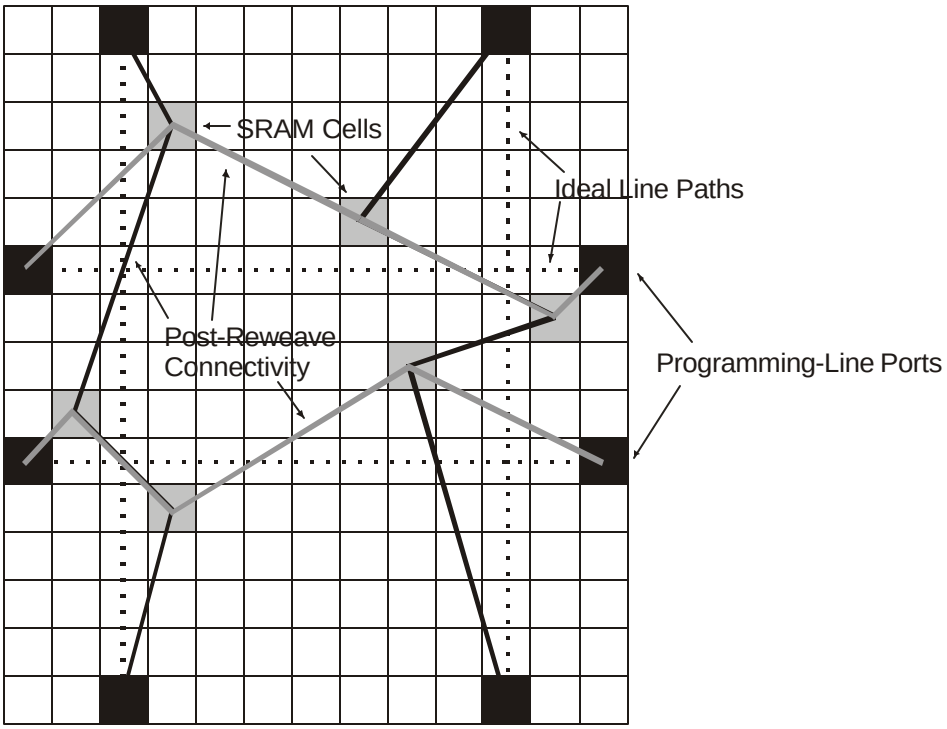


Figure 4-17 Example of an SRAM Reweaving

Notice how SRAM reweave capability can correct the bad programming-line assignment situation illustrated in Figure 4-16. The tile layout shown in Figure 4-18 illustrates a good programming line assignment achieved through the SRAM reweaving technique. Notice how the SRAMs, connected to the one inside the solid oval (same SRAM as in previous illustration, but simply rotated due to block rotation move), are aligned close to the ideal programming line paths; consequently, the connectivity is much less spread out.

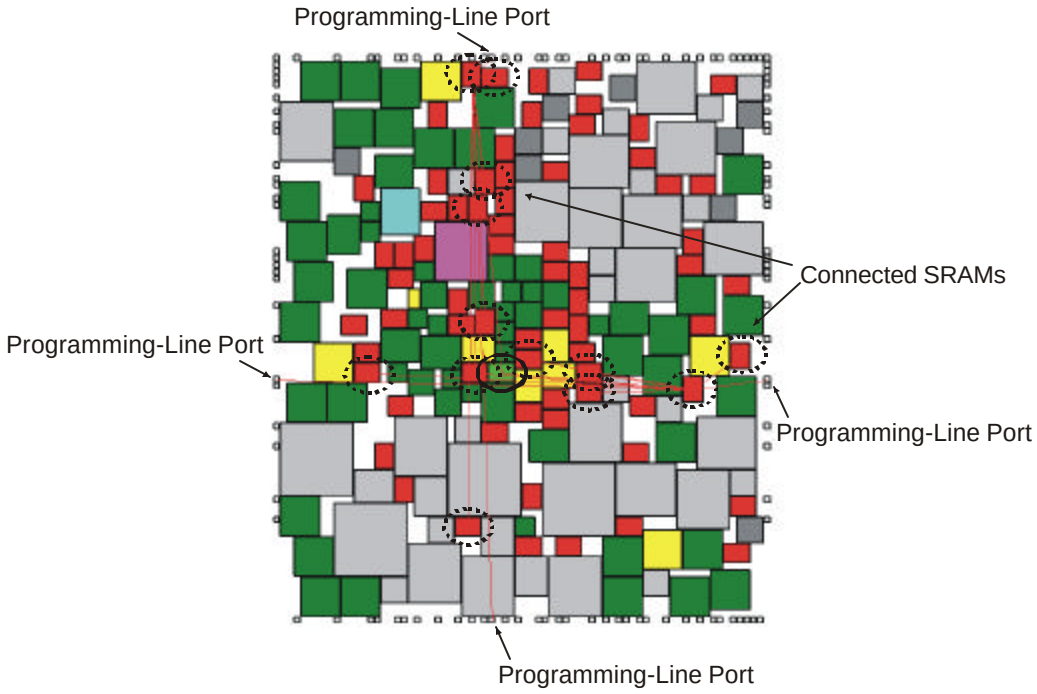


Figure 4-18 Illustration of Good Programming Line Assignment Achieved by SRAM Re weave

Table 4-8 Effect of SRAM Reweaving

	No SRAM Reweaving (?)	**SRAM Reweaving (?)**	**Improvement (? Result / ? Result)**
Number of LUTs	*Wirelength*	*Wirelength*	*Wirelength*
1	37679.8	33187.9	0.88
2	45947.8	41846.9	0.91
3	54596.2	50587.1	0.93
4	53482.9	48972.8	0.92
5	69831.6	59091.6	0.85
6	78806.2	69541.4	0.88
7	85432.9	74605.5	0.87
8	89288.7	76651.6	0.86
9	100515	88005.3	0.88
10	110434	97323.9	0.88
Average			**0.89**

Consider the impact SRAM reweaving has on wirelength results (Table 4-8). The no SRAM reweaving run constantly monitored programming lines throughout optimization because ultimately the respective connections were static (because of the lack of reweaving) and would have to be routed as is. The run times and the free space results of the two runs were the same (within experimental noise). Notice how the 11%

wirelength improvement is greater than the 4% observed in [1] (2.5.3) (when SRAMs were allowed to swap only within a regular row and column arrangement).

4.6 CUMULATIVE EFFECT OF OPTIMIZATION TECHNIQUES AND COSTS

The following results summarize the effect the implemented heuristics and costs have on placement quality (wirelength and tile area) considering placement run time. Both the initial ATL and the ATL produced by this research do not rely on particular cell sizes or the number of cells. They are designed to lay out FPGA tiles that consist of the respective cell types, irregardless of exact composition. To illustrate this, the results presented in Table 4-9 are based on 1-10 LUT architectures with more realistic amounts of wires per FPGA routing channel than those used in the benchmark circuits created by [1]; the wire counts were extrapolated from the number of wires (indicated in datasheets) used in Xilinx Inc.'s Virtex-E [10] and Virtex-II [11] architectures. Nevertheless, it should be noted that the results of the comparison apply to many varying architectures because both tools are independent of precise architecture details. Initial ATL run time was adjusted so that it had equal CPU time to that of the ATL from this research by increasing the number of moves it explores per temperature. The smallest tile (1 LUT, 436 ports and 542 cells) is laid out in about 30 seconds while the largest tile (10 LUT, 3256 ports and 4475 cells) takes about 1.5 hours to lay out. Initial ATL was modified to report final wirelength using the improved bounding-box measurement (3.3.3), after it completed optimization using its standard worst-case bounding-box measurement (2.5.2.3).

Table 4-9 Results of this Research Compared with those of Initial ATL (Tile-Area Factor: 1.4)

	Initial ATL (Tile-Area Factor: 1.4) (?)		**ATL with Heuristics from this Research (?)**		**Improvement (? Result / ? Result)**	
Number of LUTs	*Tile-Area (Placement/Routing Grid Squares)*	*Wirelength*	*Tile-Area (Placement/Routing Grid Squares)*	*Wirelength*	*Tile-Area (Placement/Routing Grid Squares)*	*Wirelength*
1	24336	38207.41	20083	30990.1	0.83	0.81
2	52441	101136.7	41814	75985.3	0.80	0.75
3	86436	182827.1	68096	131265	0.79	0.72
4	135424	227825.7	64770	159542	0.48	0.70
5	207025	358311.8	117294	256768	0.57	0.72
6	313600	496295.9	148980	339182	0.48	0.68
7	369664	594107.8	177177	410557	0.48	0.69
8	480249	727683	197132	477889	0.41	0.66
9	579121	864912	230868	550271	0.40	0.64
10	665856	978329	276315	629134	0.41	0.64
Average					**0.56**	**0.70**

The only mechanism available to decrease tile area using the initial version of ATL involved manually re-running the tool with lower and lower tile-area factors until initial placement failure or tile area was limited by port perimeter size (only one port per port location was supported).

Table 4-10 Results of this Research Compared with those of Initial ATL (Minimum Tile Area)

	Initial ATL (Minimum Tile Area) (?)		**ATL with Heuristics from this Research (?)**		**Improvement (? Result / ? Result)**	
Number of LUTs	*Tile-Area (Placement/Routing Grid Squares)*	*Wirelength*	*Tile-Area (Placement/Routing Grid Squares)*	*Wirelength*	*Tile-Area (Placement/Routing Grid Squares)*	*Wirelength*
1	22500	41967.36	20083	30990.1	0.89	0.74
2	49284	103646.6	41814	75985.3	0.85	0.73
3	82944	184952.3	68096	131265	0.82	0.71
4	135424	227825.7	64770	159542	0.48	0.70
5	207025	358311.8	117294	256768	0.57	0.72
6	313600	496295.9	148980	339182	0.48	0.68
7	369664	594107.8	177177	410557	0.48	0.69
8	480249	727683	197132	477889	0.41	0.66
9	579121	864912	230868	550271	0.40	0.64
10	665856	978329	276315	629134	0.41	0.64
Average					**0.58**	**0.69**

It turns out the initial version of ATL can only reduce the tile area of 1-, 2-, and 3-LUT architectures (Table 4-10). The other architecture tiles are limited by port perimeter. It should be noted that the tiles Initial ATL did shrink are still, at least, 10% larger than the tiles produced automatically by this research's ATL because Initial ATL can not fit

the cells within the tile (initial placement fails). Notice also how the decrease in tile area produced manually using Initial ATL is accompanied by an increase in wirelength. This is because even though the tile dimensions are smaller, non-optimal cell arrangements result because the tool can not optimize wirelength as effectively within the constraints of the smaller tile. This research's ATL produces smaller tile areas combined with less overall wirelength.

Chapter 5
CONCLUSION

5.1 FINAL RESULTS

As stated earlier, the goal of this research was to develop placement heuristics that could be incorporated into the Automatic-FPGA Tile Layout (ATL) tool that attempt to: (1) minimize the area of laid out tiles, (2) minimize the wire length needed to interconnect the cells and ports within the tile, and (3) balance wiring requirements over the tile area to prevent localized over-demand of wiring resources and, hence, avoid routing congestion.

This research has shown that various heuristics, some of which make use of knowledge of the circuitry within an FPGA, can be used to improve the layout performance of the tool. As presented in 4.6, the heuristics and costs added to ATL reduce the size of the tiles produced by 42%. The techniques developed also manage to simultaneously reduce overall wire length by 31%. Currently, the empty space in the layouts generated by ATL is, approximately, 10% of the actual cell area.

As discussed in 4.3.3, a congestion model was created to measure localized over-demand of wiring resources. A placement cost was developed along with heuristics to effectively and efficiently minimize the size and number of routing congestion "hot spots" across the chip, as measured by the congestion model proposed. It was shown that adjustment of the relevant overuse cost weighting can reduce congestion at the expense of increased overall wirelength and tile area. Further study and validation of the

congestion model, congestion cost, and the relevant heuristics are left to future work and analysis, in the presence of an FPGA tile router.

5.2 FUTURE WORK

The outcome of this research motivates several avenues of further investigation, some of which are discussed below.

As mentioned earlier, concurrent work was ongoing at the University of Toronto to develop an FPGA tile router. This router performs multi-layer metal routing of the connections attached to the cells and ports positioned (laid out) by the placement optimizer of ATL (the focus of this research). Future work can examine the congestion models and costs created by this research in the context of an actual FPGA tile router, once it is complete. Future work on the router can also examine the impact of routing the VDD, GND, and other "global signals" across the tile. It should be investigated whether the layout (placement) optimizer should consider those signals (currently ATL does not consider VDD and GND, but it does consider clocks) or whether their "global" nature implies direct consideration is detrimental.

Concurrent work was also ongoing at the University of Toronto to create actual transistor-level layouts for the functional cells floorplanned by the placement optimizer. Future work can incorporate those layout details in the netlists produced by VPR_LAYOUT so that ATL can generate layouts based on actual cell sizes and pin positions rather than the current estimates made by VPR_LAYOUT. Mechanisms should also be created to automatically generate transistor-level layouts of the cells to more fully automate the process.

Once layouts are created based on actual cell sizes and pin positions and those layouts are successfully routed (with the FPGA tile router and adequate congestion monitoring), performance of this integrated automatic layout mechanism can be compared with actual hand-layouts of FPGA tiles to analyze the time-quality trade-offs involved in the automation and convenience provided by VPR_LAYOUT and ATL.

Future work can also explore incorporating direct optimization of timing requirements within the FPGA tile. As mentioned in Chapter 1, the speed of operation of digital circuits implemented in FPGAs is an important factor that affects their practical use. By imposing strict timing requirements (carefully determined and monitored) when creating the FPGA tile, FPGAs can be produced that can implement high-speed digital designs. This work may also involve dynamic insertion and modification of logic in addition to layout optimization. Such work can also be extended to try to reduce FPGA power consumption. As mentioned previously in this paper, the reduction of wirelength (and, hence, resistance and capacitance) serves as a starting point for both these goals.

Other future work can examine, in more detail, fundamental assumptions in the netlist builder. For example, as discussed in 2.4.3, the cells considered by ATL have been bloated so that cells can be abutted against each other. That is, well spacing design rules are accommodated by increasing the cell sizes. Perhaps, directly monitoring well spacing during layout (placement) is more beneficial to reduce the overall cell area. Current, this research produces tile layouts with 10% free space, so any additional dramatic savings in tile area will be achieved by reducing cell area. Any effort on this front might also consider breaking up the cells into their constituent wells so that they can be placed next to each other, well contacts can be shared, etc.. A similar degree of abutment can be

achieved by pitch-matching the various cells like in a standard-cell flow. However, such constraints reduce the ability of the cells to "float" freely with respect to one another – hopefully, achieving more ideal positions. Also such constraints limit the freedom of the cells to be laid out in as little space as possible (no matter what shape, aspect ratio, they may end up in). It is this freedom to consider a range of highly-tuned cell layout alternatives that allows the tool to explore the range of possibilities available to a custom-layout engineer. Too large a space of layout possibilities may encumber the tool, however, so future work should examine these and similar tradeoffs in more detail.

The template moves and pin swaps discussed in 4.4.3 and 4.4.4 support different implementations of cells and different routing "options" associated with a cell (electrically equivalent contact points). Future work can explore this capability by creating new cell layouts that the tool can use during layout that have routing "options" that the tool can leverage. For example, good layouts of a cell with different aspect ratios can be created so the tool can dynamically determine which layouts are suited to the current tile placement. The move generator can also be experimented with to simultaneously move a cell at the same time as its aspect ratio is changed, for example. That said, this research did experiment which such moves, however, the particular move type experimented with allowed any area-preserving aspect ratio change to be made. It turned out that highly skewed (non-square) aspect ratios inhibited placement quality because long-thin cells tend to highly interfere with one another. Therefore, if experimentation is performed along these lines, highly skewed aspect ratios should be avoided or, at least, carefully explored.

By providing ATL with a variety of proposed cell templates to work with, the tool can also investigate the benefits of different cell layout alternatives prior to actual cell layout. Modifications can also be made to ATL to change cell aspect ratios and pin positions freely, outside of the confines of cell templates. This free exploration can be used to impose (determine) layout constraints for the cells. As indicated, experiments performed during this research suggest that aspect ratios should be constrained to be close to square. That said, since it is difficult for a layout designer (or custom-layout tool) to satisfy highly-specific (tight) cell constraints, it is unclear how fruitful this line of research is. The bottom-up flow is probably better, where cell layout engineers create a variety of highly-optimized cell layouts that the tool uses as effectively as it can.

As mentioned in 3.2.1, a multi-phase optimization scheme was developed to facilitate tile compaction during placement optimization. This research has shown that a tight integration between tile compaction and placement optimization is advantageous for balancing all the various optimization goals. Future work should explore a tighter integration between optimization and compaction that may be necessary to adequately avoid routing congestion – experienced by an actual FPGA tile router. If a new placement move responsible for tile shrinking (port collapse around cells) is implemented, this move can be made subject to the placement cost function like any other move. Therefore, tile shrinks can be aborted if the placement of cells imply a port collapse would create too much congestion around the edges, for example. Right now, the tile is shrunk between reheat anneals as much as the cell arrangement permits. By creating a tile-shrink placement move, compaction can be gradually (and slowly) performed as the anneal

proceeds; this would hopefully result in a smooth collapse of the tile and a gradual satisfaction of all placement goals.

From all these suggestions, it is clear that VPR_LAYOUT and ATL have opened up many avenues of future work that may profit from and build upon this research.

Appendix A
LAYOUT OPTIMIZER FLOW

The following pseudo-code summarizes the overall placement-phase flow:

```
BEGIN PLACEMENT-PHASE FLOW
        READ CELL-LEVEL AND TRANSISTOR-LEVEL NETLISTS FOR CELL AND PORT INFORMATION
        COMPUTE INITIAL PLACEMENT GRID SIZE TO ALLOW CELLS TO SWAP FREELY WITHOUT INTERFERENCE
        PERFORM INITIAL PLACEMENT TO CREATE AN INITIAL LEGAL PLACEMENT
        COMPUTE ALL PLACEMENT COSTS
        DETERMINE WHICH NETS UPDATE OVERUSE COST
        REMOVE COSTING OF SRAM LINES
        INITIALIZE TILE-SLOPE AND TILE-SIZE COST MULTIPLIERS TO 0
        COMPUTE INITIAL TEMPERATURE
        BEGIN INITIAL PLACEMENT ANNEAL
                FOR EACH PLACEMENT TEMPERATURE
                        PERFORM EITHER A STANDARD RANGE-LIMIT MOVE, BLOCK ROTATION AND FLIP, OR BLOCK
                                EQUIVALENT-PIN SWAP, KEEPING PORTS ALONG THE EDGE, CELLS INSIDE THE PORT
                                PERIMETER, AND THE TILE SIZE CONSTANT
                        REPEAT FOR DESIRED NUMBER OF MOVES IN TEMPERATURE

                        RECORD MAXIMUM TEMPERATURE (ß) THAT PRODUCES PROPOSED-MOVE ACCEPTANCE RATIO BELOW
                                a
                        REDUCE TEMPERATURE
                        ACCURATELY RE-COMPUTE WIRE OVERUSE COST BECAUSE INACCURACIES IN THIS COST BUILD UP
                                BECAUSE ONLY CERTAIN NETS UPDATE IT DURING AN ANNEAL TEMPERATURE
                        RE-DETERMINE WHICH NETS UPDATE OVERUSE COST
                        EXIT WHEN TEMPERATURE BELOW G AND THE COST DID NOT IMPROVE BY ? PERCENT OVER THE
                                TEMPERATURE
                END PLACEMENT TEMPERATURE
        END INITIAL PLACEMENT ANNEAL
        ADD BACK COSTING OF SRAM LINES
        BEGIN RE-HEAT AND TILE-SHRINK ITERATIONS
                SHRINK TILE BY MOVING ALL CELLS, AS A GROUP, MAINTAINING RELATIVE CELL POSITIONS, TO
                        BOTTOM-LEFT EDGE OF TILE, AND COLLAPSE PORTS AROUND CELLS
                RECORD TILE AREA IMPROVEMENT ?
                RE-WEAVE SRAM LINES
                RE-COMPUTE PLACEMENT COSTS
                ADJUST TILE-SLOPE AND TILE-SIZE COST MULTIPLIERS TO IMPROVE SHRINKAGE POTENTIAL
                BEGIN RE-HEAT PLACEMENT ANNEAL
                        REHEAT ANNEAL TEMPERATURE TO ß
                        FOR EACH PLACEMENT TEMPERATURE
                                PERFORM EITHER A STANDARD RANGE-LIMIT MOVE, A BLOCK-OFF-EDGE MOVE, BLOCK
                                        ROTATION AND FLIP, BLOCK EQUIVALENT PIN SWAP, OR COMPACTION
                                        MOVE, KEEPING PORTS ALONG THE EDGE, CELLS INSIDE THE PORT
                                        PERIMETER, AND THE TILE SIZE CONSTANT
                                REPEAT FOR DESIRED NUMBER OF MOVES IN TEMPERATURE

                                REDUCE TEMPERATURE
                                ACCURATELY RE-COMPUTE WIRE OVERUSE COST BECAUSE INACCURACIES IN THIS COST
                                        BUILD UP BECAUSE ONLY CERTAIN NETS UPDATE IT DURING AN ANNEAL
                                        TEMPERATURE
                                RE-DETERMINE WHICH NETS UPDATE OVERUSE COST
                                DETERMINE (?,?) BASED ON WHETHER TILE-AREA IMPROVEMENT HISTORY DICTATES
                                        THIS WILL BE THE LAST RE-HEAT ANNEAL
                                EXIT WHEN TEMPERATURE BELOW ? AND THE COST DID NOT IMPROVE BY ? PERCENT
                                        OVER THE TEMPERATURE
                        END PLACEMENT TEMPERATURE
                END RE-HEAT PLACEMENT ANNEAL
                EXIT WHEN TILE-AREA IMPROVEMENT (?) IS BELOW THRESHOLD d, FOR e SUCCESSIVE ANNEALS
        END RE-HEAT AND TILE-SHRINK ITERATIONS
```

END PLACEMENT-PHASE FLOW

NOTE: A LEGAL PLACEMENT IS ONE WITH THE PORTS ARRANGED ON THE PERIMETER SO THE TILE IS TILEABLE; LESS THAN ? PORTS ARE PLACED IN A GIVEN PORT POSITION; CELLS ARE PLACED IN THE INTERIOR OF THE TILE SO THEY DO NOT OVERLAP WITH EACH OTHER.

Appendix B
MOVE GENERATOR DETAILS

To support the variety of new placement moves, the move generator was modified to perform these moves. The following pseudo-code indicates the current steps in move generation:

```
MOVE_TYPE := RANDOMLY SELECT THE TYPE OF MOVE

SWITCH(MOVE_TYPE) {
        CASE STANDARD RANGE-LIMIT MOVE OR BLOCK-OFF-EDGE MOVE:
                PICK PRIMARY CELL TO MOVE AND DESTINATION LOCATION FOR CELL BASED ON RANGE LIMIT (OR
                        MODIFIED RANGE LIMIT FOR BLOCK-OFF-EDGE MOVE)
                GATHER OTHER CELLS (GROUP B) THAT THIS PRIMARY CELL DISPLACES TO MAINTAIN A LEGAL NON-
                        OVERLAPPING CELL PLACEMENT
                CONSIDER MOVING THESE CELLS INTO THE REGION THAT WOULD BE ABANDONED BY THE PRIMARY CELL
                GATHER THE CELLS AROUND THE PRIMARY CELL DISPLACED BY GROUP B CELLS; THESE CELLS, ALONG
                        WITH THE PRIMARY CELL, FORM GROUP A
                CONSIDER SWAPPING GROUP A AND B WITHOUT CREATING CELL OVERLAP

                IF THE MOVE WOULD CREATE AN ILLEGAL PLACEMENT
                        REJECT PROPOSED MOVE
                ELSE IF PROPOSED MOVE IS REJECTED FOR COST REASONS (OR MODIFIED COST DIFFERENCE FOR BLOCK-
                        OFF-EDGE MOVE)
                        DO NOT PERFORM MOVE
                ELSE
                        PERFORM PROPOSED MOVE AND UPDATE COST
        CASE BLOCK ROTATION AND FLIP:
                PICK PRIMARY CELL TO ROTATE OR FLIP
                PICK ROTATION AND FLIP TO PERFORM

                IF SELECTED RE-ORIENTATION OF CELL, IN PLACE, CREATES ILLEGAL PLACEMENT
                        REJECT PROPOSED MOVE
                ELSE IF PROPOSED MOVE IS REJECTED FOR COST REASONS
                        DO NOT PERFORM MOVE
                ELSE
                        PERFORM PROPOSED MOVE AND UPDATE COST
        CASE BLOCK EQUIVALENT-PIN SWAP:
                PICK PRIMARY CELL WHOSE CONNECTIONS WILL BE SWAPPED
                PICK PIN GROUP A ON THE BLOCK
                PICK PIN GROUP B ON THE BLOCK WHOSE CONNECTIONS CAN BE SWAPPED WITH CONNECTIONS ATTACHED TO
                        PIN GROUP A
                CONSIDER SWAPPING CONNECTIONS BETWEEN PIN GROUP A AND PIN GROUP B

                IF PROPOSED MOVE IS REJECTED FOR COST REASONS
                        DO NOT PERFORM MOVE
                ELSE
                        PERFORM PROPOSED MOVE AND UPDATE COST
        CASE COMPACTION MOVE:
                COMPACTION_MOVE_TYPE := SELECT ONE OF FOUR COMPACTION MOVE TYPES
                FOR EACH CELL, STARTING FROM CELLS CLOSER TO TILE CENTER
                        PICK A DIRECTION TO NUDGE THE CELL CLOSER TO THE CENTER (BASED ON THE COMPACTION
                                MOVE TYPE AND THE CELL POSITION)

                        IF THE CELL MOVE WOULD CREATE AN ILLEGAL PLACEMENT
                                REJECT PROPOSED MOVE
                        ELSE IF PROPOSED MOVE IS REJECTED FOR COST REASONS
                                DO NOT PERFORM MOVE
```

```
                    ELSE
                            PERFORM PROPOSED MOVE AND UPDATE COST
}
```

NOTE: THERE ARE TWO LEVELS OF MOVE REJECTION IN MOVE GENERATION AND ACCEPTANCE. MOVES MUST MAINTAIN LEGAL PLACEMENTS; THEREFORE, PROPOSED MOVES CAN BE REJECTED BECAUSE THEY WOULD CREATE ILLEGAL PLACEMENTS. THIS IS THE FIRST LEVEL OF MOVE REJECTION. MOVES CAN ALSO BE REJECTED BECAUSE THE LEGAL PLACEMENT THEY WOULD RESULT IS DEEMED WORSE THAN THE CURRENT PLACEMENT BECAUSE OF COST REASONS. THIS SECOND LEVEL OF MOVE REJECTION IS CONTROLLED BY THE PLACEMENT OPTIMIZATION SCHEME (SIMULATED-ANNEALING).

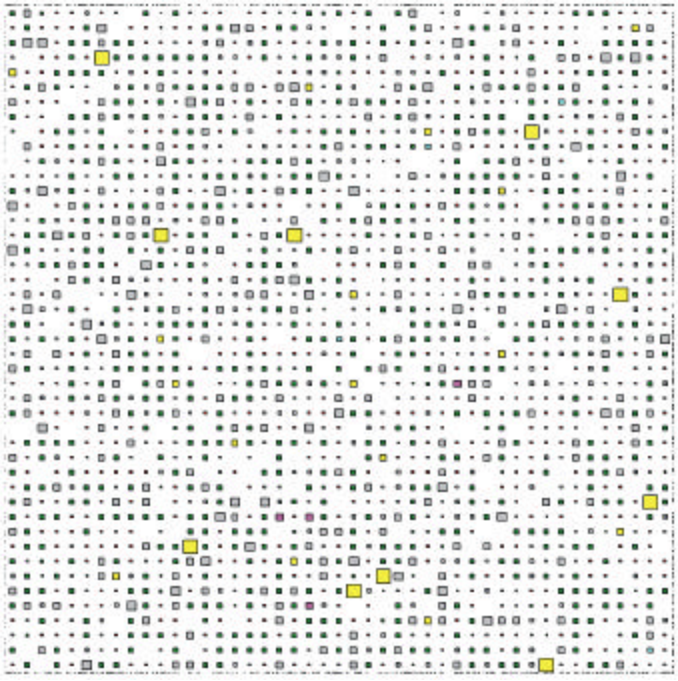
Appendix C
GRAPHICAL ILLUSTRATION OF ATL CELL PLACEMENT RUNS

ATL includes a graphical tool that displays the current cell placement of the FPGA tile under consideration. Using this tool, cell placements can be observed at various stages during the optimization process. This tool was initially created by [3], ported to the Microsoft Windows platform by [12], and extended by [1].

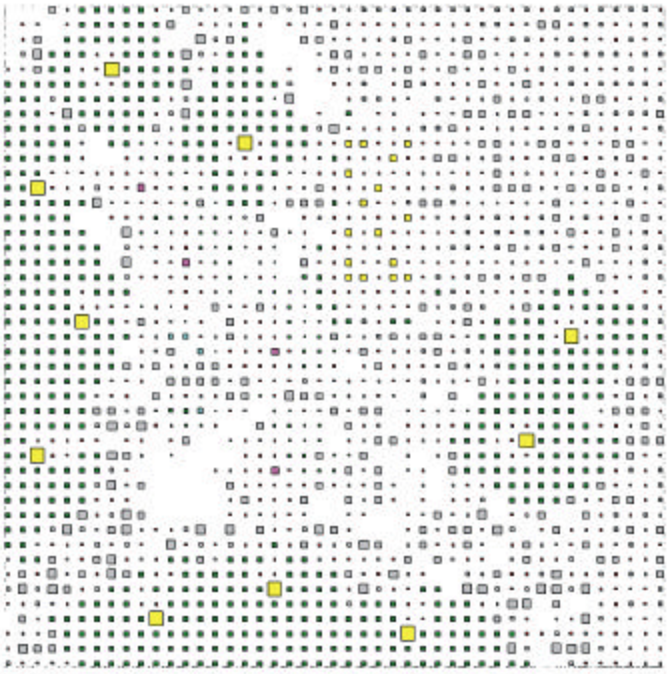
Two ATL layout runs are presented.

Both these runs use the version of ATL produced by this research. Both runs use the experimentally determined best (default) settings of the tool used to generate the experimental results in 4.6. These default settings turn off optimization of predicted routing congestion. It is recommended that this be the default behaviour of the tool until the underlying congestion model can be confirmed through experimentation, with a tile router.

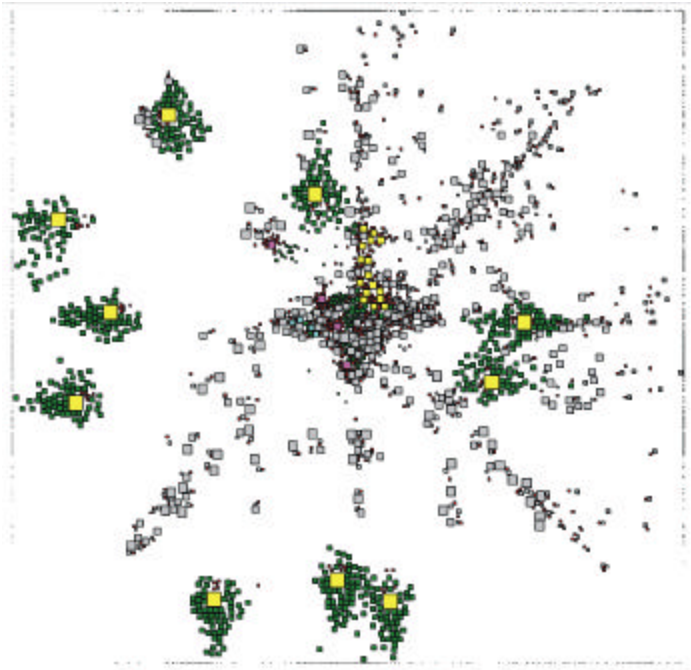
The first layout run illustrates the interim layouts generated by the ATL tool at various stages during optimization of the 4-LUT architecture used to generate the experimental results in 4.6.



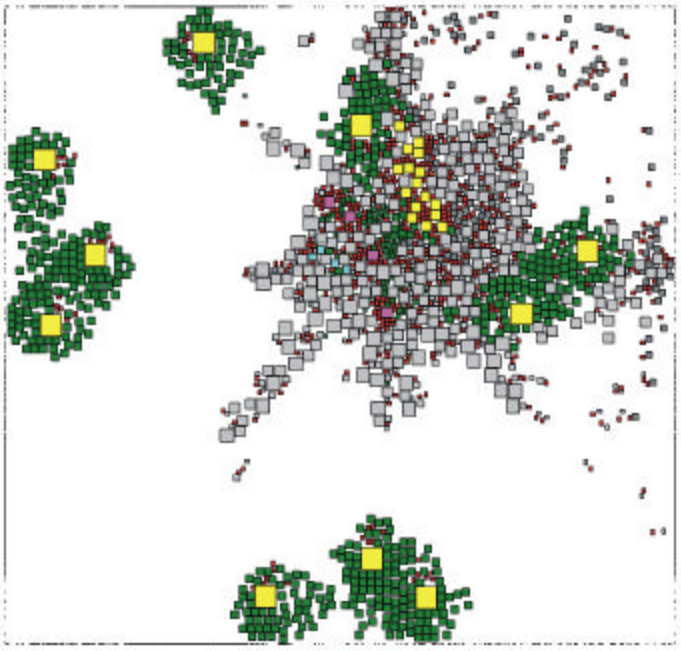
PLACEMENT TIME (0%)
WIRELENGTH COST: 1,597,020
TILE SIZE: 767X767 -- INITIAL ANNEAL (LARGE-GRID PLACEMENT)



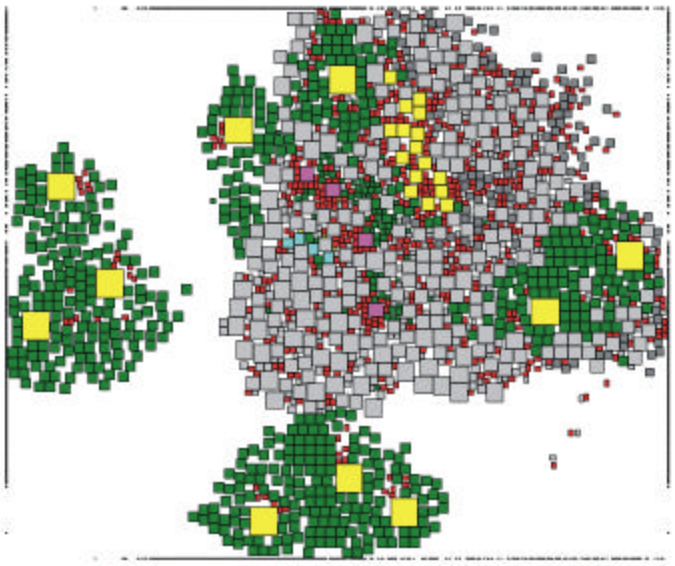
PLACEMENT COMPLETION (24%)
WIRELENGTH COST: 432,122
TILE SIZE: 757X757 -- AFTER INITIAL ANNEAL (LARGE-GRID PLACEMENT)



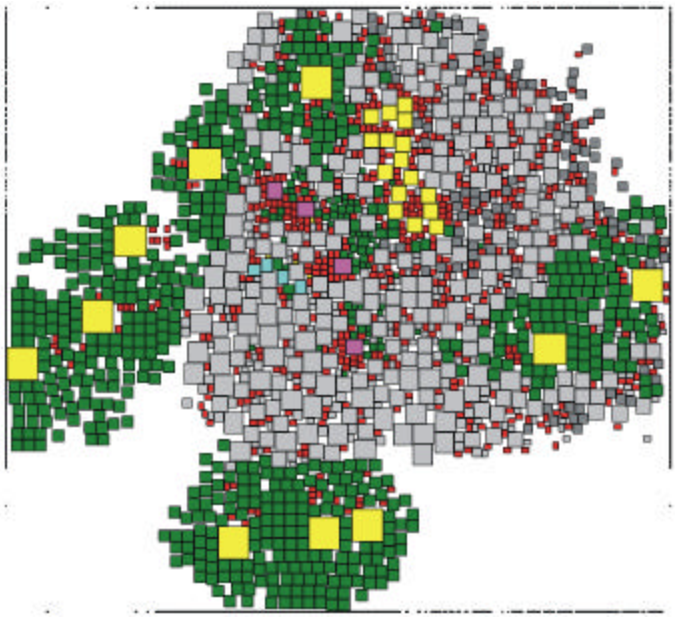
PLACEMENT COMPLETION (27.5%)
WIRELENGTH COST: 412,160
TILE SIZE: 733X714 -- BETWEEN RE-HEAT ANNEALS



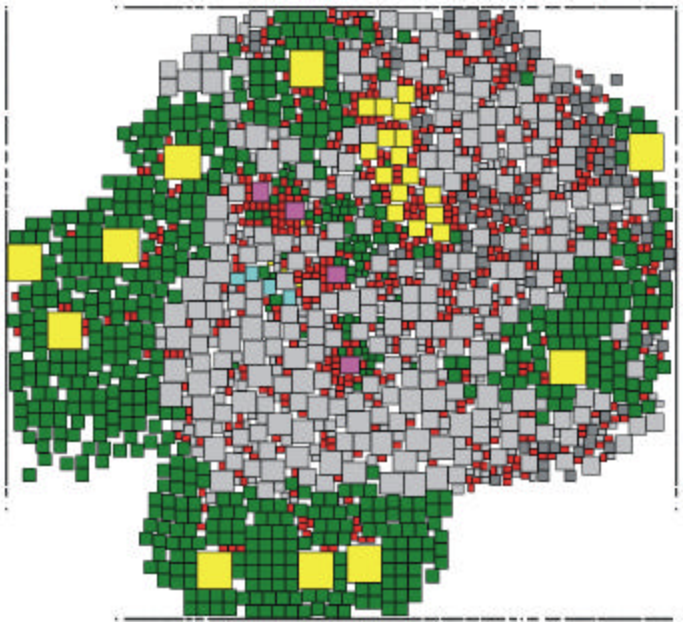
PLACEMENT COMPLETION (30%)
WIRELENGTH COST: 383,053
TILE SIZE: 496X473 -- BETWEEN RE-HEAT ANNEALS



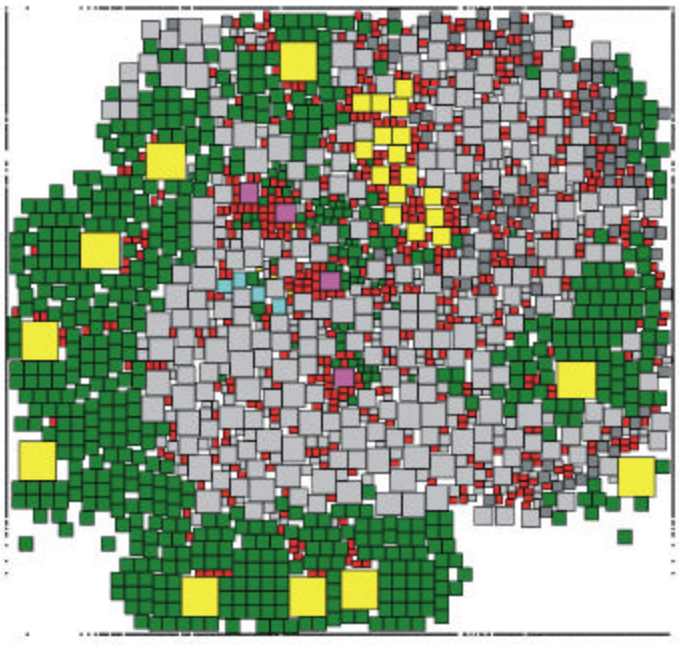
PLACEMENT COMPLETION (32.5%)
WIRELENGTH COST: 242,256
TILE SIZE: 388X323 -- BETWEEN RE-HEAT ANNEALS



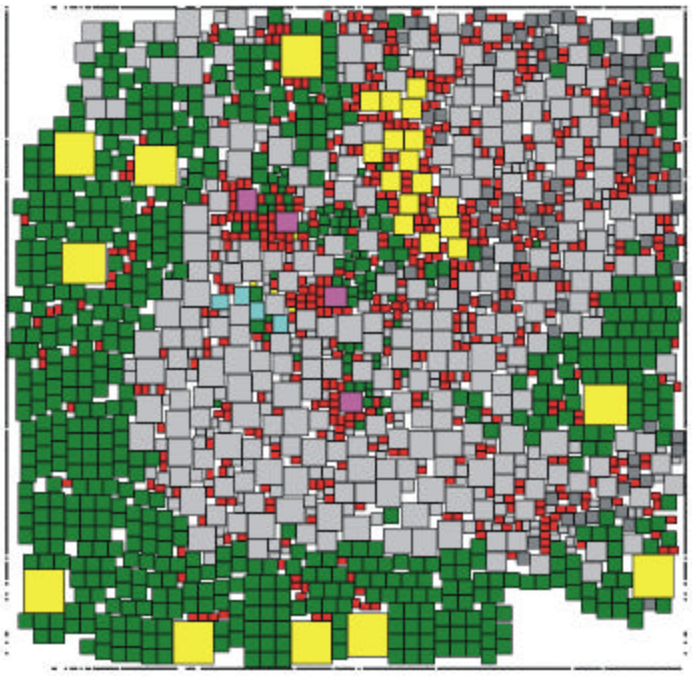
PLACEMENT COMPLETION (35%)
WIRELENGTH COST: 200,355
TILE SIZE: 338X308 -- BETWEEN RE-HEAT ANNEALS



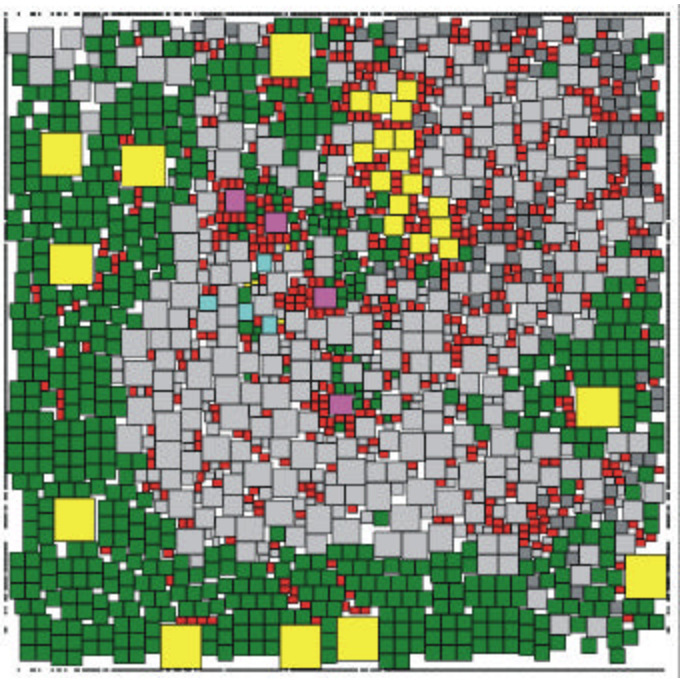
PLACEMENT COMPLETION (42.5%)
WIRELENGTH COST: 188,367
TILE SIZE: 305X279 -- BETWEEN RE-HEAT ANNEALS



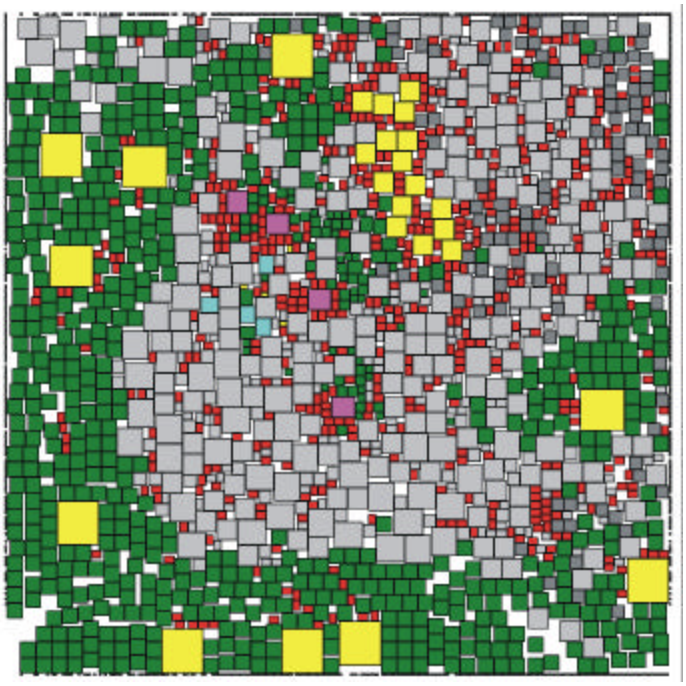
PLACEMENT COMPLETION (45%)
WIRELENGTH COST: 181,254
TILE SIZE: 285X268 -- BETWEEN RE-HEAT ANNEALS



Placement Completion (50%)
Wirelength Cost: 176,396
Tile Size: 265x258 -- Between Re-heat Anneals

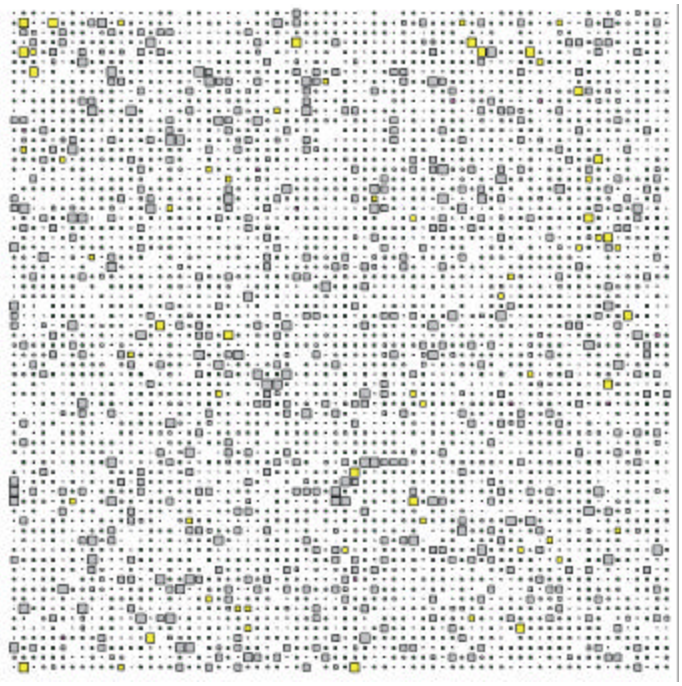


Placement Completion (77.5%)
Wirelength Cost: 167,088
Tile Size: 256x254 -- Between Re-heat Anneals

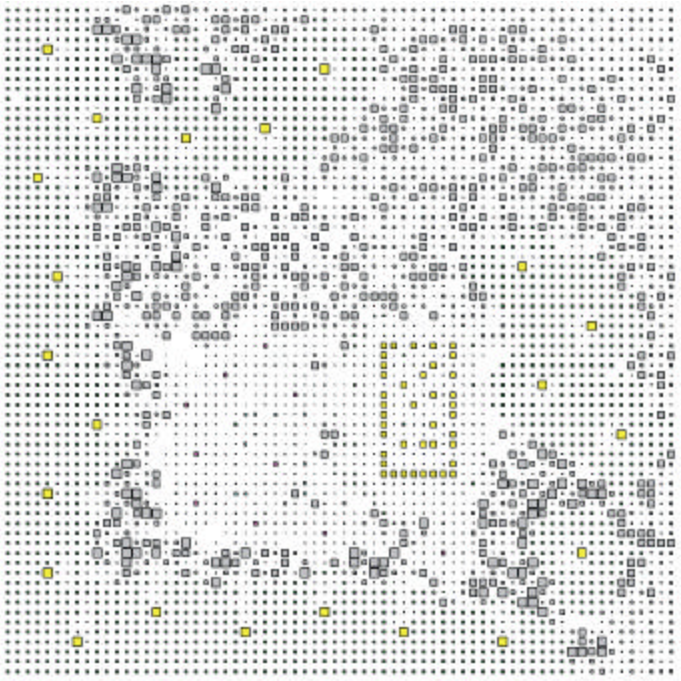


PLACEMENT COMPLETION (100%)
WIRELENGTH COST: 159,542
TILE SIZE: 255X254 -- FINAL LAYOUT

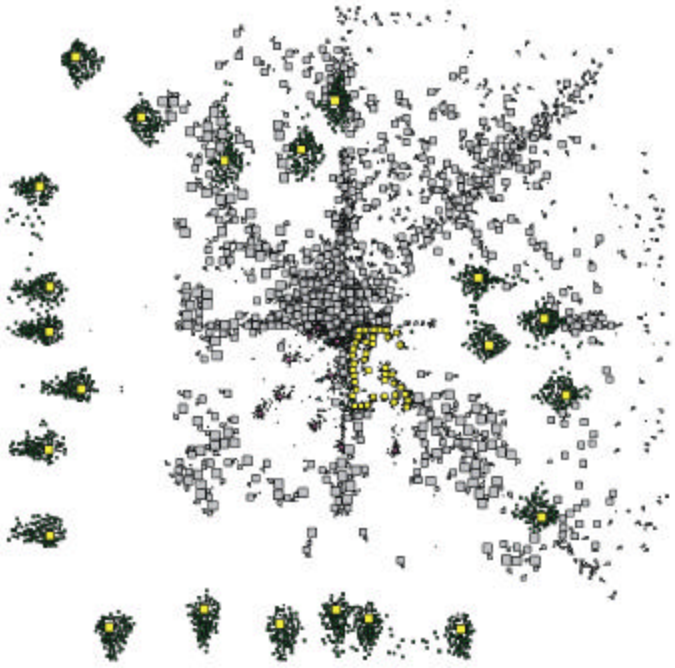
The second layout run illustrates the interim layouts generated by the ATL tool at various stages during optimization of the 10-LUT architecture used to generate the experimental results in 4.6.



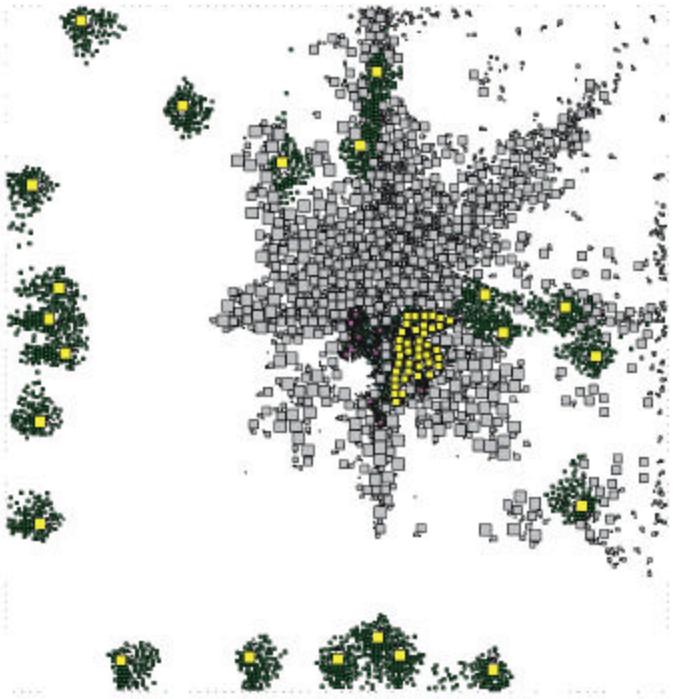
Placement Time (0%)
Wirelength Cost: 6,689,960
Tile Size: 1294x1294 -- Initial Anneal (Large-Grid Placement)



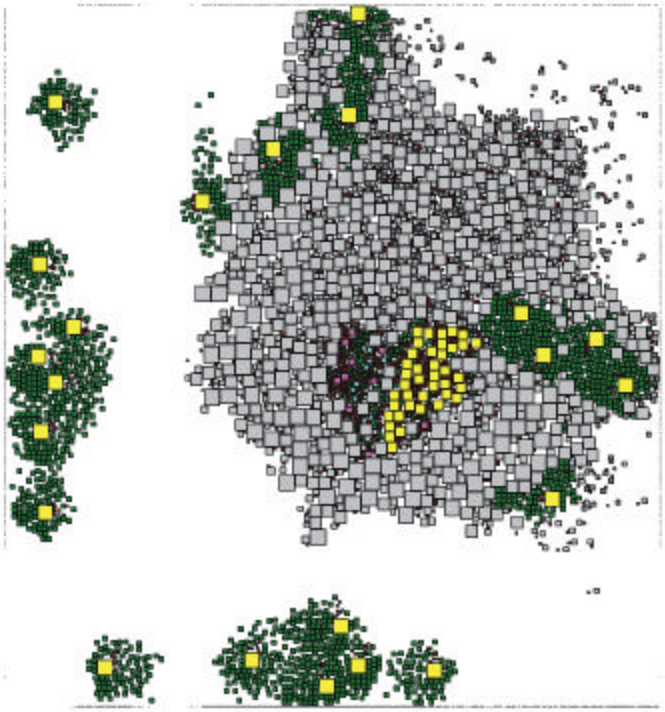
Placement Completion (15%)
Wirelength Cost: 1,491,480
Tile Size: 1285x1286 -- After Initial Anneal (Large-Grid Placement)



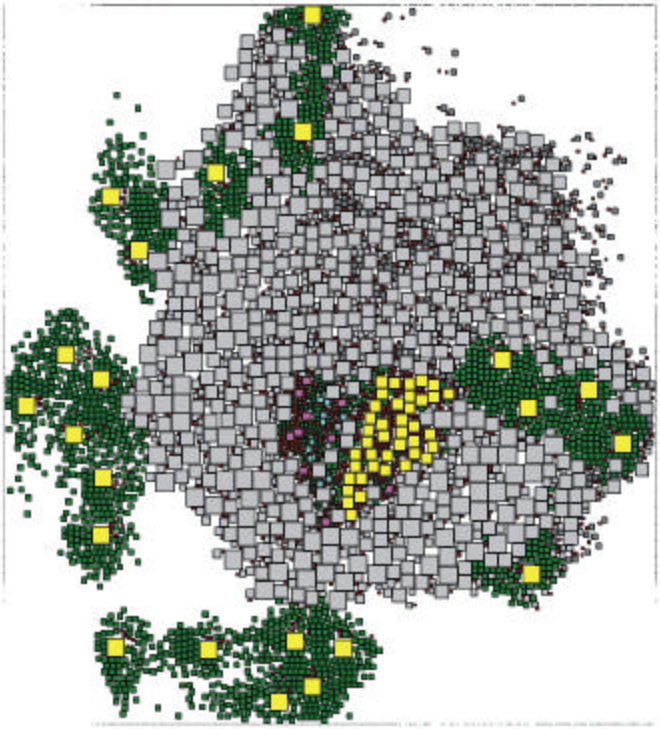
PLACEMENT COMPLETION (16%)
WIRELENGTH COST: 1,393,750
TILE SIZE: 1265X1247 -- BETWEEN RE-HEAT ANNEALS



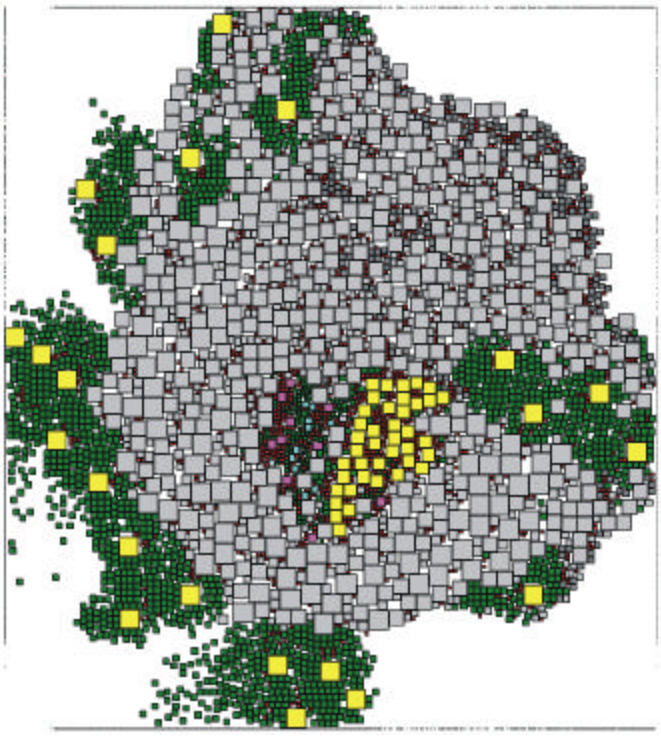
PLACEMENT COMPLETION (17%)
WIRELENGTH COST: 1,386,340
TILE SIZE: 1015X1056 -- BETWEEN RE-HEAT ANNEALS



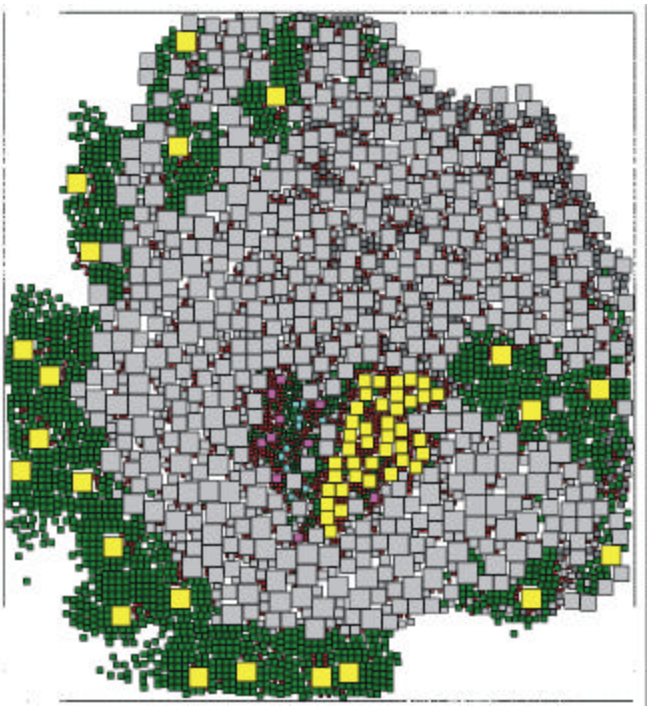
PLACEMENT COMPLETION (19%)
WIRELENGTH COST: 908,144
TILE SIZE: 753X804 -- BETWEEN RE-HEAT ANNEALS



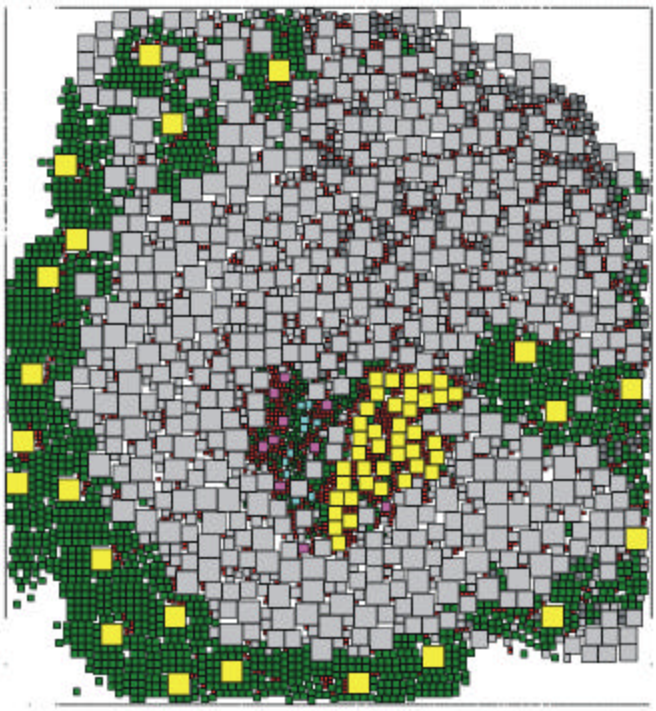
PLACEMENT COMPLETION (21%)
WIRELENGTH COST: 835,789
TILE SIZE: 648X715 -- BETWEEN RE-HEAT ANNEALS



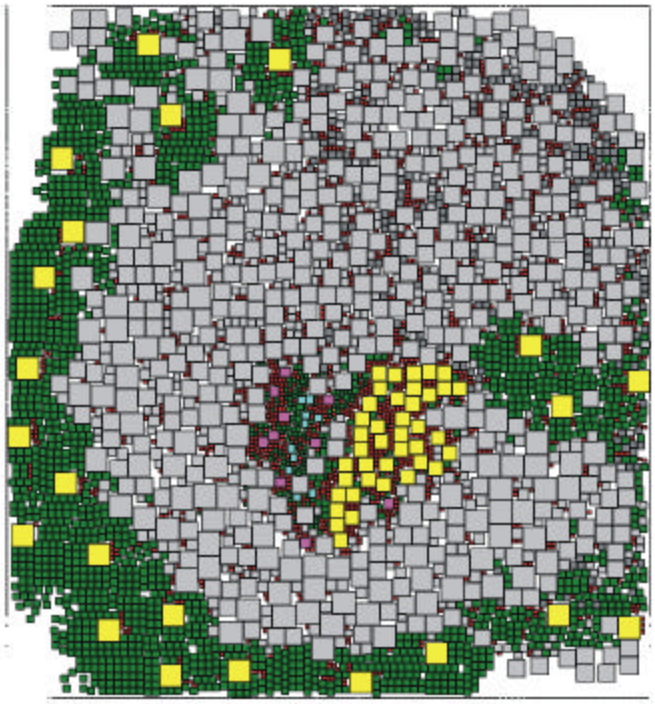
PLACEMENT COMPLETION (23%)
WIRELENGTH COST: 828,060
TILE SIZE: 592X656 -- BETWEEN RE-HEAT ANNEALS



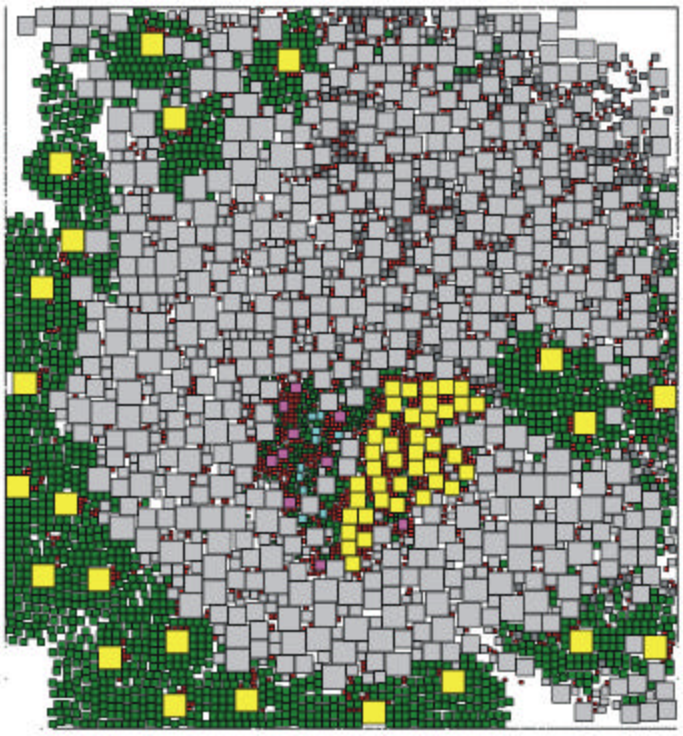
PLACEMENT COMPLETION (27%)
WIRELENGTH COST: 729,575
TILE SIZE: 554X604 -- BETWEEN RE-HEAT ANNEALS



PLACEMENT COMPLETION (54%)
WIRELENGTH COST: 670,399
TILE SIZE: 521X561 -- BETWEEN RE-HEAT ANNEALS



PLACEMENT COMPLETION (79%)
WIRELENGTH COST: 653,537
TILE SIZE: 509X548 -- BETWEEN RE-HEAT ANNEALS



PLACEMENT COMPLETION (100%)
WIRELENGTH COST: 629,134
TILE SIZE: 507X545 -- FINAL LAYOUT

REFERENCES

[1] Padalia, K.. Automatic Transistor-Level Design and Layout Placement of FPGA Logic and Routing from an Architectural Specification. Bachelor's Thesis. University of Toronto, 2001.

[2] Betz, Vaughn, Marquardt, Alexander, and Rose, Jonathan. Architecture and CAD for Deep-Submicron FPGAs. Kluwer Academic Publishers: Boston, 1999.

[3] Betz, Vaughn. Architecture and CAD for Speed and Area Optimization of FPGAs. Ph.D. Thesis. University of Toronto, 1998.

[4] Cheng, C.. "RISA: Accurate and Efficient Placement and Routing Modeling", ICCAD, 1994. pp. 690-695.

[5] Fogel, D. B. and Michalewicz Z.. How to Solve It: Modern Heuristics. Springer: Germany, 2000.

[6] Swartz, W.P.. Automatic Layout of Analog and Digital Mixed Macro/Standard Cell Integrated Circuits. Ph.D. Thesis. Yale University, 1993.

[7] Gerez, Sabih H.. Algorithms for VLSI Design Automation. John Wiley & Sons: New York, 1999.

[8] Ahmed, E.. The Effect of Logic Block Granularity on Deep-Submicron FPGA Performance and Density. M.A.Sc. Thesis. University of Toronto, 2001.

[9] Najm, F.. VLSI Systems. Course Notes, University of Toronto, 2002.

[10] Xilinx Inc.. Virtex-E Extended Memory: Detailed Functional Description. Revision 2.0. http://www.xilinx.com/partinfo/databook.htm. November 16, 2001.

[11] Xilinx Inc.. Virtex-II: Introduction and Ordering Information. Revision 1.7. http://www.xilinx.com/partinfo/databook.htm. October 2, 2001.

[12] Leventis, P.. Placement Algorithms and Routing Architecture for Long-Line Based FPGAs. Bachelor's Thesis. University of Toronto, 1999.