

To the Graduate Council

I am submitting herewith a thesis written by Larry Cunningham entitled "Selective Sequential Trunk Locking of Single-Trunk Steiner Trees in Global Routing" I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science

DP Mehta

Dinesh Mehta, Associate Professor

We have read this thesis
and recommend its acceptance

Bruce Whitely

Bruce Boman

Accepted for the Council

LeWominked

Associate Vice Chancellor and
Dean of The Graduate School

Selective Sequential Trunk Locking of Single-Trunk Steiner Trees in Global Routing

A Thesis

Presented for the

Master of Science Degree

The University of Tennessee, Knoxville

Larry Paul Cunningham

May 2000

Dedication

To Carl for reminding me that I did not let him stop on his thesis. To DJ for encouraging me even when I did not want to be encouraged. And especially to my parents for being there when I needed them.

Acknowledgements

I would like to thank my advisor, Dr Dinesh Mehta, for his guidance and support in the development of this thesis. I would also like to thank Dr. Bruce Whitehead for his assistance in the coding of the program this thesis is based on. Finally, I would like to thank Dr Bruce Bomar for serving on my committee

Abstract

This thesis presents a new algorithm for the global routing of M3-layer single-trunk Steiner trees. The new algorithm, called selective sequential trunk locking (SSTL), works by selecting a trunk from a set of trunks, then locking the trunk in place. The remainder of the trunks in the set are adjusted according to their relationships to the locked trunk. An input file provides the selection criteria along with the chip and circuit layout information. SSTL is compared to a divide and conquer algorithm, which works by recursively dividing the area the trunks exist in then moving trunks from more highly congested areas to less highly congested areas. When the size of an area is at or below a threshold size, an attempt is made to lock down all trunks in the area. The experimental results show that, although the divide and conquer algorithm has some advantages over SSTL, SSTL wins in the most important area. SSTL will lock down all trunks in the chip under a wider variety of conditions than divide and conquer.

Contents

1 Introduction	1
2 Literature Survey and Analysis	11
2.1 Divide and Conquer	13
2.2 Selective Sequential Trunk Locking (SSTL)	13
2.3 Other Global Routers	14
3 Algorithm	15
3.1 The Input File	15
3.2 Divide and Conquer	17
3.3 Selective Sequential Trunk Locking (SSTL)	18
3.4 Comparison of SSTL and Divide and Conquer	26
4 Experimental Results	27
4.1 Baselining SSTL	27
4.2 The Comparison Runs	30

4.3 The Timing Runs	30
4.4 Other Criteria	37
5 Conclusion	39
5.1 Analysis of Experimental Results	39
5.2 Analysis of the Variations in Wire Length Change in SSTL	39
Bibliography	42
Appemdicees	45
A Split Trunks	46
A.1 Connecting a Split Trunk	46
A.1.1 Connector Effect on Odd Median	46
A.1.1.1 Connector Inside Next Pin	48
A.1.1.2 Connector Outside Next Pin	48
A.1.2 Connector Effect on Even Medians	50
A.1.2.1 Connector Inside Range	50
A.1.2.2 Connector Outside Range	52
A.2 The Interaction of Split Trunks	52
A.2.1 Odd-Odd Interaction	52
A.2.1.1 Both Connectors Inside Next Pin	54
A.2.1.2 Both Connectors Outside Next Pin	54
A.2.1.3 One Inside and One Outside	57

A.2.1.3.1	Next Pin In Same Track	57
A.2.1.3.2	Next Pin Not In Same Track	61
A.2.1.4	Odd-Odd Interaction Summary	61
A.2.2	Even-Even Interaction	65
A.2.2.1	Ranges Do Not Touch	65
A.2.2.2	Ranges Touch	65
A.2.3	Even-Odd Interaction	67
A.2.3.1	Medians That Touch	67
A.2.3.2	Medians That Do Not Touch	70
Vita		75

List of Tables

4 1	Baseline criteria runs	29
4 2	Determining the best baseline criteria combination	31
4 3	Comparison runs for input file 1 (529 nets)	32
4.4	Comparison runs for input file 2 (529 nets)	33
4.5	Comparison runs for input file 3 (604 nets)	34
4.6	Comparison runs for input file 4 (604 nets)	35
4.7	User times from the timex command	36
5 1	Baseline criteria runs on the abbreviated input file with tw set to 144	40

List of Figures

1. 1	Example of a stadard-cell layout showing routing through layers M1 and M2	3
1 2	Example of a stadard-cell layout showing routing through layers M1 – M3	5
1 3	Examples of single trunk Steiner trees	6
1 4	Example of odd median indicating wire length changes due to movement	8
1 5	Examples of even medians indicating wire length changes due to movement	10
3. 1	Example of an input file	16
3 2	Example of scratch range adjustment in the divide and conquer algorithm	19
3 3	SSTL algorithm	21
3 4	Beginning trunk positions for the SSTL example	23
3. 5	Net 2 trunk movements	25
3 6	Net 4 trunk movements	25
3 7	Final trunk positions	25
A. 1	Example of a split trunk	47
A 2	Examples of the Connector Inside Next Pin situation	49
A 3	Examples of the Connector Outside Next Pin situation	51
A 4	Examples of the Connector Inside Range situation	51
A 5	Examples of the Connector Outside Range situation	53
A 6	Example of the double Connector Inside Next Pin situation	55
A 7	Examples of the double Connector Outside Next Pin situation	58
A 8	Example of the Connector Inside-Outside Next Pin situation where the next pin and the median pin are in the same track	62
A 9	Example of the Connector Inside-Outside Next Pin situation where the next pin and the median pin are not in the same track	63
A 10	Examples of two even trunks that do not touch	66

A.11	Examples of two even medians that touch	68
A.12	Example of even-odd medians that touch	71
A.13	Example of even-odd medians that do not touch	73

Chapter 1

Introduction

The first computers cost hundreds of thousands of dollars and filled entire rooms. Today, a calculator the size of a credit card can be bought for a dollar. This miniaturization of computer components came about because of physics and economics. Because electricity moves through a particular wire material at a constant speed, one way to make a component operate faster is to shorten the wires used to connect the circuits inside the component. This has the domino effect of making components built using these components smaller as well. A side effect of this operation is that the components require less material, which makes the components cheaper to produce on the same machinery. Presently, manufacturers are placing millions of components on chips a few square inches in area. In order to pack all of these components into such a small space, the components and their connections must be efficiently placed.

In standard-cell layouts, all components are the same height. This allows the components to be placed in rows with an even amount of space between the rows. Pins on each component allow the component to be connected to other components with wires routed through the spaces between the rows. All of the pins to be connected to each other and the wires connecting them are considered to be a net, and the extreme vertical and horizontal pin locations in the net are considered to be the edges of its netbox.

The wires are routed through and on layers of non-conductive material attached to the surface the components are mounted on. All layers are of equal area, with the edges determined by the extreme horizontal and vertical locations of the components' pins. Many algorithms, though not

all, consider each layer to be a grid with vertical tracks and horizontal rows. The track and the row closest to a pin's location determine the coordinates of the pin. Likewise, wires are routed from one track-row coordinate to another.

Each layer has a specific function. The vertical connection wires internal to each component and for adjacent rows occupy the layer closest to the components, called M1. The horizontal connection wires internal to each component and for adjacent rows occupy the next layer out, called M2. The areas of the M1 and M2 layers immediately "over" each component are reserved exclusively for the internal connections of that component.

Many, but not all, algorithms run the net connections only through M1 and M2. This is not a problem when the connections are between components in adjacent rows. However, connections between components in non-adjacent rows run into problems because of the reserved areas in M1 and M2. There are two ways to deal with this problem: feedthroughs or a third layer.

Feedthroughs are the spaces between components in the rows on layer M1. Because these spaces are not reserved for internal component connections, they are free to provide a path for wires passing through this row to an adjacent row. While this provides a means of making "long" connections, it has its own problems. The wire passing through may not be located on the same track as the feedthrough, requiring an M2 wire to get it to or from the feedthrough. Wires passing through multiple rows could become very convoluted. Also, if there are more wires requiring feedthroughs than spaces in a row, additional space must be used to satisfy the requirement. This adds to the size of the grid, the layer, and the chip. Power and clock connections are routed through the rest of the layers from M3 up. Refer to Figure 1.1 for an example of feedthrough routing through M1 and M2.

Other algorithms, including the two in this thesis, have a third layer, called M3. The vertical wires connecting pins in non-adjacent rows are run through M3. M3 greatly decreases the complexity of routing. M3 also decreases the length of the wires connecting components in non-

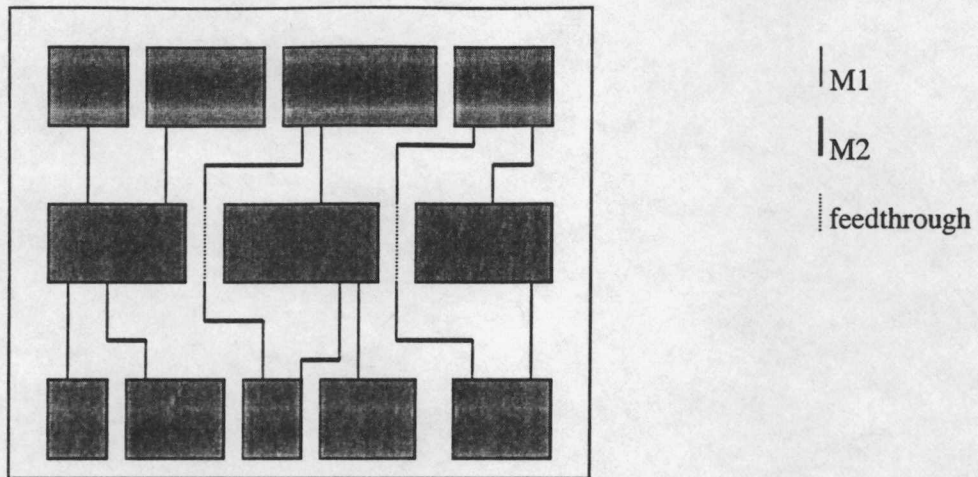


Figure 1.1 Example of a standard-cell layout showing routing through layers M1 and M2. The areas of M1 and M2 reserved for internal component connections are shaded. The feedthroughs are indicated with dotted lines.

adjacent rows while decreasing the clutter in the M1 and M2 layers. Power and clock connections are routed through the rest of the layers from M4 up. Refer to Figure 1.2 for an example of M3 layer routing.

A simple net connection method is the Steiner tree. The connection wires in a Steiner tree can connect pins either directly to other pins or to the connection wires of other pins in the net. The algorithms in this thesis use a variation of this called the single-trunk Steiner tree. All of the pins in a single-trunk Steiner tree are connected to a single vertical trunk wire in the M3 layer. This method optimizes the total vertical wire length on the M3 layer. The upper and lower ends of a trunk are located at the upper and lower edges of the netbox. The pins in the net are connected to the trunk with wires run through the M2 layer. Refer to Figure 1.3 for examples of single-trunk Steiner trees. The possibility of splitting the trunk of a single-trunk Steiner tree to increase its usefulness is presented in Appendix A. The research was undertaken, but was not implemented.

Global routing attempts to place the trunks of all of the nets in the grid so that the wires connecting each net are as short as possible. Since the pin locations determine the length of the trunks, there is no way to minimize the length of the trunks without moving the pins. Because pin movement is outside the scope of global routing, the wire length of the net is minimized by minimizing the lengths of the horizontal M2 wires connecting the pins to the trunk.

Every single-trunk Steiner tree has one or more trunk locations where the net's overall wire length is at its absolute minimum for the net's pin locations. The range of tracks in which this occurs is called the median. Mathematically, the median of any set of pins sorted by track number is the point in the set at which there are an equal number of pins to the left and to the right of the median. The median of any set of sorted pins will fit one of two basic situations: odd medians or even medians.

Odd median sets contain an odd number of pins sorted in ascending order by their x locations. The number of pins will be represented by the formula $2n+1$ (where $n>0$) and the pins themselves

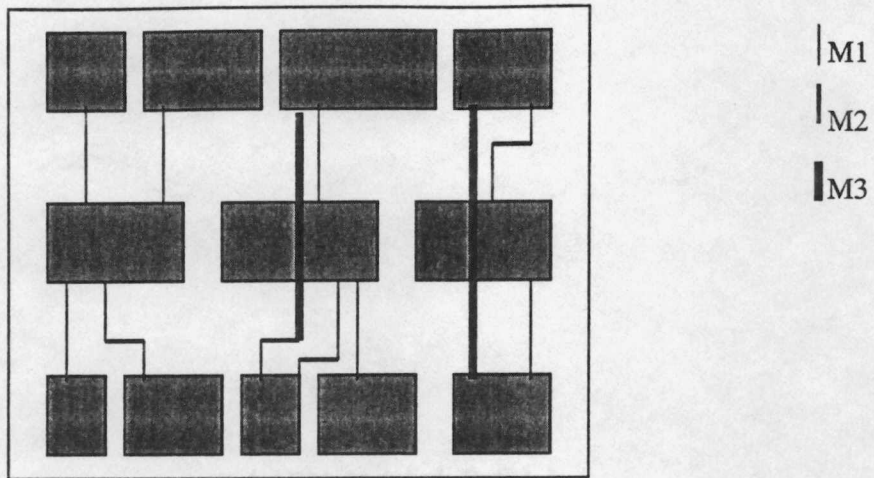


Figure 1.2 Example of a standard-cell layout showing routing through layers M1 – M3. The areas of M1 and M2 reserved for internal component connections are shaded.

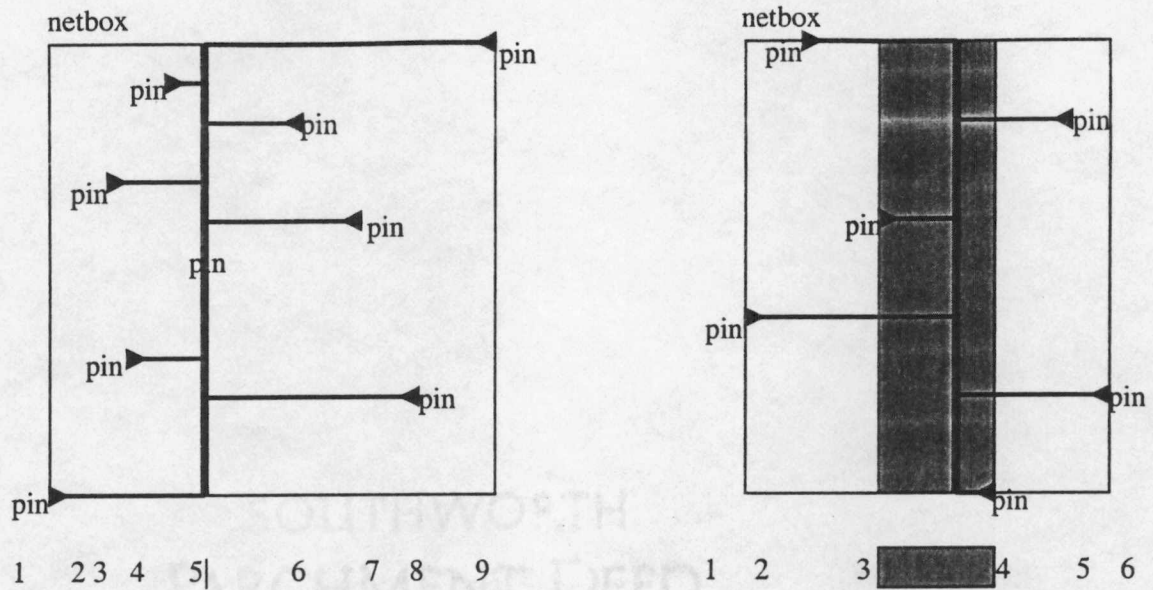


Figure 1.3 Examples of single trunk Steiner trees. A representation of the tree is shown below it. An odd median is represented by "|"; an even median is represented by "■" (medians are explained below).

will be numbered 1 through $2n+1$. Because the median requires an equal number of pins on each side, the median is located at pin $n+1$, with the n pins 1 through n on its left and with the n pins $n+2$ through $2n+1$ on its right. Because $2n/2 = n$, it follows that $(2n+1)/2 = n + .5$. Since the median has been shown to be at pin $n+1$, either $.5$ must be added or the result must be rounded up. Because rounding is not always consistent from system to system, the floating point equation $((2n+1)/2) + .5$ assigned to an integer variable should find an odd median on any system.

The proof of odd medians involves what happens when the trunk moves away from the median. The trunk increases the wire length between itself and at least $n+1$ pins (n pins on the other side of the median plus at least one pin on the median). At the same time, the trunk decreases the wire length between itself and at most n pins (n pins on this side of the median minus any pins from this side of the median located in the same track as the median). This means that the wire length will increase at least 1 unit whichever direction the trunk moves ($n+1-n = 1$). Since we are looking for the trunk location that minimizes the wire length and the trunk cannot move from the median pin without increasing the wire length, the shortest wire length is found using a trunk located on the median pin (pin $n+1$). Refer to Figure 1.4 for an example of an odd median and for graphic proof of the wire length change that results from moving the trunk outside the median.

Even median sets contain an even number of pins sorted in ascending order by their x locations. The number of pins will be represented by the formula $2n$ (where $n > 0$) and the pins themselves will be numbered 1 through $2n$. Even median sets are more complicated to deal with than odd median sets because there is no way to split an even median set so that there is one pin in the middle with an equal number of pins on each side of it. The only way to have an equal number of pins on each side of a central median is to locate the median inside the range between two central pins. Because $2n/2 = n$, the median will require pins n and $n+1$, with the $n-1$ pins 1 through $n-1$ on the left and the $n-1$ pins $n+2$ through $2n$ on the right.

$$+n \quad +1 \quad -n \quad = +1$$

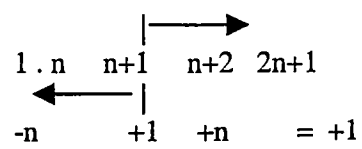
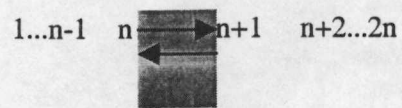


Figure 1.4 Example of odd median indicating wire length changes due to movement

As with the odd median, the proof for an even median involves what happens when the trunk moves. When the trunk moves inside the space between median pins n and $n+1$, it increases the wire length between itself and n pins ($n-1$ pins on the other side of the median range plus that boundary pin). At the same time the trunk decreases the wire length between itself and n pins ($n-1$ pins on this side of the median range plus this boundary pin). This means that the wire length will stay the same as long as the trunk only moves inside the range bounded by the median pins n and $n+1$. If the trunk moves outside the median range, at least one pin (the median boundary pin and any pins located in the same track) will go from decreasing the wire length between itself and the trunk to increasing the distance. Now more pins are increasing their wire lengths than decreasing, so the overall wire length will increase. Since we are looking for the trunk location that minimizes the wire length and the trunk cannot move outside the median range without increasing the wire length, the shortest wire length is found using a trunk inside the median range. Please note that if pins n and $n+1$ are in the same track, the median range is a single track. Refer to Figure 1.5 for an example of an even median and for graphic proof of the wire length change that results from moving the trunk outside the median.

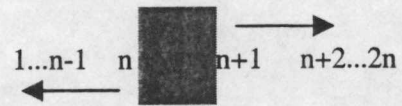
The overall wire length of any net is minimized when the trunk is located in the median. When the trunk moves outside of the net's median, the overall wire length increases. The amount of this increase is referred to as the wire length change. Wire length change can be measured individually for each net or cumulatively for all nets in the chip.

$$+(n-1) \quad +1 \quad -1 \quad -(n-1) \quad = 0$$



$$-(n-1) \quad -1 \quad +1 \quad +(n-1) \quad = 0$$

$$+(n-1) \quad +1 \quad +1 \quad -(n-1) \quad = +2$$



$$-(n-1) \quad +1 \quad +1 \quad +(n-1) \quad = +2$$

Figure 1.5 Examples of even medians indicating wire length changes due to movement

Chapter 2

Literature Survey and Analysis

A literature search did not find much material covering global routing with the exception of examples of the various global routing algorithms in various textbooks.

One global routing algorithm is simulated annealing [5][6][7]. Simulated annealing uses a 2-stage process to connect the pins in a net. The first stage attempts to minimize the average number of wires running through the spaces between the standard-cell rows when assigning pin connection wires. At the same time, the switchable pin connection wires that can be assigned to higher or lower between-row spaces are identified. The second stage attempts to minimize the wire lengths by assigning the switchable connection wires to different between-row spaces, then seeing if that decreases the wire length. This “what-if” process can be very time consuming.

Another global routing algorithm is integer linear programming [5][6][7]. Integer linear programming represents the grid as a grid-graph, where each node is a grid cell. The number of tracks that can pass between two adjacent nodes is considered the weight of the link between the nodes. All possible Steiner trees that connect all pins in each net are found. The wire length of a particular Steiner tree is the number of adjacent nodes that are traversed by the connection wires in the Steiner tree. The Steiner tree with the lowest wire length for a particular net is chosen. Attention must be paid to the number of available connections between the nodes. This “what-if” process can be very time consuming.

Another global routing algorithm is maze routing [5][6][7]. Maze routing connects two points in the grid by finding a sequence of adjacent grid cells in a path between the two. One pin is

labeled as the source, and the other pin is labeled as the target. The algorithm determines the shortest path from the source pin to the target pin by minimizing the overall length of the connection wire or by minimizing the length of each straight wire segment in the connection wire. When maze routing is extended to handle multiple nets or nets with more than two pins, any routed path is considered an obstacle by any path routed after it.

Another global routing algorithm is the line search [5][6][7]. The line search algorithm uses less memory than maze routing by representing the layer and the connection wires as line segments instead of as a grid. Vertical and horizontal lines, called trial lines, are extended from the source and target pins to whatever stops them, such as the edge of the layer or the edge of a component. If one of the source pin trial lines intersects one of the target pin trial lines, that is a path between the two pins. If none of the trial lines intersect, new trial lines perpendicular to the previous set of trial lines are extended, again until they are stopped. The new trial lines are checked for an intersection. This process continues until a source pin trial line intersects with a target pin trial line or until no more lines can be extended. The line search algorithm will find a path if one exists but does not guarantee the path will be the shortest one. When line search routing is extended to handle multiple nets or nets with more than two pins, any routed path is considered an obstacle by any path routed after it.

Another global routing algorithm is area routing [8]. Area routing begins by dividing the layers into regions, usually with one component per region. Unlike in standard-cell layouts, the regions are not required to be the same size or the same shape. Area routing then adds representations of the possible connections within and between the regions. Each net is then routed according to its relationship to the possible connection representations. Area routing does not use standard-cell format.

The two algorithms presented in this thesis adapt basic computer algorithm types (the divide and conquer type [1] and the greedy type [1]) for global routing. Both algorithms use the

standard-cell layout format, in which all components are of the same height [5]. Both algorithms use the row-based approach, which determines the location of components, wires, and pins by overlaying each layer with a grid [5]. Each cell of the grid is assumed to contain the area the wire passes through and buffer areas separating this wire from the wires in adjacent cells. Both algorithms connect the pins in nets using single-trunk Steiner trees with the trunk located in the M3 layer [5]. Finally, because another router will handle routing the pin to trunk connection wires, both algorithms consider the actual upper and lower ends of the trunk to be one row inside the edges of the netbox.

2.1 Divide and Conquer

The divide and conquer program used in this thesis recursively splits the grid in half, then processes each half. This is consistent with the definition of a divide and conquer type algorithm [1]. The general purpose of the splitting is to reduce the overall problem (the grid) into smaller subproblems (the grid sections) that can be easily solved. Because the nets are assigned to one half or to the other when a grid section splits, no net is assigned to more than one grid section at a time. This ensures that each grid section can be solved independently of the others. After all grid sections have been solved, they are reassembled back into the grid according to the track positions in each grid section. While the subproblems in this divide and conquer algorithm are of the same type as the whole problem, this is not required.

2.2 Selective Sequential Trunk Locking (SSTL)

The SSTL algorithm fits the definition of a greedy algorithm [1]. Using a specific user-chosen set of selection criteria, SSTL selects one trunk out of a set of candidate trunks, then locks it down. SSTL then adjusts the positions of the other trunks according to their relationships to the locked trunk. Finally, the locked trunk and all other existing trunks from the same net are

removed from the set of candidate trunks. This continues until the set of candidate trunks is empty. The final result of the run represents the optimal solution for that set of selection criteria for that set of candidate trunks. Each locked trunk is considered to be a member of the optimal solution

2.3 Other Global Routers

While there has been much research on the subject of global routing, there seems to be little that matches completely the basics of the two algorithms used in this thesis. Several of the routers are row-based [2][3], like SSTL. However, one [8] is not. Several of the routers use standard cells [2][3][4], as do both of the algorithms used here. However, those same routers [2][3][4] all use feedthroughs, indicating that they do not route wires through the M3 layer. The gridless area router [8] claims to be able to handle any number of layers. While the algorithms are all based on a divide and conquer style breakdown of the global routing process, the algorithms used to solve many of the steps are greedy [2][3][4][8]

Chapter 3

Algorithm

3.1 The Input File

For consistency, the two algorithms use the same input file, however, the two algorithms interpret some of the input in ways specific to them. All parameters for the grid and for the nets are either read from the input file or are derived from the information in it. Refer to Figure 3.1 for an example input file.

The first line of the input file contains basic grid and net parameters. The first four parameters define the lower, left, upper, and right edges of the grid. These parameters define the limits of the area the nets are placed in. The fifth parameter defines the width of the tracks the trunks are placed in. Dividing the total grid width by the track width determines the number of tracks available to place trunks in. By changing the track width, the number of tracks in the grid can be increased or decreased. The sixth parameter specifies the number of nets.

The second line of the input file contains the selection criteria SSTL uses to choose which trunk to lock down. The divide and conquer algorithm ignores these parameters. The first parameter in the line specifies the number of criteria. SSTL reads that number of criteria choices into an array for later use.

After the basics are read, the parameters for the nets themselves are read, one per line. The first parameter is interpreted in different ways by the two algorithms. The divide and conquer algorithm reads the first parameter as the net's aggression percentage. SSTL reads the

```
0 0 95064 22896 100 529
3 1 -3 -2
0 2 720 17 1944 2
0 2 720 29 2664 2
. .
..
```

Figure 3.1 Example of an input file

first parameter as the net's priority. The second parameter corresponds to the number of pins in the net. These two parameters along with the net number and the track width are sent into the net container to build the net object for the net. The pin locations are listed after these first two parameters, first the x location, then the row number.

The net container reads and stores the pin locations, then calculates several parameters common to the two algorithms. These include the top and bottom rows of the trunk, the median range, and the wire length that results if the trunk is locked down inside the median range. Each algorithm then calculates its own particular parameters. If the net has an odd median, SSTL sets one initial trunk location to that track number. If the net has an even median, SSTL sets two initial trunk locations at the leftmost and rightmost track numbers of the median range. The divide and conquer algorithm uses the aggression percentage and the median range to calculate the leftmost and rightmost tracks the trunk is allowed to move between. This movement allowance is called the scratch range. At zero percent aggression, the scratch range is restricted to the tracks inside or immediately surrounding the range between the left and right edges of the netbox. At one hundred percent aggression, the scratch range is restricted to the tracks inside or immediately surrounding the net's median range.

3.2 Divide and Conquer

The divide and conquer algorithm begins by dividing the grid in half. When the grid is divided, the scratch range of each net in the grid falls into one of three areas: entirely in the left half, entirely in the right half, or straddling the dividing line. The algorithm then uses the following process to assign the straddlers into one half or the other. First, the maximum row congestion, or number of pin to trunk wires in any row, is calculated for the nets lying entirely in one half or the other. Next, the straddlers are sorted by the means of their median ranges. Finally, straddlers are assigned to one of the halves according to the maximum row congestion.

values for each half and according to the scratch ranges of the nets. When a straddler is assigned to a half, its scratch range is adjusted to stop at the dividing line. Refer to Figure 3.2 for an example of the assignments and the scratch range adjustment. After all nets have been assigned to one half or the other, the halves are themselves divided in half and the process is repeated. This continues until the halves contain at most a threshold number of tracks. When a half reaches this threshold, the algorithm attempts to lock down all trunks in the half. If a trunk is not locked, it is given an arbitrary location.

3.3 Selective Sequential Trunk Locking (SSTL)

SSTL uses the infringement between trunks to determine the movement of trunks and can use it to determine which trunk to lock down. Infringement can occur in two forms: direct and indirect. Direct infringement occurs when two trunks are located in the same track and compete with each other for one or more grid cells in the track. Indirect infringement occurs when two trunks would directly infringe if they were in the same track. Each net contains a list of nets that indirectly infringe with it.

Each net determines the best location to lock down its trunk by moving 4 temporary trunks around the grid. Trunk 0 can only move to the left. For a net with an odd median, trunk 0 is initially set to the track number of the odd median pin and is the only trunk initially set. For a net with an even median, trunk 0 is initially set to the track number of the left boundary pin of the median range. Trunks 1 and 2 only exist inside the median range of a net with an even median. Trunk 1 is initially set to the next available track to the right of trunk 0 and can only move to the right. Trunk 1 is created when trunk 0 splits while moving for the first time. Trunk 2 is initially set to the next available track to the left of trunk 3 and can only move left. Trunk 2 is created when trunk 3 splits while moving for the first time. Trunk 3 can only move to the right, but is created by one of two mutually exclusive situations. If a net has an even median, trunk 3 is

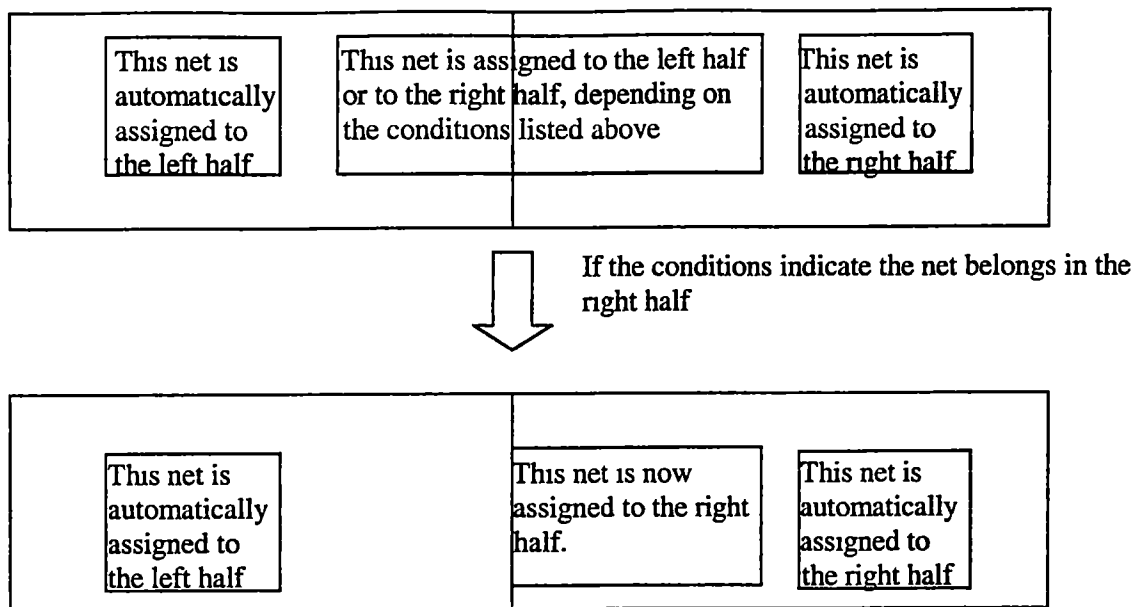


Figure 3.2 Example of scratch range adjustment in the divide and conquer algorithm

initially set to the track number of the right boundary pin of the even median range. If a net has an odd median, trunk 3 is not initially set, but is created when trunk 0 splits while moving for the first time. When a trunk is locked down, the other three temporary trunks are removed from the grid.

SSTL uses a set of criteria to determine which trunk to lock down. The criteria numbers read from the input file determine what characteristics are used, in what order they are used, and whether a particular criterion is to look for the minimum or maximum value. Each criterion uses the same process. A list of all possible candidate trunks is supplied to the criterion. The criterion reads through the list, determining if the value of a particular characteristic is the maximum or minimum value for that characteristic among the candidate trunks. Of course, if a candidate trunk has not been initially set yet or has been removed from the grid, it is not allowed to compete. The criterion returns a list of the trunks that have the maximum or minimum value of that characteristic. The list returned from one criterion can be used as the input list for another criterion.

Figure 3.3 gives an overview of the SSTL algorithm. Once the nets are read in, the algorithm begins the routing process. A list of all trunks competing for lockdown is fed into the first criterion. The list that is returned from the first criterion is sent into the second criterion, and so on. The process continues until the list of criteria is exhausted or there is only 1 trunk left in the returned list. If the number of trunks left is not 1, the algorithm stops locking down trunks and outputs information on all nets.

If the number of candidate trunks left is 1, this is the trunk to be locked down. The M3 location of the net the trunk belongs to is set to the chosen trunk's location. The other 3 trunks are set to an invalid location, their corresponding wire length changes are reset to 0. Additionally, the net's 4 trunks are removed from the list of candidate trunks fed into the first criteria. Finally, a trace line indicating which trunk has been locked down is output to the output

```

Read grid parameters, track width, and number of nets from input file
Read selection criteria from input file
For (the number of nets)
    Read net information from input file
    Create net object for the net
    Determine which other nets this net indirectly infringes with
While (there are nets to search through)
    Send a list of candidate trunks into the first criterion
    The list returned by the first criteria is sent into the second criterion
    Continue until there is one candidate or the criteria list is exhausted
    If the list does not contain 1 candidate, break out of the while loop
    Lock the candidate trunk
    Reset the locations of the other trunks in the same net to an invalid track number
    Reset the wire length change of the other trunks in the same net to 0
    Remove the other trunks in the same net from the candidate list sent to the first criterion
    For (the trunks in the candidate list sent to the first criterion)
        If the trunk directly infringes with the locked trunk
            While (either trunk needs to move)
                Move the trunk that needs to be moved
                Calculate the cumulative wire length change if the trunk is outside the median range
                If this is the first move and if it is appropriate
                    Create the appropriate trunk
                If either trunk needs to be removed
                    Remove the trunk
                If either trunk directly infringes with a locked trunk
                    Start the loop over again
    Output the statistics and the net information

```

Figure 3.3 SSTL algorithm

file

When a trunk is locked down into a track segment, no other trunk can share that track segment. The program takes care of this situation by finding the trunks that directly infringe with the locked trunk, then moving them to adjacent tracks. When a trunk is moved, the program checks for all of the following special situations. If a trunk is being moved for the first time, the trunk may need to be split into two trunks with one moving left and one moving right. If a trunk moves into a track where it would directly infringe with a previously locked trunk, the trunk is moved again in the same direction. If a trunk moves into the same track as a barrier, its location is set to an invalid track number, removing the trunk from the grid. Trunks 0 and 3 are removed when they move past the edge of the grid. Trunks 1 and 2 are removed when they move into a track presently or previously occupied by any other trunk from their net. Finally, if a net has no surviving trunks after movement completes, the net's 4 trunks are removed from the list of candidate trunks fed into the first criterion. Please note that more than one of these special situations can occur at a time. One or more trace lines indicating what actions occurred as a result of the movement are output to the output file.

Figure 3.4 shows the beginning positions for 5 trunks. Trunk 3 is the locked trunk for net 3. Trunks 1-0 and 2-0 are the 0 trunks for the odd (or single-track even) median nets 1 and 2. Trunks 4-0 and 4-3 are the 0 and 3 trunks for even median net 4. Trunks 1-0, 2-0, and 4-0 are in the left border track of the grid, trunk 3 is locked into the next track to the right, and trunk 4-3 is in the next track to the right after that. None of the unlocked trunks has moved previous to now, so trunks 1-3, 2-3, 4-1, and 4-2 have not been created yet. No other trunks remain to be placed. All nets have the same priority.

A list of all trunks whose nets have not yet been locked down or removed is sent into the first criterion read from the example input file. The first criterion (1) equates to all trunks with the maximum priority found in the list of trunks. Because all nets have the same priority, all existing

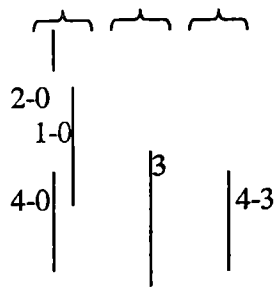


Figure 3.4 Beginning trunk positions for the SSTL example

trunks are chosen. The list of the chosen trunks is sent into the second criterion read from the example input file. The second criterion (-3) equates to all trunks located at the leftmost track position found in the list of trunks. Trunks 1-0, 2-0, and 4-0 are located at the left border of the grid and are chosen. The list of the chosen trunks is sent into the third and last criterion read from the example input file. The third criterion (-2) equates to all trunks with the lowest net number found in the list of trunks. Trunk 1-0 has the lowest net number of the trunks in the list. The list brought back from the last criterion has one candidate trunk, so trunk 1-0 is locked down.

Trunk 2-0 is an odd median that has not moved before, so trunk 2-3 is not initially set. Trunk 2-0 creates trunk 2-3. Trunk 2-0 then moves one track to the left. This moves trunk 2-0 past the left border of the grid, so trunk 2-0 is removed from the grid. Trunk 2-3 is initially set one track to the right of trunk 2-0's original location. Trunk 2-3 does not infringe with the locked trunk 3, so trunk 2-3 stays where it is. Figure 3 5 shows these movements.

Trunk 4-0 is the left boundary trunk for the median range of net 4, so trunk 4-3 is initially set. Because trunk 4 has not moved before, trunks 4-1 and 4-2 have not been initially set. Trunk 4-0 creates trunk 4-1. Trunk 4-0 moves one track to the left. This moves trunk 4-0 past the left border of the grid, so trunk 4-0 is removed from the grid. Trunk 4-1 is initially set one track to the right of trunk 4-0's original location. However, trunk 4-1 directly infringes with the locked trunk 3, so trunk 4-1 is moved another track to the right. When it arrives, it finds that it is sharing a track with trunk 4-3. Because no two trunks from the same net can be allowed to share a track, trunk 4-1 is removed from the grid. Because trunk 4-3 did not directly infringe with trunk 2-0, trunk 4-3 is unaffected. Figure 3 6 shows these movements. Figure 3 7 shows the final trunk locations.

When a trunk has been locked down and all trunk movement resulting from the lockdown has been accomplished, the process of competition, lockdown, and movement starts again. When the list of all competing trunks is exhausted, all nets have either had one of their trunks locked

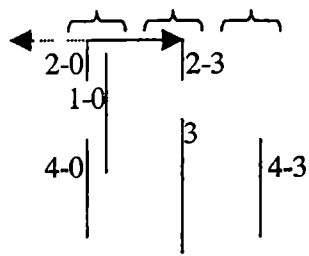


Figure 3.5 Net 2 trunk movements The dotted lines show movement that results in removal of the trunk.

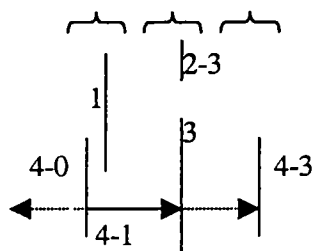


Figure 3.6 Net 4 trunk movements. The dotted lines show movement that results in removal of the trunk

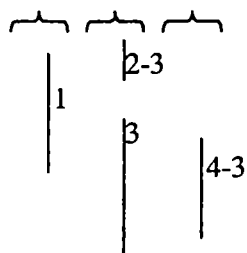


Figure 3.7 Final trunk positions

down or have been removed from competition. The process stops, and the last trace lines are output. These lines include a list of locked nets, a list of removed nets, the run statistics, and the basic information contained in the nets.

3.4 Comparison of SSTL and Divide and Conquer

SSTL removes several of the restrictions of the divide and conquer algorithm. The option of choosing the selection criteria increases the flexibility of SSTL, allowing it to home in on certain characteristics of different nets. Removing the restriction on the trunk movement allows SSTL to lock down trunks under more varied conditions. Finally, whereas divide and conquer moves trunks without regard to the number of trunks in a track, dealing with track congestion is an integral part of the SSTL algorithm.

Chapter 4

Experimental Results

This chapter presents the results from a comparison between the divide and conquer algorithm and the selective sequential locking algorithm. Both algorithms were implemented in C++, requiring 925 lines of code for the divide and conquer implementation and 3425 lines of code for the SSTL implementation. The four input files, depicting real world cases of bad component placement, were created during work my advisor did for Intel in 1997. The timing results were obtained on a Sun SPARCstation 5 running Solaris 5.6.

The trunk length of a net is set by the locations of its pins. Global routing cannot change the locations of the pins, so global routing can only minimize the lengths of the horizontal M2 wires connecting the pins to the trunks. Global routing attempts to do this by locking the trunks into the best track locations it can find for them. As shown in Chapter 1, a trunk locked into a track located inside the median range for that net is using the minimum amount of wire that will connect all pins in that single-trunk Steiner tree. A trunk locked into a track outside of the median range for that net requires additional wire to make all the connections. The additional wire required is considered to be the wire length change for the net.

4.1 Baseline SSTL

The first order of business was to baseline SSTL to determine which criteria combination was best for comparison to divide and conquer. Each criterion uses the same process. A list of all possible candidate trunks is supplied to the criterion. The criterion reads through the list,

determining if the value of a particular characteristic is the maximum or minimum value for that characteristic among the candidate trunks. Of course, if a candidate trunk has not been initially set yet or has been removed from the grid, it is not allowed to compete. The criterion returns a list of the trunks that have the maximum or minimum value of that characteristic. The list returned from one criterion can be used as the input list for another criterion.

Three criteria were chosen for the baseline. The maximum priority criterion (hp) was chosen to place the trunks of high priority nets first. Because all nets in the input files were of the same priority, the criterion was used to filter the non-existent trunks out of the candidate list. The maximum and minimum net number criteria (h₁, l₁) were chosen because they use an arbitrary net characteristic. Because the nets are numbered as they are read in, the net number is assigned in the input file. The maximum and minimum track number criteria (h_t, l_t) were chosen because they use a characteristic that can be both random and arbitrary. Each trunk is initially assigned to a track location on the boundary of that net's median range. Because the median range is determined by the net's pin locations, and because the pin locations are assigned in the input file, this initial track assignment is arbitrary. However, trunk movement during routing causes the track location to become random. Half of the combinations (h_{ph}h_t₁, h_{ph}l_t₁, h_{pl}h_t₁, h_{pl}l_t₁) selected the furthest left or the furthest right track location containing one or more existing unlocked trunks, then selected the lowest or highest numbered net with an unlocked trunk in that track location. The other half of the combinations (h_{ph}h_t, h_{ph}l_t, h_{pl}h_t, h_{pl}l_t) selected the highest or lowest numbered net containing one or more existing unlocked trunks, then selected the trunk in the net located the furthest left or the furthest right in the grid.

As shown in Table 4.1, the combinations produced a wide variety of wire length changes, but all combinations locked all trunks. The "best" combination was chosen by assigning places for each input file according to the wire length change produced by that combination. The places for each combination for the four input files were totaled up, then compared. Two combinations,

Table 4.1 Baseline criteria runs

tw = track width		h = maximum	l = minimum	wlc = wire length change				
p = priority		t = track #	i = net #	% locked = % of total trunks locked				
criteria	file 1, tw=80		file 2, tw=75		file 3, tw=110		file 4, tw=110	
	% locked	wlc	% locked	wlc	% locked	wlc	% locked	wlc
hpltli	100.00%	3498	100.00%	4786	100.00%	2283	100.00%	16489
hplilt	100.00%	2565	100 00%	6412	100.00%	2217	100.00%	17956
hphthu	100.00%	5846	100.00%	13208	100.00%	3255	100.00%	5910
hphiht	100 00%	7765	100 00%	12585	100 00%	2733	100.00%	7600
hpliht	100.00%	3535	100.00%	11836	100.00%	4529	100.00%	33012
hphtli	100 00%	4718	100 00%	13216	100.00%	4409	100.00%	30598
hplthi	100.00%	23454	100.00%	17153	100.00%	20429	100.00%	21430
hphilt	100.00%	30419	100 00%	23515	100 00%	34111	100.00%	21703

hpltl1 and hplilt, tied overall, so the tie was broken by adding up the wire length changes for the four input files, then comparing. As shown in Table 4 2, hpltl1 was the winner.

4.2 The Comparison Runs

hpltl1 and divide and conquer were run on the same input files with the track width being the only variable. The track width was incremented by 5 from 5 until both algorithms had failed to lock all trunks for at least 5 consecutive track width increments. Two numbers were deemed to be important: the percentage of the trunks locked down and the overall wire length change. Divide and conquer was run with 0% aggression (should place the most trunks) and with 100% aggression (should give the lowest wire length change). However, to avoid having to change 500+ lines in each input file, the aggression percentage for the test runs was hard-coded. The results for each input file are summarized in Tables 4 3, 4 4, 4 5, and 4 6. The divide and conquer algorithm at 100% aggression produced lower wire length changes in most cases. However, SSTL would lock down 100% of the trunks at a wider variety of track widths than divide and conquer could. In fact, Table 4 1 shows that the baseline runs of SSTL all locked down 100% of the trunks at track widths that divide and conquer could not match.

4.3 The Timing Runs

The only track widths at which all three programs would place 100% of the trunks in all cases were 5 and 10. For this reason, timings were limited to those cases. Table 4 7 summarizes the user times output when the programs were run using the timex command. The divide and conquer algorithm was at least twice as fast as SSTL in all cases.

Table 4.2 Determining the best baseline criteria combination

h = maximum l = minimum wlc = wire length change p = priority t = track # i = net #						
	file 1	file 2	file 3	file 4	total	total
	wlc place	wlc place	wlc place	wlc place	wlc place	wlc
hpltli	2	1	2	3	8	27056
hplilt	1	2	1	4	8	29150
hphthi	5	5	4	1	15	28219
hphiht	6	4	3	2	15	30683
hpliht	3	3	6	8	20	52912
hphtli	4	6	5	7	22	52941
hplthi	7	7	7	5	26	82466
hphilt	8	8	8	6	30	109748

Table 4.3 Comparison runs for input file 1 (529 nets)

hpltli = SSTL dac = divide and conquer			tw = track width 0% = 0% aggression		wlc = wire length change 100% = 100% aggression	
tw	hpltli % locked	wlc	dac 0% % locked	wlc	dac 100% % locked	wlc
5	100.00%	444	100.00%	173274	100 00%	253
10	100 00%	587	100 00%	86796	100.00%	275
15	100 00%	703	100 00%	57950	100 00%	201
20	100 00%	764	99 81%	n/a	100 00%	289
25	100 00%	742	99 81%	n/a	100.00%	227
30	100 00%	844	99 81%	n/a	98 87%	n/a
35	100 00%	918	100 00%	24802	100 00%	486
40	100 00%	1037	99 81%	n/a	100 00%	376
45	100 00%	1273	99 05%	n/a	99 05%	n/a
50	100 00%	1658	98 30%	n/a	98 11%	n/a
55	100 00%	1434	96 03%	n/a	96 22%	n/a
60	100 00%	2522	95 09%	n/a	95 09%	n/a
65	100 00%	3243	94 33%	n/a	94 90%	n/a
70	100 00%	2411	96 03%	n/a	95 27%	n/a
75	100 00%	3348	95 46%	n/a	94 33%	n/a
80	100.00%	3498	91 12%	n/a	90 55%	n/a
85	100.00%	4608	90 55%	n/a	88 28%	n/a
90	100 00%	5386	87 71%	n/a	84 69%	n/a
95	100.00%	5200	85 26%	n/a	82 80%	n/a
100	100 00%	7067	82 61%	n/a	79 77%	n/a
105	100 00%	10873	77 13%	n/a	76 75%	n/a
110	98 68%	n/a	75 80%	n/a	73 53%	n/a
115	96 41%	n/a	73 72%	n/a	72.21%	n/a
120	94 52%	n/a	72 97%	n/a	70 70%	n/a
125	93 01%	n/a	73 72%	n/a	71 27%	n/a
130	90 36%	n/a	70 70%	n/a	69 00%	n/a
135	88 47%	n/a	71 27%	n/a	69 00%	n/a

Table 4.4 Comparison runs for input file 2 (529 nets)

hplti = SSTL dac = divide and conquer			tw = track width 0% = 0% aggression		wlc = wire length change 100% = 100% aggression	
tw	hplth % locked	wlc	dac 0% % locked	wlc	dac 100% % locked	wlc
5	100 00%	532	100 00%	156112	100 00%	326
10	100.00%	667	100 00%	78530	100 00%	311
15	100 00%	744	100 00%	52734	99.62%	n/a
20	100 00%	796	100 00%	39487	99.81%	n/a
25	100 00%	844	100 00%	31680	99 05%	n/a
30	100 00%	889	99 81%	n/a	98 68%	n/a
35	100 00%	1016	100.00%	22730	99 24%	n/a
40	100 00%	1358	100.00%	20024	97 92%	n/a
45	100 00%	2127	99 81%	n/a	96 03%	n/a
50	100 00%	4625	99 24%	n/a	93 95%	n/a
55	100 00%	6890	97 54%	n/a	89 22%	n/a
60	100 00%	6016	96 98%	n/a	90 36%	n/a
65	100 00%	4948	97 35%	n/a	90 93%	n/a
70	100 00%	4038	97 54%	n/a	89 22%	n/a
75	100 00%	4786	94 71%	n/a	86 39%	n/a
80	100 00%	5146	92 25%	n/a	85 26%	n/a
85	100 00%	6014	89 98%	n/a	82 42%	n/a
90	100 00%	7078	85 82%	n/a	78 07%	n/a
95	99 62%	n/a	82.04%	n/a	77 50%	n/a
100	97 73%	n/a	77 88%	n/a	74 67%	n/a
105	95 65%	n/a	74 86%	n/a	71 64%	n/a
110	93 38%	n/a	72.59%	n/a	69 38%	n/a
115	91 87%	n/a	70 13%	n/a	67 67%	n/a

Table 4.5 Comparison runs for input file 3 (604 nets)

hpltli = SSTL dac = divide and conquer			tw = track width 0% = 0% aggression		wlc = wire length change 100% = 100% aggression	
tw	hpltli % locked	wlc	dac 0% % locked	wlc	dac 100% % locked	wlc
5	100 00%	422	100 00%	86592	100 00%	409
10	100 00%	506	100 00%	43602	100 00%	359
15	100 00%	571	100 00%	29187	100 00%	328
20	100 00%	628	100 00%	21997	100 00%	370
25	100 00%	649	100.00%	17509	100 00%	287
30	100 00%	661	100 00%	14764	100 00%	377
35	100 00%	690	100 00%	12628	100 00%	517
40	100 00%	736	100 00%	11102	100 00%	492
45	100 00%	797	99 83%	n/a	100 00%	362
50	100 00%	885	99 83%	n/a	100 00%	338
55	100 00%	987	99 67%	n/a	99 83%	n/a
60	100 00%	1482	100 00%	7466	99 83%	n/a
65	100 00%	1444	99 83%	n/a	100 00%	508
70	100 00%	2416	99 67%	n/a	100 00%	521
75	100 00%	2604	100 00%	6171	100 00%	451
80	100 00%	2800	99 34%	n/a	100 00%	455
85	100.00%	2600	99 01%	n/a	99 01%	n/a
90	100 00%	2613	98 68%	n/a	98 84%	n/a
95	100 00%	2173	97 19%	n/a	97 85%	n/a
100	100 00%	2234	96 52%	n/a	97 02%	n/a
105	100 00%	2306	95 36%	n/a	94 70%	n/a
110	100 00%	2283	93 71%	n/a	93 05%	n/a
115	100 00%	3392	93 21%	n/a	91 56%	n/a
120	100 00%	4715	92 05%	n/a	91 72%	n/a
125	100 00%	4369	92 55%	n/a	90 07%	n/a
130	100 00%	6873	91 23%	n/a	90 89%	n/a
135	100 00%	6625	89 07%	n/a	88 74%	n/a
140	100 00%	5745	87 91%	n/a	87 75%	n/a
145	100 00%	9851	85 10%	n/a	86 59%	n/a
150	99 01%	n/a	81 79%	n/a	82 78%	n/a
155	97 19%	n/a	80 30%	n/a	81 46%	n/a
160	95 86%	n/a	78 48%	n/a	80 46%	n/a
165	94 37%	n/a	75 83%	n/a	77 32%	n/a
170	92 88%	n/a	73 84%	n/a	75 17%	n/a

Table 4.6 Comparison runs for input file 4 (604 nets)

hplth = SSTL dac = divide and conquer			tw = track width 0% = 0% aggression		wlc = wire length change 100% = 100% aggression	
tw	hplth % locked	wlc	dac 0% % locked	wlc	dac 100% % locked	wlc
5	100.00%	430	100.00%	265339	100.00%	784
10	100.00%	510	100.00%	133194	100.00%	735
15	100.00%	570	100.00%	90333	100.00%	713
20	100.00%	619	100.00%	68153	100.00%	787
25	100.00%	676	100.00%	53571	99.83%	n/a
30	100.00%	708	100.00%	45691	99.83%	n/a
35	100.00%	777	100.00%	39488	99.83%	n/a
40	100.00%	815	100.00%	34753	99.50%	n/a
45	100.00%	917	100.00%	31064	98.84%	n/a
50	100.00%	1605	99.83%	n/a	99.17%	n/a
55	100.00%	2454	99.50%	n/a	97.52%	n/a
60	100.00%	3162	99.50%	n/a	97.52%	n/a
65	100.00%	4035	100.00%	21611	99.01%	n/a
70	100.00%	5684	99.67%	n/a	98.51%	n/a
75	100.00%	8091	99.01%	n/a	96.36%	n/a
80	100.00%	10179	98.18%	n/a	94.37%	n/a
85	100.00%	14840	97.02%	n/a	93.87%	n/a
90	100.00%	19715	95.86%	n/a	91.56%	n/a
95	100.00%	20732	94.87%	n/a	90.23%	n/a
100	100.00%	33802	93.05%	n/a	88.08%	n/a
105	100.00%	22675	90.73%	n/a	88.08%	n/a
110	100.00%	16489	89.57%	n/a	86.09%	n/a
115	100.00%	14246	88.74%	n/a	85.93%	n/a
120	100.00%	12601	89.07%	n/a	83.77%	n/a
125	100.00%	11932	87.91%	n/a	84.27%	n/a
130	100.00%	12565	86.42%	n/a	82.28%	n/a
135	100.00%	13402	84.77%	n/a	81.62%	n/a
140	99.01%	n/a	82.95%	n/a	78.97%	n/a
145	97.68%	n/a	81.46%	n/a	78.97%	n/a
150	96.19%	n/a	76.99%	n/a	77.15%	n/a
155	94.54%	n/a	76.49%	n/a	75.50%	n/a
160	93.38%	n/a	75.33%	n/a	72.52%	n/a

Table 4.7 User times from the timex command

hplti = SSTL dac = divide and conquer		tw = track width 0% = 0% aggression		time = user time from timex 100% = 100% aggression
input file	tw	hplti time	dac 0% time	dac 100% time
1	5	16.52	5.50	4.44
	10	16.13	4.75	4.33
2	5	16.77	5.82	4.62
	10	16.74	4.89	3.97
3	5	21.01	6.80	6.30
	10	20.37	5.38	4.85
4	5	21.81	7.56	5.91
	10	21.37	5.43	5.15

4.4 Other Criteria

Although priority, track number, and net number were selected as the criteria for the baseline combination, other criteria are available. These include

- **Trunk Length** – selects the trunks spanning the largest or the smallest number of rows.
Because the trunk length is determined by the pin locations, the selections are arbitrary
- **Number Of Trunks** – selects all trunks in nets containing the largest or the smallest number of existing temporary trunks. Because the number of trunks will fluctuate during execution, the selections are random
- **Wire Length Change** – selects the trunks with the largest or the smallest wire length change value. Because a trunk's wire length change will fluctuate during execution, the selections are random
- **Number of Infringements Unadjusted** – selects trunks according to the number of other trunks in the grid that either directly or indirectly infringe with each trunk. Each trunk counts as 1 infringement no matter where on the grid it is located
- **Number of Infringements Adjusted by Distance** – selects trunks according to the number of other trunks in the grid that either directly or indirectly infringe with each trunk. Each trunk is counted as $1 / \text{the number of tracks traversed to get from the trunk in question to the other trunk}$
- **Number of Infringements Adjusted by Column Width** – selects trunks according to the number of trunks that either directly or indirectly infringe with each trunk. Each trunk is counted as $(\text{the number of columns in the grid} - \text{the number of tracks traversed to get from the trunk in question to the other trunk}) / \text{the number of columns in the grid}$
- **Number of Direct Infringements** – selects the trunks with the largest or the smallest number of other trunks that directly infringe with them. Because trunks move during execution, the selections are random

- Number of Direct Priority Infringements – selects the trunks with the largest or the smallest number of other trunks of equivalent priority that directly infringe with them. Because trunks move during execution, the selections are random.
- Number of Direct Other Infringements – selects the trunks with the largest or the smallest number of other trunks of differing priority that directly infringe with them. Because trunks move during execution, the selections are random.

When it was found that the baseline criteria combinations beat divide and conquer, it was decided not to run additional combinations.

Chapter 5

Conclusion

5.1 Analysis of Experimental Results

The basic purpose of any global routing algorithm is to place all candidate trunks. By this definition, the selective sequential trunk locking algorithm is the winner. As can be seen from the comparison run tables in chapter 4, SSTL placed 100% of the trunks long after the divide and conquer algorithm could not

However, the divide and conquer algorithm consistently ran in at most half the time of SSTL. Also, Table 4.1 shows that different criteria combinations can cause SSTL to produce a wide variety of wire length changes. This could cause problems when trying to find the lowest wire length, because finding it would require runs with all combinations. Coupled with the higher run times, finding the shortest wire lengths could become an impractical process.

5.2 Analysis of the Variations in Wire Length Change in SSTL

In an attempt to find out what was causing the wide variation in wire length changes in SSTL, the first input file was cut down to 31 nets. Running the baseline criteria combinations with this abbreviated input file produced the results shown in Table 5.1.

A common thread occurred: the lower wire length changes occurred when the combination looked for the maximum track and the higher wire length changes occurred when the combination

Table 5.1 Baseline criteria runs on the abbreviated input file with tw set to 144

tw = track width		h = maximum	l = minimum	wlc = wire length change
p = priority		t = track #	i = net #	% locked = % of total trunks locked
criteria	% locked	wlc		
hphtli	100.00%	21		
hpliht	100.00%	22		
hphiht	100 00%	29		
hphthi	100 00%	35		
hpltli	100.00%	105		
hplthi	100 00%	106		
hplilt	100 00%	120		
hphilt	100 00%	140		

looked for the minimum track. An analysis of the input file revealed that nearly all of the nets had even medians (refer to Chapter 1) with the left edges of their even median ranges located in the leftmost sixteen tracks of the grid. If a trunk were locked down inside this congested area, the trunks that directly infringe on the locked trunk must be moved. This situation alone could result in the moved trunks directly infringing on other trunks. In addition, if a trunk were moved for the first time, the trunk would split into right and left trunks. This situation would cause the one trunk to become two trunks, increasing the congestion in the immediate area. Either of these two situations would increase the chance that a created and/or moved trunk would directly infringe with a trunk it would not otherwise infringe with. Combine these two situations with an already congested area and the cycle of creation and movement would greatly increase the chance that a trunk would be locked down outside of its optimal median range (refer to the basic median sections below). The implication is clear: locking down trunks in congested areas is to be avoided whenever possible.

The order in which the nets were read in may contribute to this problem. If two trunks infringe with a locked trunk, both must be moved. Because the indirect infringement lists are built as the nets are read in, directly infringing trunks are moved in the order in which their nets appear in the locked trunk's indirect infringement list. The first trunk chosen will move until it can stop or until it hits a barrier. If the two trunks infringe on the same trunks during movement in the same direction, the second trunk will move at least one track farther than the first trunk.

Bibliography

Bibliography

- [1] Ellis Horowitz and Sartaj Sahni. Fundamentals of Computer Algorithms. Galgotia Publications (P) Ltd. New Delhi, 1985.
- [2] William Swartz and Carl Sechen. A New General Row-Based Global Router. *Proceedings of the 1993 IEEE/ACM International Conference on Computer Aided Design*, pages 491-498, November 1993
- [3] Kai-Win Lee and Carl Sechen. A New Global Router for Row-Based Layout. *IEEE International Conference on Computer-Aided Design*, pages 180-183, November 1988
- [4] Jinsheng Cong and Bryan Preas. A New Algorithm for Standard Cell Global Routing. *IEEE International Conference on Computer-Aided Design*, pages 176-179, November 1988.
- [5] Sadiq M. Sait and Habib Youssef. VLSI Physical Design Automation: Theory and Practice. IEEE Press and McGraw-Hill International, 1995
- [6] M. Sarrafzadeh and C. K. Wong. An Introduction to VLSI Physical Design. McGraw-Hill, 1996
- [7] Naveed Sherwani. Algorithms for VLSI Physical Design Automation, 2nd Edition. Kluwer Academic Publishers, 1995

[8] Le-Chin Eugene Liu, Hsiao-Ping Tseng, and Carl Sechen Chip-Level Area Routing.

Proceedings of the 1998 International Symposium on Physical Design, pages 197-204,

April 1998.

Appendices

Appendix A

Split Trunks

As was shown in the introduction, there is no way to decrease the wire length of a single-trunk Steiner tree below the wire length that results when the trunk is inside the median range. A possible means of increasing the flexibility of either algorithm involves splitting the trunk into two connected sub-trunks. An example of a split trunk is shown in Figure A.1. The smaller sub-trunks would allow the algorithms more flexibility in placement and could reduce the optimal wire length. However, splitting trunks raises some new issues.

A.1 Connecting a Split Trunk

Splitting the trunk introduces a new variable: the connector between the sub-trunks. While necessary, the connector has a significant side effect: it acts as an additional pin for each pin set. This makes odd medians even and vice versa.

A.1.1 Connector Effect on Odd Medians

Because the connector acts as a pin, the number of pins in an odd median set increases from $2n+1$ to $2n+2$. $2n+2$ can be reduced to $2(n+1)$ which satisfies the definition for an even median. Because $2(n+1)/2 = n+1$, the resulting even median will be bounded by pins $n+1$ and $n+2$, with the n pins 1 through n on the left and the n pins $n+3$ through $2n+2$ on the right. When the connector pin is added and the new set of pins is resorted, one of two situations will occur:

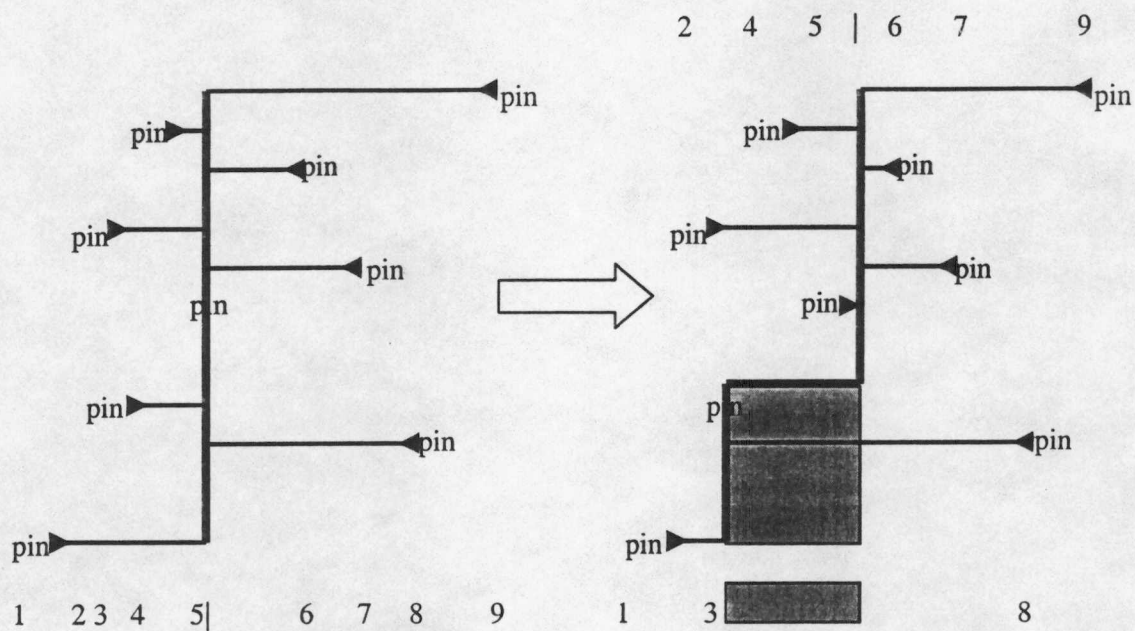


Figure A.1 Example of a split trunk. Representations of the two sub-trunks are shown above and below them. Odd medians are represented by “|”; even medians are represented by “■”.

A.1.1.1 Connector Inside Next Pin

The first situation (“Connector Inside Next Pin”) occurs when the connector pin is between the odd median and the next original pin in the direction of the connector pin. In this situation, the odd median and the connector pin become the boundaries of the resulting even median. Please note that if the connector pin has the same track number as the odd median, the two pins are in line, resulting in an odd median.

Proving this situation involves figuring out which pins the connector ends up between. If the connector pin is between pin n and the odd median (pin $n+1$), the connector pin becomes pin $n+1$, and pin $n+1$ becomes pin $n+2$. The resulting even median will be bounded by the connector pin (now pin $n+1$) and by the original odd median (now pin $n+2$). Conversely, if the connector pin is between the odd median (pin $n+1$) and pin $n+2$, the connector pin becomes pin $n+2$, leaving the odd median (pin $n+1$) unaffected. The resulting even median will be bounded by the original odd median (still at pin $n+1$) and by the connector pin (now pin $n+2$). Examples of this situation are shown in Figure A.2.

A.1.1.2 Connector Outside Next Pin

The second situation (“Connector Outside Next Pin”) occurs when the next pin from the median in the direction of the connector pin is between the median and the connector pin. In this situation, the odd median and the next pin in the direction of the connector pin become the boundaries of the resulting even median. Please note that if the next pin has the same track number as the odd median, the two pins are in line, resulting in an odd median.

As with the Connector Inside Next Pin situation, the proof involves figuring out which pins the connector ends up between. If pin n is between the connector pin and the odd median (pin $n+1$), pin n becomes pin $n+1$ and the odd median becomes pin $n+2$. The resulting even median will be bounded by pin n (now pin $n+1$) and by the original odd median (now pin $n+2$).

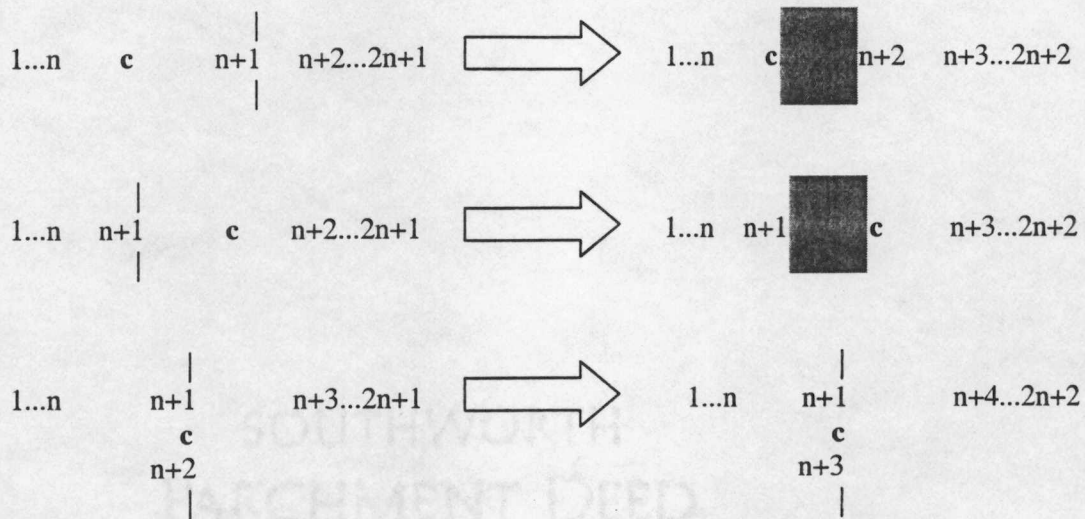


Figure A.2 Examples of the Connector Inside Next Pin situation.

Conversely, if pin $n+2$ is between the connector pin and the odd median (pin $n+1$), pins $n+1$ and $n+2$ retain their positions in the sort order. The resulting even median will be bounded by the original odd median (still at pin $n+1$) and by pin $n+2$ (still at pin $n+2$). Examples of this situation are shown in Figure A.3.

A.1.2 Connector Effect on Even Medians

Because the connector acts as a pin, the number of pins in an even set increases from $2n$ to $2n+1$, which satisfies the definition for an odd median. As such, the even median bounded by pins n and $n+1$ collapses to the odd median at pin $n+1$, with the n pins 1 through n on its left and with the n pins $n+2$ through $2n+1$ on its right. When the connector pin is added and the new set of pins is resorted, one of two situations will occur.

A.1.2.1 Connector Inside Range

The first situation ("Connector Inside Range") occurs when the connector pin is between the median pins. In this situation, the resulting odd median takes the track number of the connector pin.

Proving this situation involves figuring out which pins the connector pin ends up between. Because there are n pins to the left of an even median range and there are n pins to the right of an even median range, a connector pin inside the even median range becomes pin $n+1$. The resulting odd median is located at the connector pin (pin $n+1$). Please note that if the connector pin has the same track number as one of the even boundary pins, the resulting odd median will have the track number of that pin. Examples of this situation are shown in Figure A.4.

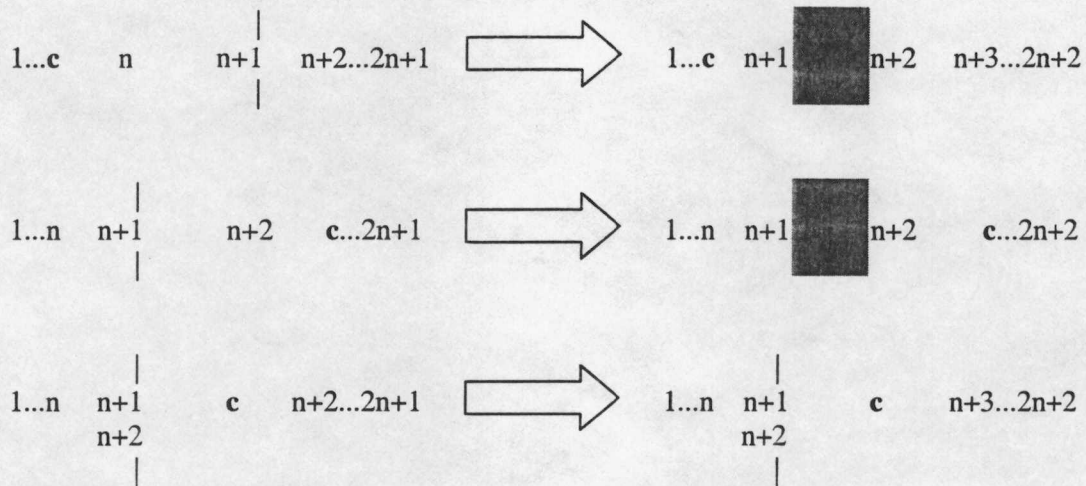


Figure A.3 Examples of the Connector Outside Next Pin situation.

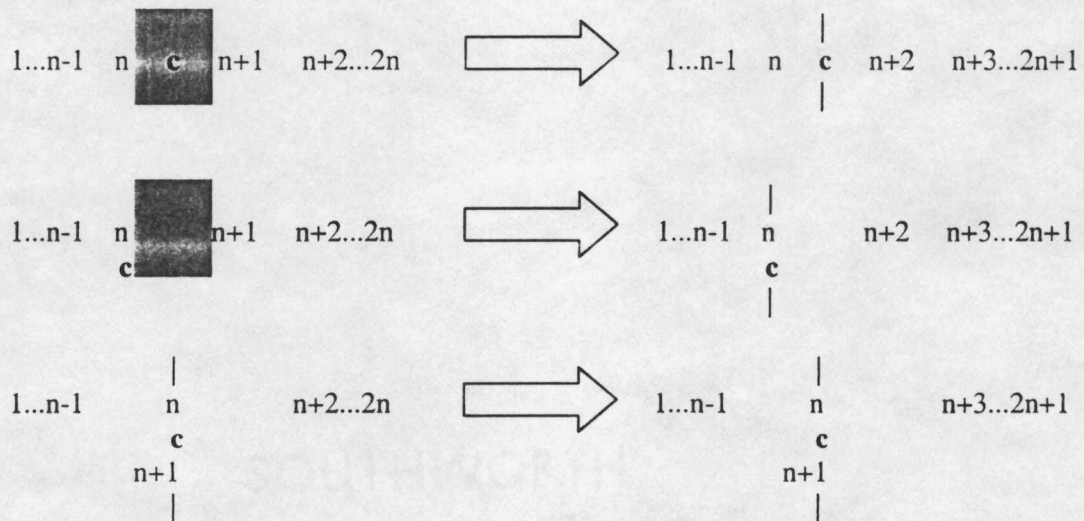


Figure A.4 Examples of the Connector Inside Range situation.

A.1.2.2 Connector Outside Range

The second situation (“Connector Outside Range”) occurs when the location of the connector pin is outside the median range. In this situation, the even median boundary pin closest to the connector pin becomes the resulting odd median.

As with the Connector Inside Range situation, the proof involves figuring out which pins the connector pin ends up between. If the connector pin has a track number less than the track number of pin n (the left-hand even median boundary), pin n becomes pin $n+1$. The resulting odd median will be located at the track number of the left-hand even median boundary pin (now pin $n+1$). Conversely, if the connector pin has a track number greater than the track number of pin $n+1$ (the right-hand even median boundary), pins n and $n+1$ retain their positions in the sort order. The resulting odd median will be located at the track number of the right-hand even median boundary pin (still pin $n+1$). Examples of this situation are shown in Figure A 5

A.2 The Interaction of Split Trunks

We have analyzed the two basic medians (odd and even). We have analyzed what happens to the two basic medians when a connector is added. We now analyze the interaction that occurs when combinations of the two basic medians are connected. There are four possible combinations of the two basic medians: two odds, two evens, one even and one odd, and one odd and one even.

A.2.1 Odd-Odd Interaction

When two odd medians are connected, the first situation that can arise occurs when the medians touch. This can only occur when the medians are in line with each other, which produces a single median. Since we already know about single medians, we will move on

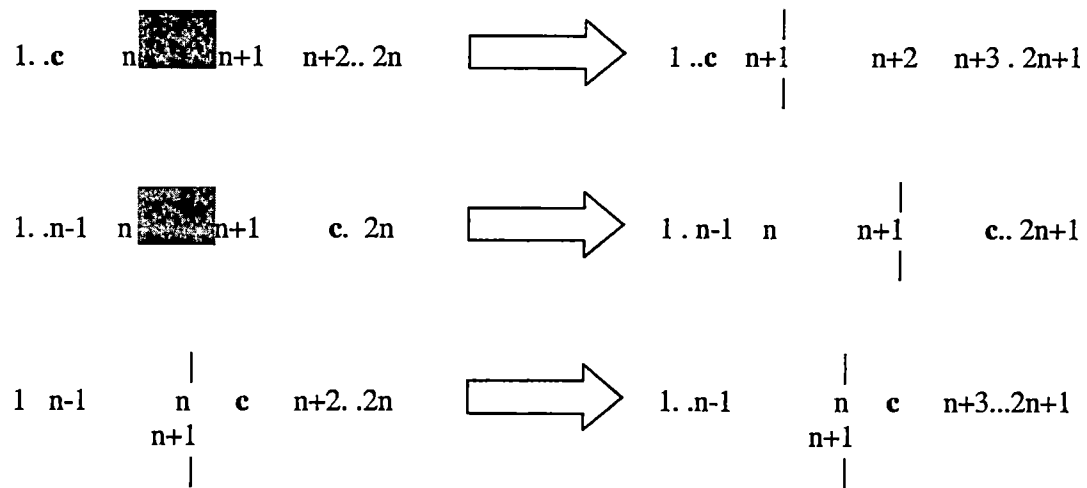


Figure A.5 Examples of the Connector Outside Range situation

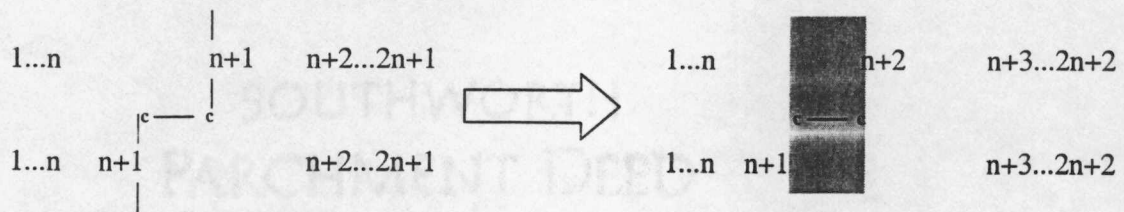
The second odd-odd situation occurs when the medians do not touch. This situation can have several variations, depending on the connector situations of the two medians.

A.2.1.1 Both Connectors Inside Next Pin

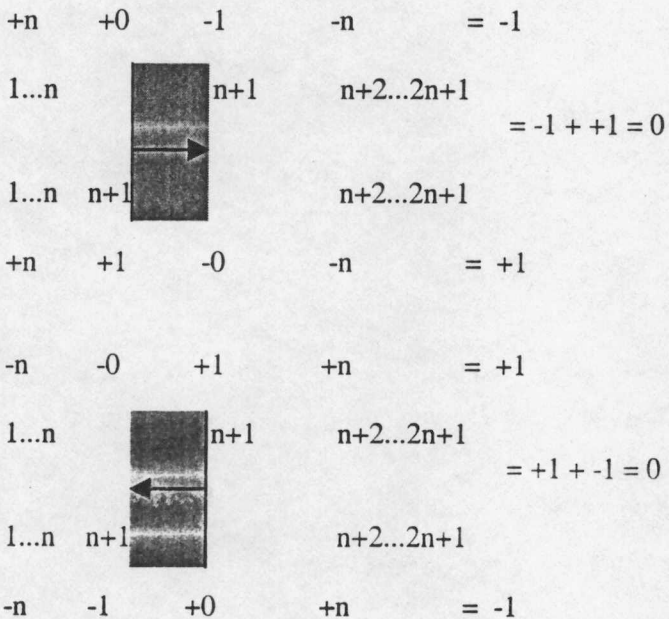
One variation occurs when the Connector Inside Next Pin situation applies to both medians. In this variation, the odd median pins become the boundary pins for the resulting even median. As expected for an even median, a single trunk moving inside the median range moves both away from and toward $n+m+1$ pins (n pins from one original odd arrangement plus $m+1$ pins from the other original odd arrangement). This allows the single trunk to move without changing the overall wire length. However, if the trunk splits, consideration must be given to the direction(s) in which the sub-trunks move. If either sub-trunk moves toward its original odd median location, it moves toward $n+1$ pins (n pins from the original odd arrangement plus the median pin) and away from $n+1$ pins (n pins from the original odd arrangement plus the connector pin). There is no penalty for this movement. However, if a sub-trunk moves away from its original location, it moves toward n pins (from the original odd arrangement) and away from $n+2$ pins (n pins from the original odd arrangement plus the odd median plus the connector pin). This causes a 2 unit penalty per unit of movement. As can be seen, it is possible to make a split trunk work in this situation, but care must be taken of what is going where. Examples of this situation and of the wire length changes associated with it are shown in Figure A.6.

A.2.1.2 Both Connectors Outside Next Pin

Another variation occurs when the Connector Outside Next Pin situation applies to both medians. In this variation, the odd medians become even medians extending from the odd median pins to the next pin in the direction of the other median. The resulting even medians might or might not touch. If the even medians do not touch, there is no overlap in their median



single trunk without connector



split trunk with connector

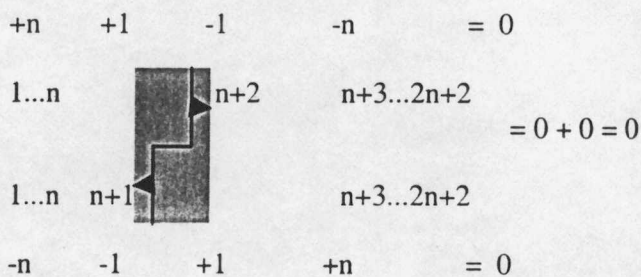


Figure A.6 Example of the double Connector Inside Next Pin situation. Examples of the wire length changes resulting from trunk movement are included.

$$-n \quad -0 \quad +2 \quad +n \quad = +2$$

$$1 \dots n \quad \begin{array}{|c|} \hline \text{Diagram} \\ \hline \end{array} \quad n+2 \quad n+3 \dots 2n+2 \quad = +2 + +2 = +4$$

$$1 \dots n \quad n+1 \quad \begin{array}{|c|} \hline \text{Diagram} \\ \hline \end{array} \quad n+3 \dots 2n+2 \quad +n \quad +2 \quad -0 \quad -n \quad = +2$$

Figure A.6 (cont.)

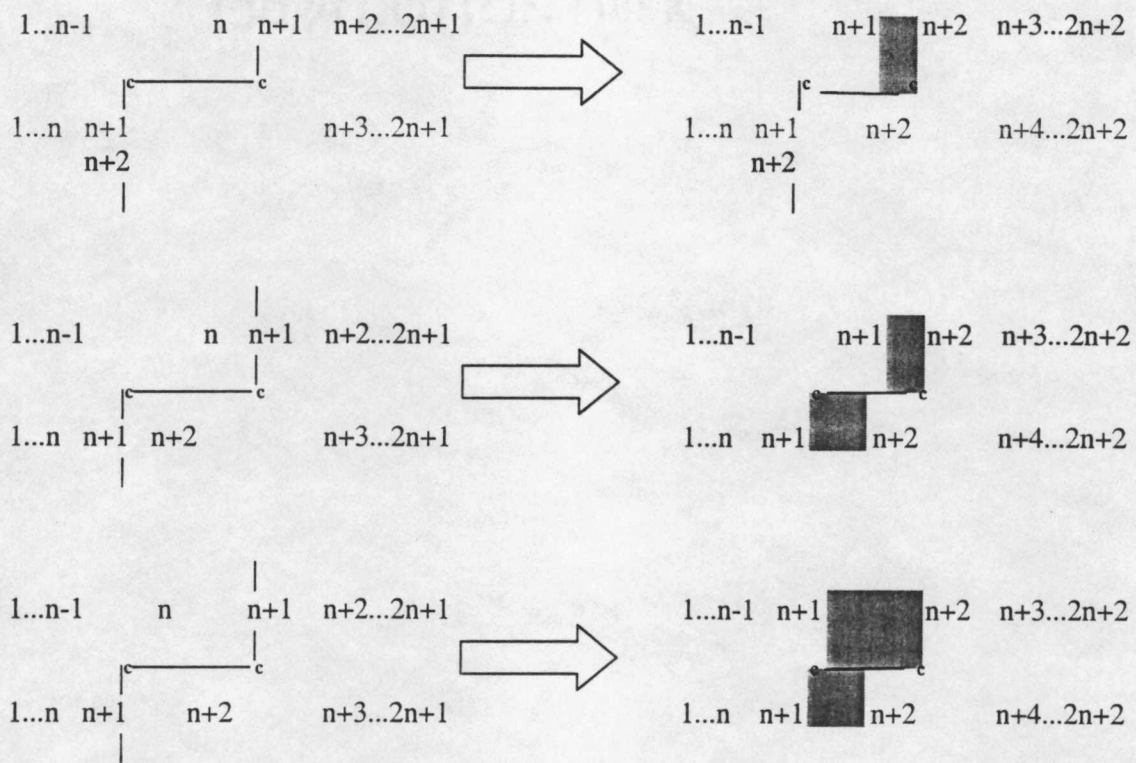
ranges. If the even medians touch, there is some overlap in their ranges. This situation can produce a single trunk or a split trunk. Because each new even median is bounded by the original odd median and the next pin toward the other median, the area between the "next" pins is where the ranges overlap. A single trunk moving in this area moves both away from and toward $n+m+1$ pins (n pins from one original odd arrangement plus $m+1$ pins from the other original odd arrangement). This allows the single trunk to move without changing the overall wire length. However, if the trunk splits, the sub-trunks behave as in the Double Connector Inside Next Pin situation above. Again, it is possible to make a split trunk work in this situation, but care must be taken of what is going where. Examples of this situation and of the wire length changes associated with it are shown in Figure A.7.

A.2.1.3 One Inside and One Outside

The third variation occurs when the Connector Inside Next Pin situation applies to one odd median and the Connector Outside Next Pin situation applies to the other odd median. The resulting even medians will overlap. However, the nature of that overlap can cause two possible cases to occur.

A.2.1.3.1 Next Pin In Same Track

The first possible case occurs when the "next" pin in the Connector Outside Next Pin situation is on the original odd median. In this possible case, the resulting even median will be limited to a single track, effectively keeping it an odd median. Because the other original median becomes an even median extending to the same track, this arrangement results in an odd median on the boundary edge of an even median. This case can produce either a single or a split trunk. A single trunk can only be located at the track number of the odd median. A split trunk can move inside the resulting even median range. This will cause the sub-trunk to move toward $n+1$ pins (n pins



single trunk without connector

$$\begin{aligned}
 &+(n-1) \quad +1 \quad -1 \quad -n \quad = -1 \\
 &1...n-1 \quad n \quad \text{[shaded box]} \quad n+1 \quad n+2...2n+1 \quad = -1 + +1 = 0 \\
 &1...n \quad n+1 \quad \text{[shaded box]} \quad n+2 \quad n+3...2n+1 \\
 &+n \quad +1 \quad -1 \quad -(n-1) \quad = +1 \\
 &-(n-1) \quad -1 \quad +1 \quad +n \quad = +1 \\
 &1...n-1 \quad n \quad \text{[shaded box]} \quad n+1 \quad n+2...2n+1 \quad = +1 + -1 = 0 \\
 &1...n \quad n+1 \quad \text{[shaded box]} \quad n+2 \quad n+3...2n+1 \\
 &-n \quad -1 \quad +1 \quad +(n-1) \quad = -1
 \end{aligned}$$

Figure A.7 Examples of the double Connector Outside Next Pin situation. Examples of the wire length changes resulting from trunk movement are included.

split trunk with connector

$$\begin{array}{ccccccc}
 +(n-1) & & +2 & -1 & -n & & = 0 \\
 1\dots n-1 & & n+1 & \xrightarrow{\quad} & n+2 & n+3\dots 2n+2 & \\
 & & & & & & = 0 + 0 = 0 \\
 1\dots n & n+1 & \downarrow & & & n+4\dots 2n+2 & \\
 & n+2 & & & & &
 \end{array}$$

$$\begin{array}{ccccccc}
 -(n-1) & & -2 & +1 & +n & & = 0 \\
 1\dots n-1 & & n+1 & \xleftarrow{\quad} & n+2 & n+3\dots 2n+2 & \\
 & & & & & & = 0 + 0 = 0 \\
 1\dots n & n+1 & \downarrow & & & n+4\dots 2n+2 & \\
 & n+2 & & & & &
 \end{array}$$

$$\begin{array}{ccccccc}
 +(n-1) & & +2 & -1 & -n & & = 0 \\
 1\dots n-1 & & n+1 & \xrightarrow{\quad} & n+2 & n+3\dots 2n+2 & \\
 & & & & & & = 0 + 0 = 0 \\
 1\dots n & n+1 & \xleftarrow{\quad} & n+2 & & n+4\dots 2n+2 & \\
 -n & -1 & +2 & & +(n-1) & & = 0
 \end{array}$$

$$\begin{array}{ccccccc}
 -(n-1) & & -2 & +1 & +n & & = 0 \\
 1\dots n-1 & & n+1 & \xleftarrow{\quad} & n+2 & n+3\dots 2n+2 & \\
 & & & & & & = 0 + 0 = 0 \\
 1\dots n & n+1 & \xrightarrow{\quad} & n+2 & & n+4\dots 2n+2 & \\
 +n & +1 & -2 & & -(n-1) & & = 0
 \end{array}$$

Figure A.7 (cont.)

$$\begin{array}{ccccccc}
 +(n-1) & +2 & -1 & -n & = 0 \\
 1\dots n-1 & n & \begin{array}{|c|} \hline \rightarrow \\ \hline \end{array} & n+2 & n+3\dots 2n+2 & = 0 + 0 = 0 \\
 1\dots n & n+1 & \begin{array}{|c|} \hline \leftarrow \\ \hline \end{array} & n+3 & n+4\dots 2n+2 \\
 -n & -1 & +2 & +(n-1) & = 0
 \end{array}$$

$$\begin{array}{ccccccc}
 -(n-1) & -1 & +2 & +n & = +2 \\
 1\dots n-1 & n & \begin{array}{|c|} \hline \leftarrow \\ \hline \end{array} & n+1 & n+2\dots 2n+1 & = +2 + +2 = +4 \\
 1\dots n & n+1 & \begin{array}{|c|} \hline \rightarrow \\ \hline \end{array} & n+2 & n+3\dots 2n+1 \\
 +n & +2 & -1 & -(n-1) & = +2
 \end{array}$$

Figure A.7 (cont.)

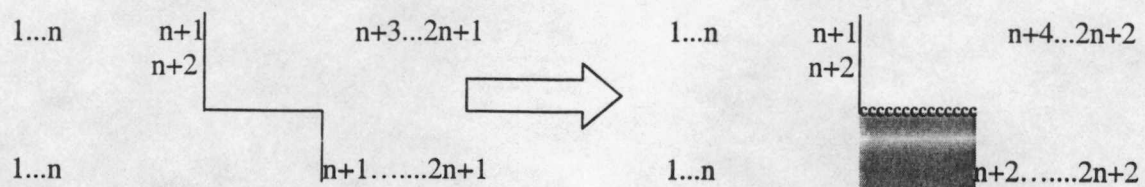
from the original odd arrangement plus the odd median pin) and away from $n+1$ pins (n pins from the original odd arrangement plus the connector pin). There is no penalty for this movement as long as the sub-trunk remains in the resulting even median range. Examples of this situation and of the wire length changes associated with it are shown in Figure A.8.

A.2.1.3.2 Next Pin Not In Same Track

The second possible case occurs when the "next" pin in the Connector Outside Next Pin situation is not in the same track as the original odd median pin. In this case, the resulting even medians will overlap from the original odd median pin to its "next" pin. This situation can produce either a single or a split trunk. A single trunk moving in the overlap area moves both away from and toward $n+m+1$ pins (n pins from one original odd arrangement plus $m+1$ pins from the other original odd arrangement). This allows the single trunk to move without changing the overall wire length. However, if the trunk splits, the sub-trunks behave as in the Double Connector Inside Next Pin situation above. Again, it is possible to make a split trunk work in this situation, but care must be taken of what is going where. Examples of this situation and of the wire length changes associated with it are shown in Figure A.9.

A.2.1.4 Odd-Odd Interaction Summary

As can be seen, any odd-odd interaction produces one of three outcomes. First, the odd medians touch, producing a single trunk. Second, the resulting even medians do not touch, producing a split trunk. Third, the resulting even medians touch. This can produce a single trunk or a split trunk, with the sub-trunks preferring movement toward their original odd medians.



split trunk with connector

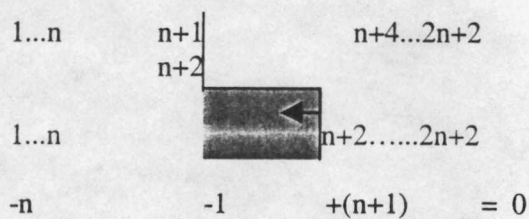
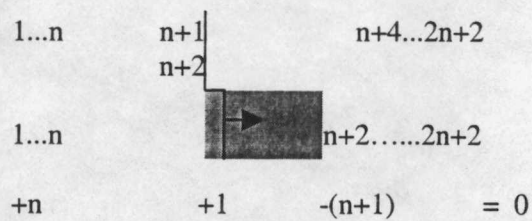
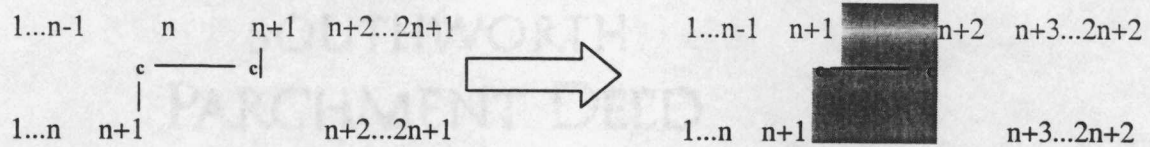
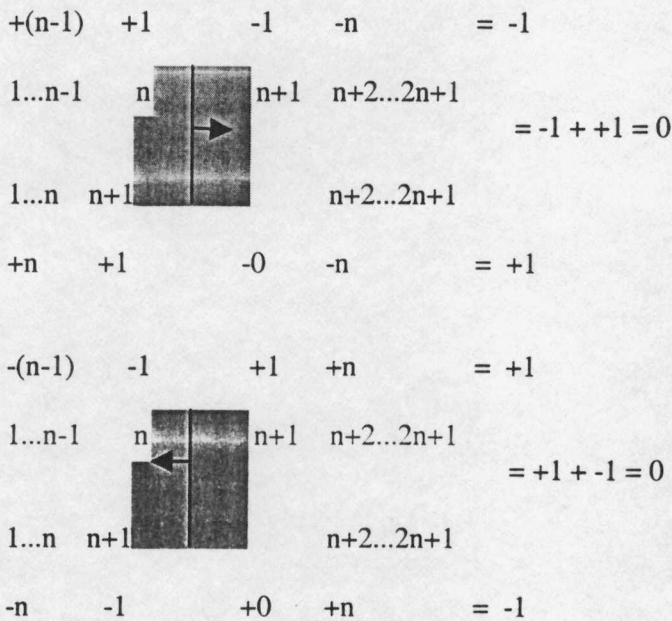


Figure A.8 Example of the Connector Inside-Outside Next Pin situation where the next pin and the median pin are in the same track. Examples of the wire length changes resulting from trunk movement are included.



single trunk without connector



split trunk with connector

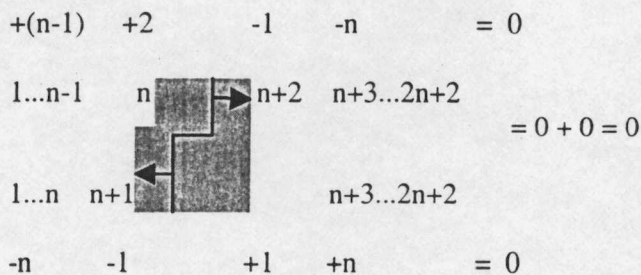


Figure A.9 Example of the Connector Inside-Outside Next Pin situation where the next pin and the median pin are not in the same track. Examples of the wire length changes resulting from trunk movement are included.

$$-(n-1) \quad -1 \quad +2 \quad +n \quad = +2$$

$$1 \dots n-1 \quad n \quad n+2 \quad n+3 \dots 2n+2 \quad = +2 + +2 = +4$$

$$1 \dots n \quad n+1 \quad n+3 \dots 2n+2$$

$$+n \quad +2 \quad -0 \quad -n \quad = +2$$

Figure A.9 (cont.)

A.2.2 Even-Even Interaction

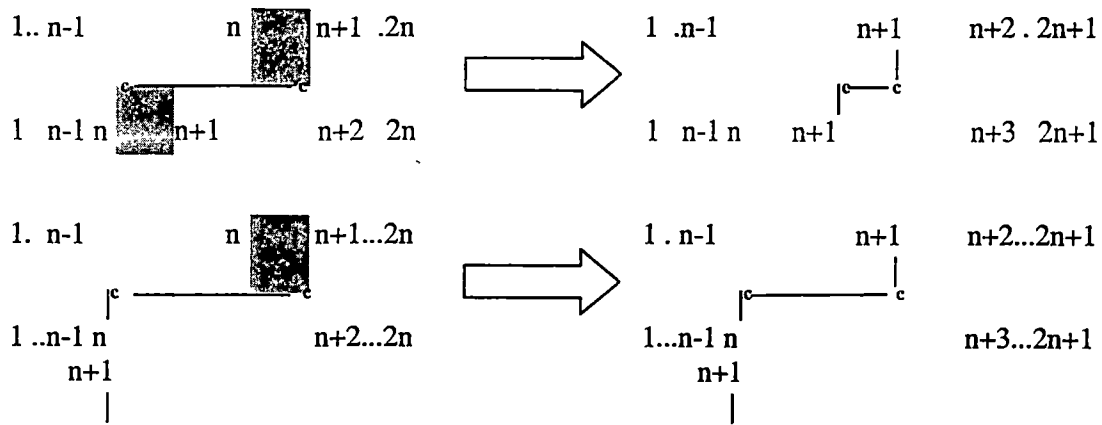
When two even medians are connected

A.2.2.1 Ranges Do Not Touch

The first situation that can arise occurs when the median ranges do not touch. For each of the medians, this is a Connector Outside Range situation. As was shown in the Connector Outside Range situation explanation, each of the even medians collapses into an odd median located on the boundary pin closest to the other median. Because each median's connector pin is outside the other median's range, the two medians do not touch. This situation can only produce a split trunk. Examples of non-touching even medians and of the wire length changes associated with them are shown in Figure A 10.

A.2.2.2 Ranges Touch

The second situation that can arise occurs when the medians touch. Because this situation results in two ranges acting on each other, the result will be as if all possible median locations for one median act upon the other median. For this reason, the even medians will remain even medians, but will adjust their ranges according to their orientation to each other. If one median's boundary edge is inside the range of the other median, the Connector Inside Range situation causes the connector pin to become one of the other median's boundary edges. Similarly, if one median's boundary edge is outside the range of the other median, the Connector Outside Range situation causes one of the other median's boundary edges to become whichever of its original boundary edges is closest to the connector pin. The end result is that the original even medians become a single full-length even median with a range corresponding to the area in which the original medians touched. A single trunk moving in this range moves both away from and toward



split trunk with connector

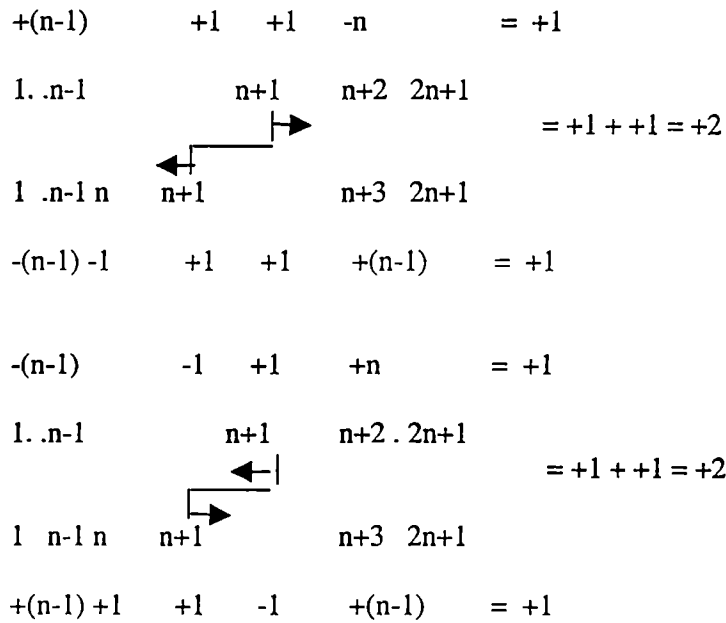


Figure A.10 Examples of two even medians that do not touch. Examples of the wire length changes resulting from trunk movement are included.

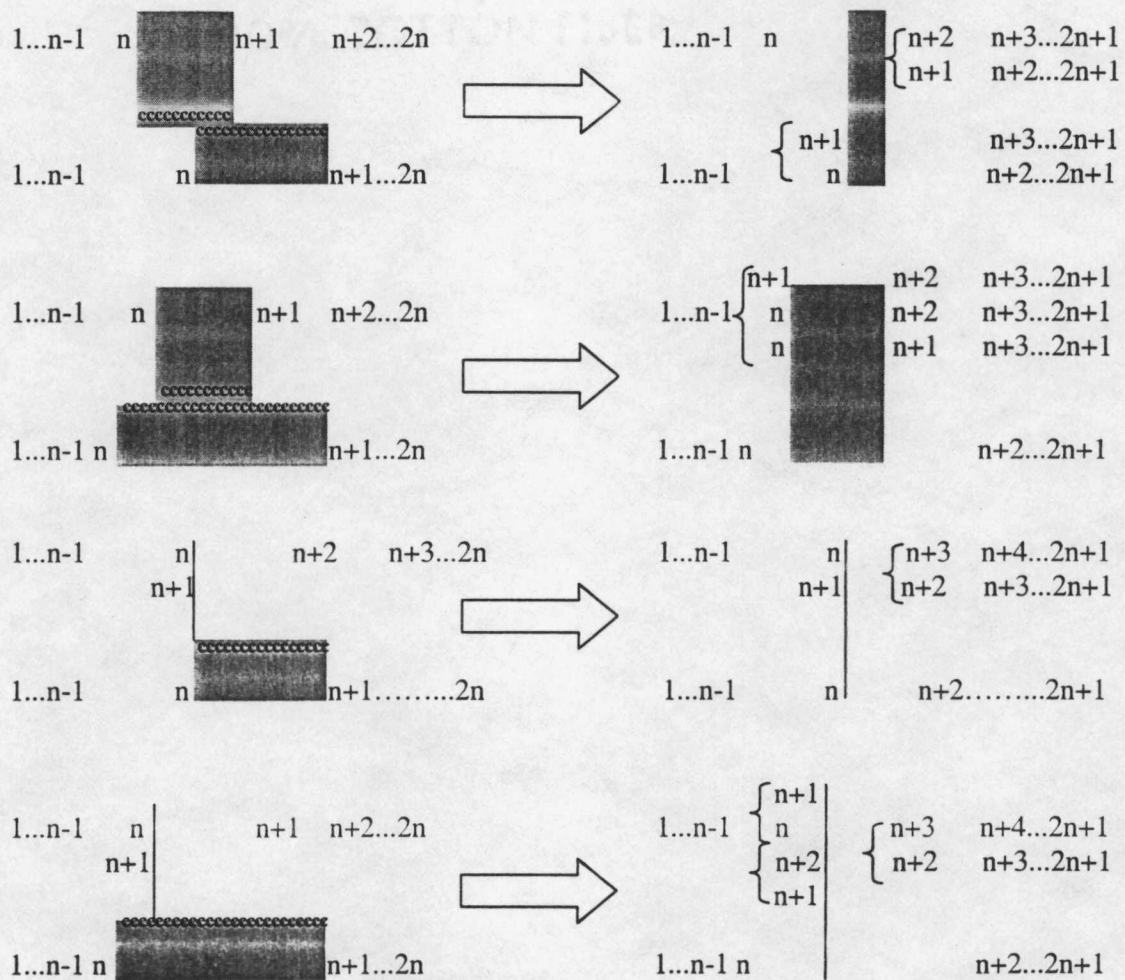
$m+n$ pins (m pins from one original even arrangement plus n pins from the other original even arrangement). This allows the single trunk to move inside the median range without changing the overall wire length. If the trunk splits, each sub-trunk will be moving toward half of its pins, but it will be moving away from half of its pins plus the connector pin. This will cause the overall wire length to increase any time the trunk splits. In the interest of minimal wire length change, this situation should only produce a single trunk moving inside the resulting full-length even median range. Examples of touching even medians and of the wire length changes associated with them are shown in Figure A 11

A.2.3 Even-Odd Interaction

When an even median and an odd median are connected, two basic situations can occur: the medians touch or the medians do not touch

A.2.3.1 Medians That Touch

The first situation that can arise occurs when the medians touch. For the original even median, this is a Connector Inside Range situation. As was shown in that situation's explanation, the original even median collapses into an odd median located on the track of the original odd median. For the original odd median, both connector situations can apply. Because the connector pin from the original even median can be located anywhere inside that median, use the boundary edges from the original even median for situation determination. Also, the fact that the original even median boundaries could extend on both sides of the original odd median causes the original odd median to expand into an even median surrounding the original odd median pin. If the Connector Outside Next Pin situation applies on either side of the original odd median, that side becomes a range extending from the original odd median pin to the next pin from the median in that direction. If the Connector Inside Next Pin situation applies to either side of the original odd



single trunk without connector

$$+(n-1) \quad +1 \quad +0 \quad -1 \quad -(n-1) \quad = 0$$

$$1...n-1 \quad n \quad \begin{array}{c} \text{trunk} \\ \rightarrow \end{array} \quad n+1 \quad n+2...2n \quad = 0 + 0 = 0$$

$$1...n-1 \quad n \quad \begin{array}{c} \text{trunk} \\ \rightarrow \end{array} \quad n+1...2n$$

$$+(n-1) \quad +1 \quad +0 \quad -n \quad = 0$$

Figure A.11 Examples of two even medians that touch. Examples of the wire length changes resulting from trunk movement are included.

$$\begin{array}{ccccccc}
 -(n-1) & -1 & +0 & +1 & +(n-1) & = 0 \\
 1\dots n-1 & n & & n+2 & n+3\dots 2n & & \\
 & & \text{[Diagram: A vertical bar with a left-pointing arrow inside, representing a connector.]} & & & & = 0 + 0 = 0 \\
 1\dots n-1 & n & & n+2\dots 2n & & & \\
 -(n-1) & -1 & +0 & +n & = 0
 \end{array}$$

split trunk with connector

$$\begin{array}{ccccccc}
 +(n-1) & +1 & +1 & -1 & -(n-1) & = +1 \\
 1\dots n-1 & n & & n+2 & n+3\dots 2n+1 & & \\
 & & \text{[Diagram: A vertical bar with a right-pointing arrow inside, representing a connector.]} & & & & = +1 + +1 = +2 \\
 1\dots n-1 & n & & n+2\dots 2n+1 & & & \\
 -(n-1) & -1 & +1 & +n & = +1 \\
 \\
 -(n-1) & -1 & +0 & +2 & +(n-1) & = +1 \\
 1\dots n-1 & n & & n+2 & n+3\dots 2n+1 & & \\
 & & \text{[Diagram: A vertical bar with a left-pointing arrow inside, representing a connector.]} & & & & = +1 + +1 = +2 \\
 1\dots n-1 & n & & n+2\dots 2n+1 & & & \\
 +(n-1) & +2 & -0 & -n & = +1
 \end{array}$$

Figure A.11 (cont.)

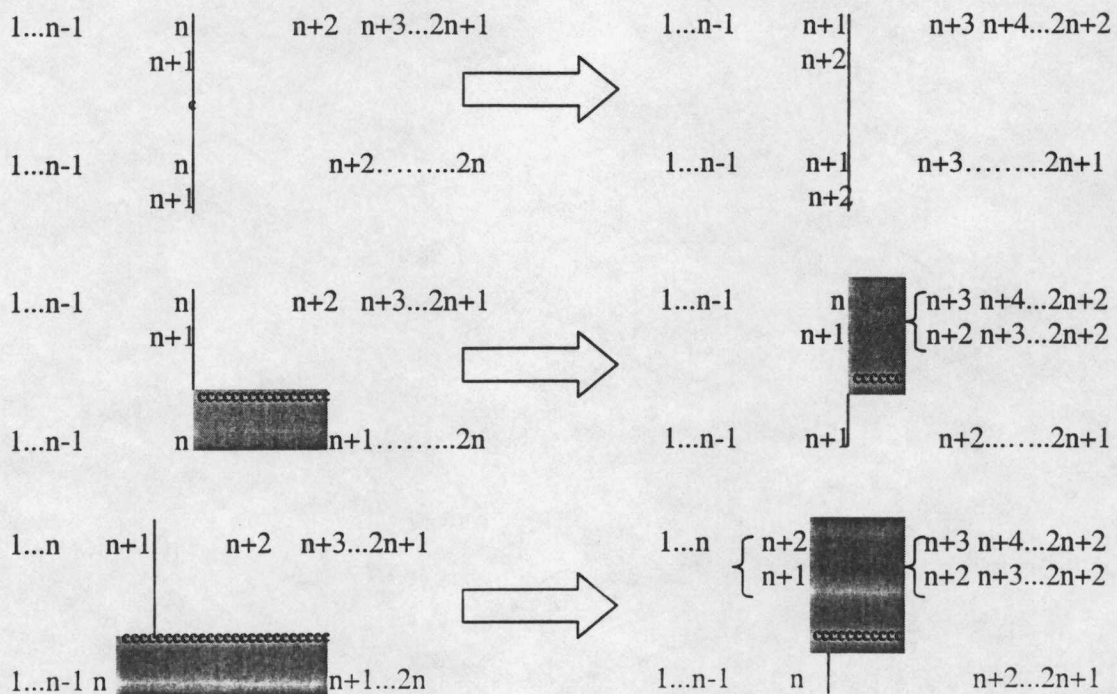
median, that side becomes a range extending from the original odd median pin to the original even median boundary on that side. Because the two medians touch at the location of the odd median, this situation can produce either a single or a split trunk. A single trunk can only be located at the track of the odd median. When the trunk splits, any movement of the sub-trunk in the resulting even median produces a result similar to movement of the original odd median (refer to the proof of the basic odd median situation in Chapter 1). In fact, the addition of the connector increases the additional wire length. Therefore, the only practical arrangement is a single trunk located on the track of the original odd median. Examples of touching even-odd medians and of the wire length changes associated with them are shown in Figure A.12.

A.2.3.2 Medians That Do Not Touch

The second situation that can arise occurs when the medians do not touch. For the original even median, this is a Connector Outside Range situation. As was shown in that situation's explanation, the original even median collapses into an odd median located on the boundary pin closest to the original odd median. For the original odd median, both connector situations can apply.

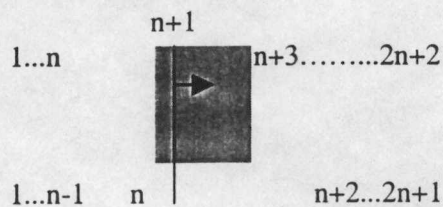
If the Connector Outside Next Pin situation applies, the original odd median becomes an even median extending from the original odd median pin to the next pin toward the original even median. Because the connector pin is outside the next pin, the two medians do not touch. This situation can only produce a split trunk.

If the Connector Inside Next Pin situation applies, the original odd median becomes an even median extending from the original odd median pin to the original even median boundary closest to it. Because the two resulting medians touch at a boundary edge of the original even median, this situation can produce either a single or a split trunk. Because the resulting odd median is located between the original odd median pin and the next pin toward the original even median,



split trunk with connector

$$+n \quad +2 \quad -n \quad = +2$$



$$-n \quad +2 \quad +n \quad = +2$$

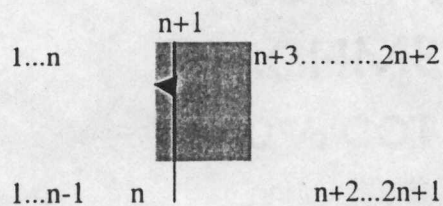
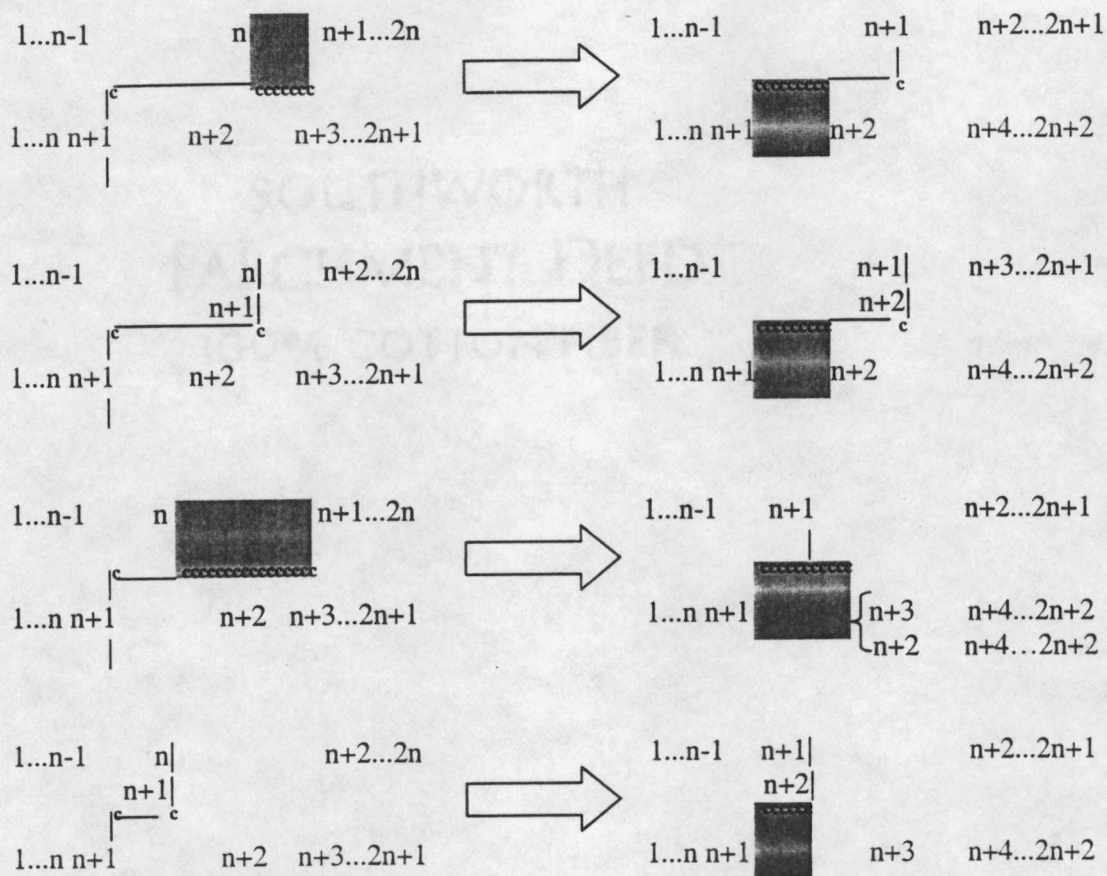


Figure A.12 Example of even-odd medians that touch. Examples of the wire length changes resulting from trunk movement are included.

trunk movement can have one of two results. A sub-trunk moving away from the original odd median is moving away from $n+2$ pins (n pins from the original odd arrangement plus the original odd median pin plus the connector pin) and is moving toward n pins (n pins from the original odd arrangement). The wire length increases by 2 units for each unit of movement in this direction. A sub-trunk moving toward the original odd median is moving away from $n+1$ pins (n pins from the original odd arrangement plus the connector pin) and is moving toward $n+1$ pins (n pins from the original odd arrangement plus the original odd median pin). The wire length is unchanged by movement in this direction. The sub-trunk behaves as in the Double Connector Inside Next Pin situation above. Again, it is possible to make a split trunk work in this situation, but care must be taken of what is going where. Examples of non-touching even-odd medians and of the wire length changes associated with them are shown in Figure A.13.



split trunk with connector

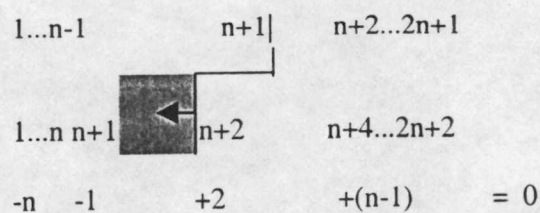
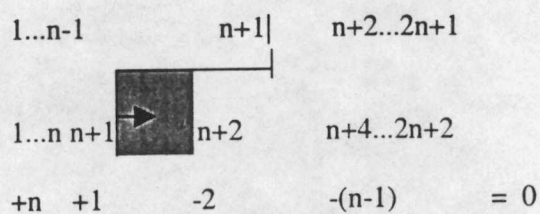


Figure A.13 Example of even-odd medians that do not touch. Examples of the wire length changes resulting from trunk movement are included.

$$\begin{array}{ccccccc}
 1 \dots n-1 & & n+1 & & n+2 \dots 2n+1 \\
 & & | & & \\
 1 \dots n & n+1 & \boxed{\rightarrow} & n+3 \dots 2n+2 \\
 +n & +1 & +1 & -n & = +2
 \end{array}$$

$$\begin{array}{ccccccc}
 1 \dots n-1 & & n+1 & & n+2 \dots 2n+1 \\
 & & | & & \\
 1 \dots n & n+1 & \boxed{\leftarrow} & n+3 \dots 2n+2 \\
 -n & -1 & +1 & +n & = 0
 \end{array}$$

Figure A.13 (cont.)

Vita

Larry Paul Cunningham was born on March 12th, 1961 in Honolulu, Hawaii. He graduated from Coffee County Central High School located in Manchester, Tennessee in 1979. He received an Associate of Science degree while majoring in Mathematics and minoring in Computer Science from Motlow State Community College in 1981. He received his Bachelor of Science degree while majoring in Computer Science from Tennessee Technological University in Cookeville, Tennessee in 1984. In May of 2000 he received his Master of Science degree from the University of Tennessee Space Institute.

•