ABSTRACT

TRANSISTOR LEVEL MICRO PLACEMENT AND
ROUTING FOR TWO-DIMENSIONAL DIGITAL VLSI
CELL SYNTHESIS

by
Michael Anthony Riepe

Chair: Karem A. Sakallah

The automated synthesis of mask geometry for VLSI leaf cells, referred to as the cell synthesis problem, is an important component of any structured custom integrated circuit design environment. Traditional approaches based on the classic functional cell style of Uehara & VanCleemput pose this problem as a straightforward one-dimensional graph optimization problem for which optimal solution methods are known. However, these approaches are only directly applicable to static CMOS circuits and they break down when faced with more exotic logic styles.

There is an increasing need in modern VLSI designs for circuits implemented in high-performance logic families such as Cascode Voltage Switch Logic (CVSL), Pass Transistor Logic (PTL), and domino CMOS. Circuits implemented in these non-dual ratioed logic families can be highly irregular with complex geometry sharing and non-trivial routing. Such cells require a relatively unconstrained two-dimensional full-custom layout style which current methods are unable to synthesize. In this work we define the synthesis of complex two-dimensional digital cells as a new problem which we call transistor-level micro-placement and routing. To address this problem we develop a complete end-to-end methodology which is implemented in a prototype tool named TEMPO. A series of experiments on a new set of benchmark circuits verifies the effectiveness of our approach.

Our methodology is centered around techniques for the efficient modeling and optimization of geometry sharing, and is supported at two different levels. Chains of diffusion-merged transistors are formed explicitly and their ordering optimized for area and global routing. In addition, more arbitrary merged structures are supported by allowing electrically compatible adjacent transistors to overlap during placement. The synthesis flow in TEMPO begins with a static transistor

chain formation step. These chains are broken at the diffusion breaks and the resulting sub-chains passed to the placement step. During placement, an ordering is found for each chain and a location and orientation is assigned to each sub-chain. Different chain orderings affect the placement by changing the relative sizes of the sub-chains and their routing contribution. We conclude with a detailed routing step and an optional compaction step.

TRANSISTOR LEVEL MICRO PLACEMENT AND ROUTING FOR TWO-DIMENSIONAL DIGITAL VLSI CELL SYNTHESIS

by

Michael Anthony Riepe

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical Engineering)
in the University of Michigan
1999

Doctoral Committee:

Professor Karem Sakallah, Chair
Professor Richard Brown
Professor John Hayes
Assistant Professor Marios Papaefthymiou
Professor Rob Rutenbar, Carnegie Mellon University

Two roads diverged in a yellow wood,
And sorry I could not travel both
And be one traveler, long I stood
And looked down one as far as I could
To where it bent in the undergrowth;

Then took the other, as just as fair,
And having perhaps the better claim,
Because it was grassy and wanted wear;
Though as for that the passing there
Had worn them really about the same,

And both that morning equally lay
In leaves no step had trodden black.
Oh, I kept the first for another day!
Yet knowing how way leads on to way,
I doubted if I should ever come back.

I shall be telling this with a sigh
Somewhere ages and ages hence:
Two roads diverged in a wood, and I—
I took the one less traveled by,
And that has made all the difference.

The Road Not Taken, Robert Frost (1874–1963)

© Michael Anthony Riepe 1999
All Rights Reserved

In memory of my father

ACKNOWLEDGMENTS

This dissertation represents the end of a long and winding road which I have followed here at the University of Michigan, and I did my best to explore every fork. I will always be indebted to my advisor, Karem Sakallah, for his patient support. He always let me follow my own course, but his well timed advice never failed to prove its value. Karem always treated me as a colleague and a friend, and I expect our relationship to continue and grow after I leave his immediate care.

I am also grateful to the remaining members of my committee, John Hayes and Marios Papaefthymiou of the Advanced Computer Architecture Lab, Richard Brown from the Solid State Electronics Lab, and Rob Rutenbar from Carnegie Mellon University. My association with John and Marios through the weekly CAD reading group contributed immensely to my enjoyment of this field. In my second life as a member of the GaAs MIPS and PUMA research groups I have built a tremendous variety of skills in the field of VLSI circuits, and my involvement with these projects provided the inspiration for the ideas that culminated in this work. I can thank Rich for keeping my electrical engineering skills intact while I was forced to program computers, and for providing a tremendously stimulating environment in which to explore new ideas. I owe a special thanks to Rob for his generous collaboration, as a great deal of this work grew from the ground-breaking research of his past and present students at Carnegie Mellon.

My stay at Michigan would have been far less interesting in the absence of the vigorous intellectual environment provided by my friends and fellow students in ACAL. I wish to thank Dave Nagel, Mike Upton, Tom Huff, Ajay Chandna, and P.J. Shearhart of the GaAs MIPS project for showing me that it is possible to graduate and move on. I will also never forget Tim Stanley, a close friend whose time with us was far too short. The past and present students of Karem’s CAD group deserve special mention. These include Tim Burks, Matthew Jones, João Paulo Marques De Silva, Jeff Bell, Victor Kravets, Jesse Whittemore, Joon #1 and Joon #2, and honorary members David van Campenhout, Phiroze Parakh, and Juan Antonio Carballo. As vice president and only other member of the physical CAD research cabal (until the arrival of Joon #2, I guess he can be secretary) Phiroze receives a special medal of honor. We spent many evenings at the whiteboard discussing our problems and he is responsible for more of this work than I care to admit.

Somehow I managed to gather together many friends outside of the engineering department. I do not want to leave anybody out, so I will mention nobody by name. You all know who you are. I thank you for helping to keep me sane and for always providing a source of diversion when the weather was good. Come to think of it, the weather didn't have to be all that good! On second thought I will make an exception and mention one name. After all, Omar did mention me in his acknowledgments. As a result, everyone in the field of solid state optics now knows that I broke Omar's collar bone.

Finally I have to thank my family for their love and constant support. To my parents, I guess maybe you raised me right after all, I know how proud you must be.

PREFACE

In this dissertation we define a number of special terms for concepts which are especially critical to understanding. For clarity, and to allow the reader to locate these definitions when needed, we will call out these terms in **bold** the first time that they are used. We will also call them out in places where their meaning may be clarified through example or further explanation. When we wish to emphasize a point we may call a word or phrase out in *italics*.

When describing computational algorithms we make use of a functional pseudo-code format adapted from the book "Introduction to Algorithms" by Cormen, Leiserson & Rivest [20]. We wish to minimize any similarity to a particular computer language, and our philosophy was to limit the number of semantic-free symbols to an absolute minimum as well. An example of a short algorithm (Kruskal's Minimum Spanning Tree algorithm) is given below:

```
MST-KRUSKAL($G$,$w$)
1       $A \leftarrow \emptyset$
2       for each vertex $v \in V[G]$
3           MAKE-SET($v$)
4       sort the edges of $E$ by non-decreasing weight $w$
5       for each edge $(u,v) \in E$, in order
6           if FIND-SET($u$) $\neq$ FIND-SET($v$)
7               $A \leftarrow A \cup \{(u,v)\}$
8               UNION($u$,$v$)
9       return $A$
```

This pseudo-code adopts the following conventions.

1. Algorithms are written in the proportional `Courier` font to resemble computer printouts. Variables, for example *A* above, are marked in italics. Reserved words such as **`if`** and **`for`** are marked in bold. Function names are specified in all capital letters. Detailed algorithm steps which are more clearly written in proper English sentence form, for example line #4 above, are written in ordinary un-formatted text.

2. We distinguish the assignment operator from the equality operator by using two

different symbols. The equal sign “=” is used to indicate an equality comparison while the left-arrow symbol “←” is used to indicate assignment. We freely make use of standard mathematical operators such as the set membership operator $a \in b$.

3. We eliminate superfluous parentheses whenever possible. They are only used to indicate cases of ambiguous operator precedence and to mark the operands of function calls and function definitions.

4. Indentation is used to indicate block structure. We eliminate all semantic-free symbols such as “begin” and “end” block delimiters and “do” statements in **`for`** and **`while`** loops.

5. Variables are all assumed to be local to a given procedure. We do not show the definition or initialization of variables unless necessary. The type of a variable is assumed to be correct for its use and can be derived from context. By default ordinary variables are assumed to be initialized to a zero or null value. Data structures such as the set *`V`* above are assumed to be empty when initialized. Input parameters to functions are assumed to be passed by value.

6. The elements of array variables are accessed with the use of the post-fix square bracket operator “`[]`”. Arrays are assumed to have been initialized with adequate space to hold any objects assigned to them.

7. For brevity we often make use of standard functions, such as the set functions `MAKE-SET()`, `FIND-SET()` and `UNION()` above, without defining them. They should be assumed to perform the expected actions without any side effects. If there is some ambiguity in the meaning of the name we will describe the action of the function in explanatory body text which accompanies each algorithm.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF APPENDICES

CHAPTER 1

Introduction

1.1 Motivation

The complexity of VLSI circuit designs has been increasing steadily since the planar integrated circuit was invented by Robert Noyce and Jack Kilby in 1959. This growth is especially evident in the design of high volume circuits such as microprocessors, solid state memories, and ASICs (Application Specific Integrated Circuits). The Digital Equipment Corporation Alpha 21264 microprocessor, which achieved “full production power-on” in June 1998 [2], features 15.2 million transistors in a 0.35µm CMOS process and runs at 500MHz. The next generation Alpha microprocessor, the 21364, is projected to debut in the year 2000 at 1000+ MHz, with 100 million transistors (8 million logic transistors, the remainder in RAM) in a 0.18µm CMOS process [2]. In 1965, Gordon Moore first made the observation, now called Moore’s Law, that this growth in performance follows an exponential curve with a doubling of chip complexity approximately every 18 months.

The 1997 edition of the SIA (Semiconductor Industry Association) National Technology Roadmap for Semiconductors [104] provides a projection of the future rate of growth in all aspects of semiconductor manufacturing. Some of their estimates are shown below in Table 1. However, it has been observed that this rate of growth in chip complexity currently exceeds the rate of growth in designer productivity that results from new developments in design automation. From 1997 to 2012 the data in Table 1 shows a 38% compound annual growth rate (CAGR) in the number of transistors per chip. According to the SIA roadmap[1], designer productivity has historically increased at a rate of only 21% CAGR. This gap between design complexity and designer productivity, which grows at a rate of approximately 18% per year, has in the past been addressed by increasing the size of design teams. However, this approach is experiencing diminishing returns as

1. [104] page 24.

Table 1: 1997 National Technology Roadmap for Semiconductors

year	1997	1999	2001	2003	2006	2009	2012
# transistors/micro-processor chip	11M	21M	40M	76M	200M	520M	1.40B
microprocessor chip size (mm^2)	300	340	385	430	520	620	750
microprocessor logic transistors/cm^2	3.7M	6.2M	10M	18M	39M	84M	180M
ASIC logic transistors/cm^2	8M	14M	16M	24M	40M	64M	100M
“feature size”, 1/2 DRAM pitch (nm)	250	180	150	130	100	70	50
microprocessor gate length (nm)	200	140	120	100	70	50	35
# wiring levels	6	6–7	7	7	7–8	8–9	9

the overhead of coordinating these large design projects becomes apparent. If current trends continue this productivity gap, shown in Figure 1, presents a looming crisis which has the potential of increasing the time to market of complex chip designs and driving up design costs. Anecdotal evidence suggests that the effects of this pending crisis are already being felt—the Alpha 21264 was projected to be in full production in the second half of 1997 [49], one year before it actually began sampling. The Intel Merced microprocessor is also projected to ship at least 8 months behind its initial scheduled debut [38]. It seems clear that the Electronic Design Automation (EDA) community must redouble its efforts to accelerate the development of new productivity enhancing design automation technology.

Engineers have traditionally addressed the issue of design complexity in integrated circuits with a hierarchical divide and conquer approach. The design is hierarchically partitioned into blocks of decreasing complexity which can be designed, placed, and routed together at one level of hierarchy and then passed up to a higher level as a fixed unit. Only limited information about each fixed block is required at higher levels: its size and aspect ratio, the locations of its ports, a model for its logic and timing behavior, etc.

At the bottom of this hierarchical design process lies the **leaf cell layer**. These leaf cells (or just “**cells**”) are circuits of between one and about one hundred transistors, and represent

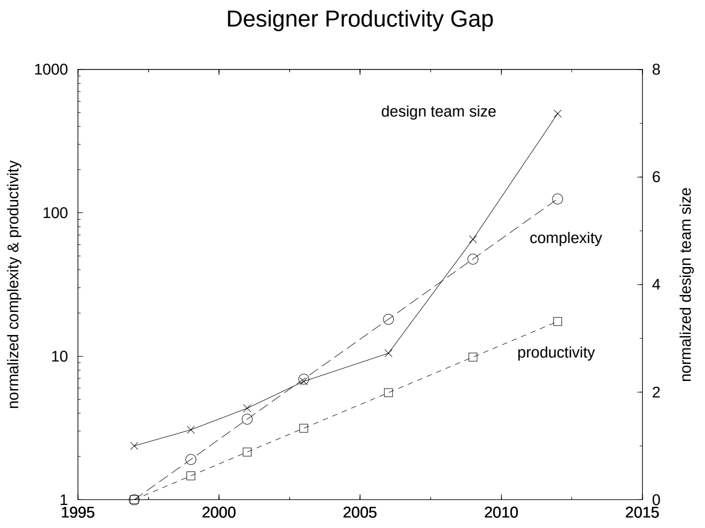


Figure 1: A projection of the increasing gap between design complexity and designer productivity

objects on the order of individual logic gates and flip-flops. The leaf cells exist for two primary reasons. The first is that they encapsulate the complexity of detailed transistor-level electrical design and analysis. While a designer may be required to perform repeated 2D or 3D parasitic extractions and time-domain transient simulations of a cell under design, the final electrical behavior of the cells can be characterized into higher level macro-models for use at the next level of hierarchy, reducing the effort required to analyze and verify the correct behavior of the complete system.

The second reason for the existence of the leaf cell library is that it encapsulates the transistor geometry into a standard set of dimensions so that they can be placed and routed by higher level tools in a regular fashion. An example of this is the popular standard-cell design style in which the cells are designed with a fixed height, but variable width, so that they can be abutted in long rows without wasting space. In addition, the standard-cell style also typically fixes the locations of cell I/O ports onto a grid so that the interconnections between cells can be made in a regular and efficient manner by higher level routing tools.

Clearly, the quality of the cells will have a direct impact on the quality of the final design. The cells must be designed to be compact and fast, with minimized power and parasitics, and with careful attention to requirements on the physical appearance of the cells as viewed by the higher-level placement and routing tools. Automated synthesis techniques have found limited application at the cell level because existing tools are unable to match the quality of human-designed cells. For this reason cells are often designed by hand, requiring a significant investment in manpower.

An additional difficulty lies in the fact that the lifetime of a typical cell library may be as short as one or two years. Compaction techniques [1] may be used to migrate a cell library to a new process technology if little more than a linear shrink is required, but this is unlikely to extend the lifetime for more than one or two generations before the loss in performance necessitates a complete redesign of the library. These problems are only becoming worse as device geometries shrink into the deep submicron regime.

In order to account for deep submicron effects, ever closer interaction is required between front-end synthesis tools and back-end placement and routing tools, power and delay optimization tools, and parasitic extraction tools. The SIA Roadmap states that, among several barriers to productivity gains at the system level, there is a requirement for

> “System-level modeling tools linked through module generators and synthesis tools to the final implementation. …These tools must use contemporary module generators, behavioral synthesis, and logic synthesis tools as targets like logic synthesis tools currently use physical design tools as a target.” [1]

In order to enable this interaction, cell libraries must become ever more flexible. Multiple versions of each cell with different drive strengths are required. It may even be necessary to support versions of cells in different logic families with different power/delay trade-offs. In addition, it has been demonstrated that standard-cell placement and routing tools are able to obtain significantly higher routing quality if they have the ability to choose from multiple instances of cells with a wide variety of pin orderings. In one study, over five benchmark circuits, an average reduction in the number of routing tracks of 10.8% was demonstrated [63].

It seems clear that, as the number of cells in a typical cell library grows from the hundreds into the thousands, a dramatic increase in designer productivity will be required, necessitating a move toward more automated cell synthesis techniques. The authors of [63], in fact, advocate a move completely away from static cell libraries as we know them, toward a system which permits the automated synthesis of cells on demand. This would permit logic synthesis tools to request specific logic decompositions, doing away with the traditional technology mapping step; standard-cell and datapath placement and routing tools to request cells with an exact pin ordering; interconnect optimization tools to request cells with specific input and output impedance values; and power

1. [104] p. 30

optimization tools to request cells, perhaps from one of several different logic families, with a specific power/delay trade-off. In a first step toward this goal, Lefebvre and Liem [62] have developed a tree-matching algorithm for the technology mapping phase of logic synthesis which allows arbitrary logic functions of bounded complexity to be requested on-demand. The C5M system from IBM [9], by virtue of its short runtimes, has been described by its authors in the context of on-the-fly cell synthesis as well.

The complete realization of an on-demand cell synthesis system will require effort on many fronts:

1. Automated transistor schematic generation: constraint-driven logic family selection, netlist creation, and transistor sizing.
2. Automated cell geometry synthesis.
3. Automated cell testing and characterization.
4. Development of enabling logic synthesis, cell placement and routing, and power/delay optimization technology.

1.2 Problem Definition and Previous Work

In this dissertation we address the second item in the above list: the fully automatic synthesis of library cell mask geometry. In the remainder of this section we develop a definition for the specific problem which we will address. This definition will be presented in the context of previous work in the field.

Our input specification consists of a sized transistor-level schematic, a process technology description (design rules, parasitics, etc.), and a description of the constraints imposed by the higher-level placement and routing environment. We refer to this last item as the *cell template*. In order to synthesize circuits of hand-crafted quality in an industrial setting, the list of constraints imposed by this template can become quite complex. A partial list from [63] follows:

1. position and size of power rails
2. nature and spacing of substrate and well ties
3. constraints to/from the cell boundary (to maintain design rule correctness when the cells are abutted)
4. size and alignment of wells
5. size and offset of routing grid
6. constraints on cell size/aspect ratio (i.e. exact height constraint for standard cells)

7. I/O pin policy (i.e. gridded, staggered, etc.)
8. source/drain region strapping requirements
9. policy on non-manhattan geometry
10. routing layer policies

This entire input specification can be viewed as a set of constraints imposed on the final synthesized mask geometry. Even the schematic itself can be viewed simply as a constraint which forces the geometry to implement the desired electrical circuit. In order to simplify the geometry generation problem, many authors impose additional constraints on the structure of the geometry. The nature of these additional constraints represents a convenient way in which to characterize and differentiate the different approached which have been taken, and by which to characterize the novelty of the work presented in this dissertation. In the remainder of this section we will discuss the techniques employed by previous researchers and conclude with a characterization of our own proposed methodology.

1.2.1 One-Dimensional Cell Synthesis

The CMOS cell synthesis problem has a rich history going back approximately 15 years. Most research has centered on a formulation of the problem which was referred to as the "functional cell" in a seminal paper by Uehara and VanCleemput [119]. In this style the transistors are constrained to take on a very regular structure. The dual N and P transistors are arranged to lie in two parallel strips of diffusion so as to minimize the number of breaks in the diffusion. We will refer to layouts in this style as 1-dimensional, or 1D, layouts because the related optimization problem is only required to explore a single (horizontal) degree of freedom.

The 1D layout style can be characterized by the following six assumptions, which were summarized in [73]. A visual example is provided in Figure 2.

1. Static CMOS technology is used
2. The pMOS and nMOS subcircuits are series-parallel connections of transistors
3. The pMOS and nMOS subcircuits are geometric duals of each other
4. Each cell consists of two horizontal diffusion rows for pMOS and nMOS transistors. This implies that single p-and n-wells can be used.
5. Complimentary pMOS and nMOS transistor pairs are vertically aligned. This allows their gate terminals to be interconnected by vertical polysilicon columns without use of crossovers.

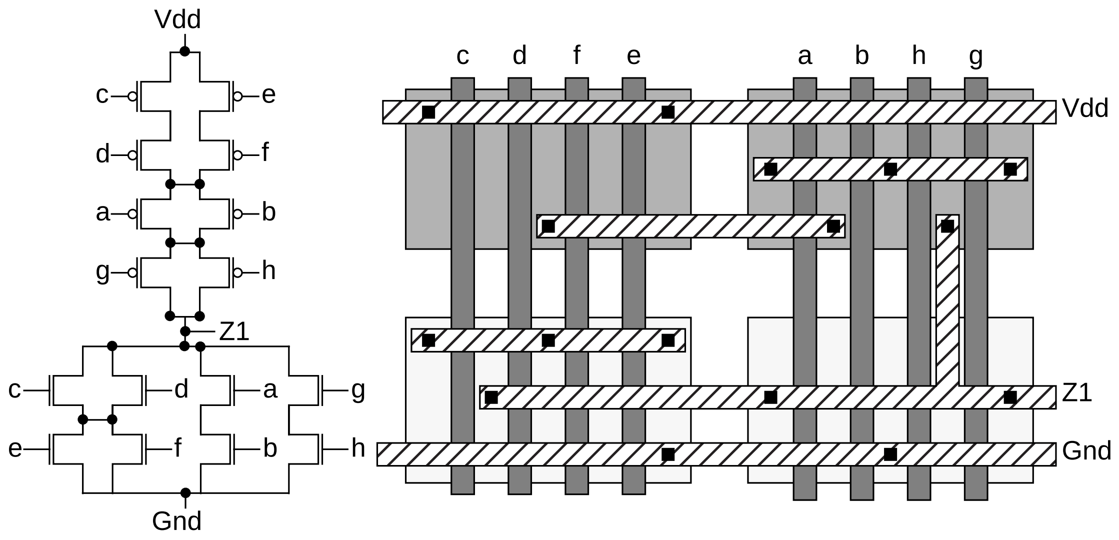


Figure 2: An example of a cell designed in the "functional cell" style of Uehara and Van-Cleemput [119].

6. Transistor drain and source terminals are connected by diffusion if the terminals are physically adjacent in the layout, or by metal if they are not adjacent. Only one metal layer is used.

Obviously, if each transistor in the p- and n-diffusion strips can be connected to both of its neighbor's source and drain terminals, then the width of the cell will be proportional to 1/2 the number of transistors in the circuit. However, if a transistor cannot be connected to one of its neighbors, some extra space corresponding to the diffusion separation design rule must be inserted between the transistors, increasing the width of the cell. Each such situation, either in a pMOS transistor, its dual nMOS transistor, or both, is referred to as a **diffusion break**. The width of a cell in the 1D style is therefor proportional to 1/2 the number of transistors plus the number of diffusion breaks. The optimization of such a cell is therefore obtained through the minimization of the number of diffusion breaks.

The synthesis of 1D layouts can be formulated as a straightforward graph optimization problem: the identification of a minimal dual Euler-trail covering for a pair of dual series-parallel multigraphs. Uehara & VanCleemput developed an approximate solution technique for this problem, while Maziasz & Hayes [73] presented the first provably optimal algorithms. We will review both works below.

Uehara & VanCleemput suggest the following graph-theoretic model for the diffusion break minimization problem. Two graphs are constructed, the p-side graph and the n-side graph representing the pMOS and the nMOS halves of the circuit, respectively. These graphs are defined as follows:

1. Each source/drain terminal is represented by a vertex, *v*
2. Each transistor is represented by an edge, *e*, connecting its source and drain terminals

Because the input circuits mist use static CMOS technology, the p-side and n-side graphs will be duals of each other. The linear order in which the transistors appear in one of the diffusion strips is represented in this graph model by a path in the corresponding graph which traces through every edge exactly once. In an ideal solution this path will be unbroken, implying that each transistor can be merged with both of its neighbors and no diffusion breaks will be required. Such an unbroken path in a single graph is called an *Euler path*. In a pair of dual graphs, if Euler paths can be found that traverse the edges of the two graphs in the same order, they are referred to as *dual Euler paths*. A diffusion break represents a situation in which a single dual Euler path does not exist and the graphs can only be covered using two disjoint dual euler paths.

Uehara & VanCleemput propose a simple heuristic algorithm for locating a small (hopefully minimum) number of dual Euler paths that cover a given series-parallel CMOS circuit. They make the observation that if the number of inputs to each AND/OR gate in the circuit is odd, then there exists a single dual euler path. Obviously this will not be the case in every circuit, so they introduce artificial **pseudo inputs** to every gate with an even number of inputs. These pseudo inputs represent *possible* diffusion breaks in the resulting functional cell. Pairs of pseudo inputs that are adjacent in the resulting input ordering represent only a single diffusion break, so the total number of diffusion breaks can be reduced by maximizing the number of adjacent pseudo inputs. They present a constructive heuristic algorithm that re-orders the gate inputs in the solution to maximize pseudo input adjacencies, taking advantage of the fact that a pseudo input can be chosen as any one of the equivalent inputs to a symmetric logic gate such as an AND/OR gate.

Many later authors have proposed refined solutions to this classic 1D layout problem. An excellent survey of the algorithms presented in early works, which were all heuristic in nature, is presented in [73]. Maziasz & Hayes [71,72,73] presented the first algorithms which were capable of obtaining provably optimal solutions to all instances of the problem which they considered to be of practical size[1]. Maziasz & Hayes discuss three algorithms: *TrailTrace* [71,73], *R-TrailTrace*

1. the authors limit the number of series connected transistors between VDD/GND and the output to four for performance reasons

[71,73], and *HR-TrailTrace* [72,73]. *TrailTrace* solves the problem of locating a minimum-width dual Euler path cover of a graph assuming that equivalent transistor inputs may not be re-ordered in the schematic. *R-TrailTrace* allows the gate input order to be permuted in logically equivalent series/parallel chains, which often results in smaller dual Euler covers. *HR-TrailTrace* locates a minimum width dual Euler cover, with gate input re-ordering, that, among all such covers, is one that leads to the smallest number of horizontal metal routing tracks, and thus has minimum height cell as well.

The *TrailTrace* algorithm uses a dynamic programming approach that builds up an optimal solution in a bottom-up fashion. A key observation is that there is a closed set of only 11 classes of dual trails for any pair of dual graphs, and when these trail classes are concatenated through a series/parallel of parallel/series connection, there are only 42 ways to combine these 11 classes into connected or disjoint dual Euler paths. The search begins by treating every individual transistor as a degenerate dual Euler path. It then concatenates, in a bottom-up fashion, those paths which are connected in the dual graphs, recording the complete set of possible dual euler path coverings at each step. Since each of these sets can be no larger than 42, each concatenation operation involves at most $42 \times 42 = 1764$ operations. Since $n - 1$ concatenations must be performed, the algorithmic complexity is $1764(n-1)$, or linear. In practice the authors report that the constant is generally much less that 1764, since it is rare that all 42 path classes will be present at one node in the graph.

R-TrailTrace is similar to *TrailTrace*, with one important modification. At each step in the algorithm, when a series/parallel or parallel/series operation is being performed by concatenating the sets of trail covers for the sub-trails, *R-TrailTrace* explores every possible permutation of the ordering in which the sub-trails are combined. If any new trail covers are discovered, they are added to the set of possible trail covers that are recorded for the concatenated graph. The number of permutations that must be explored exhibits exponential growth. However, since the problems are limited in size, in practice *R-TrailTrace* has been demonstrated to have acceptable performance.

In *R-TrailTrace,* two path covers are considered equivalent if they have the same path cover type from among the 42 possibilities. *HR-TrailTrace* extends the notion of equivalence to include cell height as well. Two trails are considered equivalent only if they define precisely the same trails. Thus, the classification of trail types into 42 equivalence classes cannot be performed, and the potential size of the set of possible covers becomes $O(e!)$. In addition, *HR-TrailTrace* must explore all possible left-right orientations and permutations when ordering the disjoint dual Euler paths. When a dual Euler path consists of a closed loop, *R-TrailTrace* breaks the loop at an arbitrary location to form a linear trail. *HR-TrailTrace* must explore all possible cut points. *HR-TrailTrace* computes the minimum number of horizontal metal tracks required to route the nMOS and pMOS transistors in each candidate solution and chooses the best one. An analysis of the run time complexity demonstrates that *HR-TrailTrace* runs in $O(h^{3h^2})$ where h is the height of the series/parallel circuit graph. Despite its exponential nature, experiments performed on all 3503 "practical size circuits" demonstrated an average run time of about 3 minutes on a 0.9 MIP machine.

A major drawback of the 1D layout style is that it directly applies only to fully complementary non-ratioed series-parallel CMOS circuits. Several significant systems have extended this style to cover circuits with limited degrees of irregularity. Among these are *Excellerator* [87], *LiB* [43], *Picasso* [61] and *Picasso-II* [64]. Highlights of these works are discussed in the remainder of this section.

Excellerator [87], part of the hierarchical layout system *Cadre* from AT&T Bell Labs, implemented a 1D layout style that supports non-dual schematics. Transistor placement is hierarchical, proceeding in three steps. First, transistor pairs sharing the same gate input are formed. If more than two same-polarity transistors have the same gate input, the transistors may be stacked vertically. In addition, if wide transistors above a certain threshold are present, they are **folded** (split into multiple parallel connected transistors), and may also be vertically stacked. The transistors are then formed into groups and linearly ordered such that at least one nMOS or pMOS transistor abuts with each of its neighbors. Unlike the previously discussed 1D algorithms, *Excellerator* does not attempt to maximize diffusion abutments, but instead tries to balance the degree of abutment with routing concerns using a heuristic "score" based on the number of transistors abutted, the type of electrical node abutted, and the proximity of transistors to other groups and signals they join to. After the groups are formed, they are ordered with respect to each other using an exhaustive exploration of all permutations.

Excellerator uses a very general maze routing algorithm named A-Star (after the "Algorithm A*" developed by Nilsson[80] in the context of artificial intelligence). A-Star offers a large computational advantage over traditional Lee style maze routers by biasing the search preferentially in the direction of the target subnet. In this strategy, the Lee-style “wavefront” representing the end of a partially completed route is expanded only in the direction that would lead to an optimal length route. For an individual net, this strategy is guaranteed to find an optimal solution provided that the distance cost estimate is a lower bound on the true cost. Nets are routed one at a time, but the order-induced bias is reduced through a recursive rip-up and re-route strategy. *Excellerator* begins by ordering the nets for routing based on their perceived difficulty, the most difficult nets being routed first. When a conflict is encountered with a previously routed net, the older net is ripped up and re-routed. To prevent rip-up cycling, ripped up nets are not re-routed in their entirety, but instead a recursive look-ahead strategy is employed. As soon as a conflict is discovered, the algorithm begins to recursively route the ripped-up net, attempting to make compromises much as a human designer would. Of course the depth of this recursion is limited, up to about two or three levels.

The *Picasso* system of Lefebvre et al [61] takes an approach similar to *Excellerator’s*. *Picasso* takes a gate-level view of the circuit, first partitioning the netlist into “clusters” that are amenable either to an optimal dual series-parallel row-based implementation [60] or a table lookup in terms of simple gates such as inverters and transmission gates. These clusters are then given a linear ordering using a greedy iterative improvement algorithm which tries to maximize an “affinity function”. This is a weighted function of “morphology” (the likelihood that two clusters will abut to form common diffusion connections) and “connectivity” (a weighted function of routing connections between the clusters). In the connectivity term, connections between clusters are weighted based on the type of connection being made. Internal connections not connected to inputs or outputs are given the highest weight. Internal connections which are also connected to a cell output are next, followed by cell-level inputs connected to more than one cluster. These latter two are given reduced weights because the router may choose to abandon them and cause them to be routed external to the cell. Routing is performed on a virtual grid using interval graph techniques similar to a channel router.

The dual series-parallel cluster optimization algorithms used in *Picasso* adopted an idea similar to the dynamic programming approach of *TrailTrace*. *Picasso* also categorizes sub-graphs into equivalence classes based on their structure. However, the graphs in this approach are limited to CMOS complex gates with at most three levels of logic and gates of four or fewer inputs. One of the more interesting aspects of *Picasso* is that the system relaxes the restriction that pMOS and nMOS transistors with the same input be aligned vertically with one another. When transistor gates are “split” in this way the pMOS and nMOS pair of transistors are connected with a 45-degree strip of polysilicon or they are routed horizontally in metal. This technique can often find solutions with fewer diffusion breaks than *TrailTrace* because the pMOS and nMOS Euler paths are no longer required to be duals of each other.

A later version of *Picasso*, called *Picasso-II*, is described in [64]. It uses the same basic philosophy as the original system, but with improved algorithms. The greedy iterative improvement algorithm for cluster placement is replaced with an exhaustive branch-and-bound search, and the simple channel-based router is replaced with an algorithm similar to that in *Excellerator,* but with a simpler and more efficient non-recursive rip-up and re-route strategy. Both *Picasso* and *Picasso-II* allow 45-degree bends in transistors and polysilicon gate routing, and Picasso-II also contains a much improved symbolic compaction algorithm. While *Picasso* used a simple one-dimensional compactor, *Picasso-II* uses a two-dimensional 0/1 mixed integer linear programming strategy with jog and bend insertion and wire length minimization. Compaction constraints are generated using methods described in [8]. The 0/1 variables represent either jogs or bends for which a direction has not been established, or “dual” (diagonal) constraints between objects which may slide horizontally past each other.

The *LiB* system, described by Hsieh et al [43], is also a 1D row-based system that accepts non-dual schematics. A highlight of this tool is a specialized channel router designed to handle non-rectilinear regions that result from unequally sized transistors. This router breaks the cell area up into five separate regions. The M region lies between the transistor rows; the N and P regions lie over the transistor diffusions and are the preferred routing medium; while the U and L regions lie, respectively, above and below the diffusion regions and can be used for unrestricted channel routing. All five regions can be non-rectilinear.

The LiB algorithm consists of nine steps:

1. clustering and pairing: find vertically aligned pairs, form these into clusters of strongly connected gates that can source/drain share.
2. chain formation: a branch-and-bound optimal chaining algorithm is used locally in each cluster to minimize diffusion breaks
3. chain placement: place clusters in rows using an exhaustive search procedure. If the number of clusters is greater than five a min-cut partitioning algorithm us used to reduce the size of the problem

4. routing region modeling: extract data concerning periphery geometry and blockages for each of the five routing regions.
5. large transistor folding: to ensure that folded transistors don't break chains, fold into odd number of pieces
6. route diffusion island: the N and P regions are routed first, modeled as a maximal clique linear routing problem. The tool must avoid blockages from vertical metal Vdd/GND and output (nMOS to pMOS) connections. These routes may block vertical connections needed by layer U and L region routes, so they may be ripped up.
7. net assignment: Route unrouted nets in U, L, M regions. Uses a heuristic bipartite graph based algorithm
8. detailed routing: uses *Silk*, a simulated evolution channel router with rip-up and re-route, to route remaining nets in non rectilinear routing regions.
9. compaction: converts symbolic layout into final geometry.

Work on the 1D cell synthesis problem has become fairly mature, and with the optimal solutions developed by Maziasz & Hayes, we regard this as essentially a solved problem (at least from a modeling perspective.) However, this is not to say that work in this area has ceased, as continued innovation has been seen in the performance and usability of commercial tools. Two very interesting modern industrial tools are discussed by Burns & Feldman [9], and by Guruswamy et al [37].

We conclude our discussion of 1D cell synthesis techniques by noting that, if the p-channel pull up transistors are ignored, dynamic CMOS circuits can be optimized using a single-row 1D formulation. A good summary of previous work in this area is presented by Basaran [5].

1.2.2 One and One-Half Dimensional Cell Synthesis

Cell synthesis environments which support the traditional "functional cell" style, such as those discussed in the previous section, can be extended by permitting the transistors to be placed in multiple P/N diffusion row pairs. For a large cell, the presence of multiple rows permits the cell to achieve a more square aspect ratio. In contrast to the 1D layout style, which only requires optimization in a single (horizontal) direction, multiple row optimization problems contain a new (vertical) degree of freedom. However, this freedom is discretized by the integer row number. In order to distinguish this style from the unrestricted 2-dimensional style discussed in the following section we will refer to the multiple-row style as being 1-1/2 dimensional.

Tani et al [117] and Gupta & Hayes [34,35,36] have both presented multiple-row 1-1/2 dimensional cell synthesis systems. The former discuss a heuristic technique based on min-cut partitioning while the latter developed an exact formulation, called *CLIP*, based on integer linear programming. A commercial tool, *LAS* from Cadence Design Systems [16], can generate single or multiple-row layouts in a row-based style for block-level circuits with several thousand transistors. *LAS* makes use of heuristics and user-supplied hierarchy information to perform transistor pairing and chaining, and then places and routes these chains in a symbolic environment.

The layout style supported by *CLIP* allows a user-specified number of diffusion row pairs, which are flipped in relation to their neighbors so that Vdd/GND rails can be shared. As in 1D tools, dual nMOS and pMOS transistors are paired so that common inputs can be routed vertically in polysilicon. Nets that span adjacent rows are routed in the channel between the rows in polysilicon or metal. Inter-row routing is not permitted over diffusion areas or between diffusion gaps, so nets that span non-adjacent rows must be routed around the sides of the cell. Integer variables are assigned to represent the row, location, and orientation of each transistor pair, the diffusion sharing among adjacent pairs, and vertical nets that connect transistor terminals across different diffusion rows. This problem defines *CLIP-W*, the width minimization problem.

CLIP-WH adds a new set of 0/1 variables that represent the occupancy of a net in each possible row of a vertical routing column, and locates the minimum height layout among the family of all minimum width layouts. *HCLIP* performs a hierarchical problem reduction by introducing additional constraints that force the adjacent placement of transistors which are in series-connected AND stacks. While *CLIP-WH* is capable of finding optimal solutions to problems of up to 16 transistors in a number of seconds, *HCLIP* extends the range to circuits of 30 or more transistors while preserving optimality in over 80% of the benchmark circuits that were tried. Recent work by Gupta & Hayes [36] also integrates transistor folding into the optimization problem.

1.2.3 Two-Dimensional Cell Synthesis

Despite the elegant optimization formulation permitted by the 1-dimensional and 1-1/2 dimensional functional cell abstraction, it must break down at some point. When designing aggressive high-performance circuits the designer may call upon logic families such as domino-CMOS, pseudo-NMOS, Cascode Voltage Switch Logic (CVSL), and Pass Transistor Logic (PTL). These provide the designer with different size/power/delay trade-offs than are available in a static CMOS implementation. However, these non-complementary ratioed logic families often result in physical layouts which are distinctly 2-dimensional in appearance.

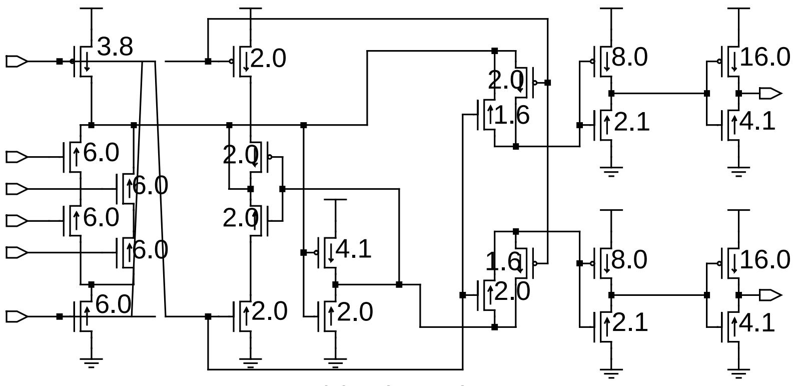
An example of such a circuit is shown in Figure 3(c). The example is a hand-designed mux-flipflop standard cell implemented in a complementary GaAs process [6]. A schematic for the circuit, and a two-row 1D version of the circuit generated by a commercial tool, are shown in Figures 3(a) and 3(b). This example demonstrates a number of properties which deviate from the standard 1-dimensional style:

1. It is non-dual and highly irregular. Some regularity is present in the rows, or "chains" of merged transistors, but these chains are of non-uniform size and are not arranged in two simple rows.
2. There are instances of complex geometry sharing, such as the "L" shaped structure in the upper-left corner.
3. The transistors are given a wide variety of channel widths.
4. The routing is non-trivial.

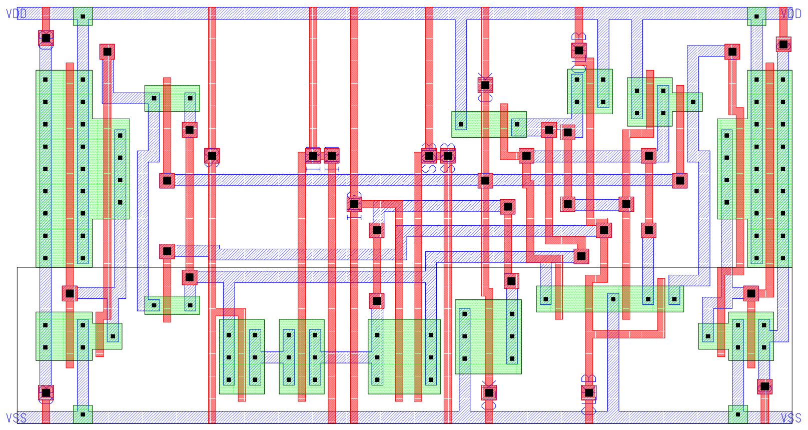
The synthesis of cell layout in an unconstrained 2-dimensional style is a complex problem which has only recently begun to receive attention. In this section we review three systems which represent the extent of current progress.

Xia et al [127] have developed a 2D model intended for BiCMOS cell generation. They treat the digital CMOS transistors in a conventional manner, clustering them into locally optimal chains to maximize diffusion sharing and minimize routing. The treatment of the bipolar transistors, capacitors, and resistors, however, requires additional complexity—bipolar transistors and passive components generally have a fixed area but a flexible aspect ratio and they are usually quite large compared to the digital FETs. A slicing-tree-based floorplanning model coupled with a branch-and-bound search engine is used to locate a compact 2D placement. Each FET cluster and each bipolar or passive device is treated as a separate static object during the placement phase. The key idea behind the slicing tree approach is that each object is assigned an analytic *shape function*, an idea attributed to Stockmeyer [114]. This shape function presents a linearized approximation of the object width as a function of its height (or vice versa). The shape functions for different objects can be added together to obtain a lower bound on the size of each group of objects as the placement in the slicing structure is recursively refined. This lower bound is used to prune the branch-and-bound search during placement.

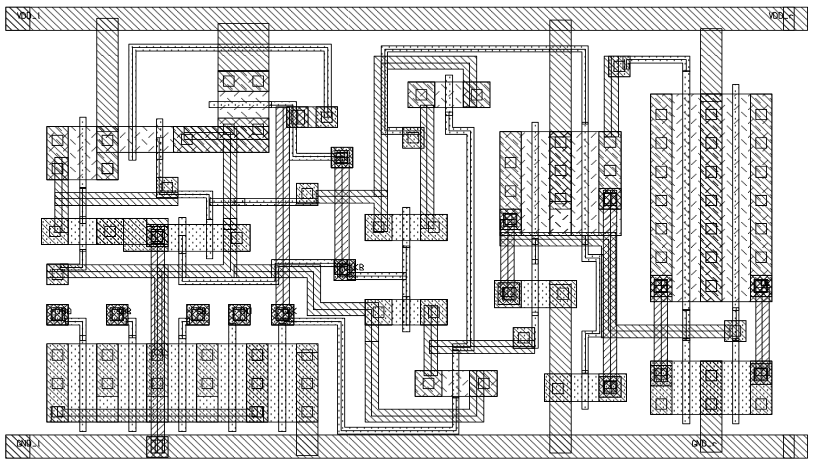
Fukui et al [28] developed an innovative system for true 2-dimensional digital transistor placement and routing. A simulated annealing algorithm is used to find good groupings of transistors into diffusion-shared chains and a greedy exploration of a slicing structure is performed to



(a) schematic



(b) 1-dimensional automatic layout



(c) two dimensional hand-drawn layout

Figure 3: A manually designed cell showing complex two-dimensional layout structure (not to scale.)

find a compact 2-dimensional virtual grid floorplan. A symbolic router is then used to perform detailed routing, and a final compaction step is used to produce a layout in the target design rules. This compactor permits transistors within chains to slide into locally optimal vertical positions within the chain if the transistors consist of a variety of channel widths.

In a more recent work by the same group, Saika and Fukui et al [95] present a second tool which operates by statically grouping the transistors into maximally-sized series chains and then locating a high quality 1-dimensional solution in order to form more complex chains. Then a simulated annealing algorithm is used to modify this linear ordering by placing the diffusion-connected groups onto a 2-dimensional virtual grid. Routing was reportedly done by hand in the prototype system.

It is also relevant to discuss work in analog circuit placement and routing in the context of 2D digital cell synthesis. The *KOAN* and *ANAGRAM-II* tools described by Cohn et. al. [18] are intended for cell-level analog circuits, but their methods could be applied to digital cells as well. *KOAN* is an analog placement tool based on a simulated annealing core that has been specially tailored to the demands of detailed analog transistor-level placement. *ANAGRAM-II* is a general line-probe router that supports iterative rip-up and re-route capabilities and analog symmetry constraints.

KOAN's simulated annealing move set consists of three types of moves: relocation moves, reshaping moves, and group moves. Relocation moves can translate, rotate, mirror, or swap objects. Reshaping moves replace the selected object with a different variant produced by the device generators. For example, a wide FET could be replaced with one folded into two smaller parallel connected FETs. In order to support important analog design concerns, object symmetry and matching constraints can be enforced during relocation and reshaping moves. Group moves are critical to the optimization of beneficial geometry sharing. If an object is selected for a relocation or reshaping move, it is checked to see if it is part of a merged structure. If a merged object is selected for relocation, a random subset of the merged group participates in the move. If a reshaping move was selected, a random decision is made whether to align the new object variant with the other objects in its group so as to retain the same overlap.

The optimization of object abutment and geometry sharing, which is critical to achieve compact results, is handled implicitly by *KOAN* which specifically encourages electrically compatible geometry overlaps. This is accomplished through the action of its unique cost function, shown below

$$\begin{aligned} \text{cost} = {} & w_0 \mathit{Overlap} + w_1 \mathit{Area} + w_2 \mathit{AspectRatio} + \\ & w_3 \mathit{NetLength} + w_4 \mathit{Proximity} + w_5 \mathit{Merge} \end{aligned} \qquad (1)$$

In Equation 1 the w_i are experimentally determined weights. Below we summarize the utility of each term.

- The *Overlap* term penalizes illegal overlaps. Instead of always forcing an annealing move to result in a design rule correct layout, arbitrary overlaps are allowed to occur when objects are moved. These illegal overlaps are penalized in the cost function and driven to zero at low temperatures. In order to ensure that adequate empty space remains in the layout for routing, each object is enclosed in a *wire halo* of empty space. The overlap of these wire halos is included in the overlap term.
- The *Area* and *AspectRatio* terms penalize the total area of the layout and the deviation from the desired aspect ratio, respectively.
- The *NetLength* term attempts to include the cost of wiring into the placement so that layout objects which are electrically connected will be placed close together. *KOAN* makes use of a minimum rectilinear spanning tree approximation to represent multi-terminal nets.
- The *Proximity* term allows a designer to improve device matching by specifying a desired clustering between arbitrary objects. It does this by placing an artificial net connecting the centers of the matched objects.
- The final term, *Merge*, is used to encourage beneficial geometry sharing between objects. If two objects are electrically connected, this term permits them to overlap and increases the benefit as the degree of overlap increases up to the maximum amount allowed by the design rules. This has the effect not only of improving the electrical behavior of the circuit by reducing parasitic capacitances and resistances, but also has the potential to improve the area as well.

ANAGRAM-II is an area router based on a line-search algorithm, which is similar to the *A-Star* router implemented in *Excellerator.* The main difference between the routing algorithm in *Excellerator* and that in *ANAGRAM-II* lies in the details of the rip-up and re-route strategy. While the former uses a recursive search strategy, the latter uses a simpler strategy called “net aging”. In *Anagram-II*, when a net is ripped up it is placed back into the routing queue and it will be re-routed at a later time. It is therefore possible that two nets could get into a competitive cycle and cyclically rip each other up in a process that never converges. In order to prevent this, a *Rip-Up* cost parameter is assigned to each net, and this cost is incremented each time the net is ripped up. Eventually a net's *Rip-Up* cost will increase to a point at which the net becomes fixed in the layout and can no longer be changed.

1.3 Proposed Methodology

In this work we discuss a problem which represents a new category of the digital cell synthesis problem: the synthesis of mask geometry for complex non-complementary digital cells. Such cells find application in modern high-speed and low-power sub-micron chip projects, and may be designed in such non-complementary logic families as CVSL, PTL, and domino CMOS. Most previous treatments of the cell synthesis problem, discussed in Section 1.2.1 and Section 1.2.2, use a 1-dimensional row-based framework which is extremely effective when applied to static CMOS complex gates, but which proves inadequate when applied to our target application (an example of this was shown in Figure 3.)

It seems clear that our problem requires us to abandon these highly constrained row-based 1-dimensional techniques and instead adopt a more flexible approach which allows true 2-dimensional transistor arrangements. In this work we present the architecture for such a 2D cell synthesis system. We approach the problem as a general transistor-level placement and routing problem, and show how existing placement and routing models and algorithms can be tailored specifically to digital transistor-level design.

Several visionary authors have begun work on 2-dimensional cell-level transistor placement and routing, as discussed in Section 1.2.3. A common theme among the existing 2D cell synthesis systems [127,28,95] is that they attempt to capture localized regularity by grouping transistors into locally optimal diffusion connected transistor rows, or "chains". The optimization of transistor geometry sharing through chaining of this type appears to be critical in obtaining compact high-performance cell layouts. However, in all three systems this grouping is performed statically before placement. *KOAN* [18], which is targeted at analog synthesis, performs no static clustering or chain formation, but instead allows these structures to form dynamically during placement. We found that *KOAN*'s dynamic cluster formation ability, when applied to digital cells, was not able to discover the large regular clusters which are characteristic of digital designs, but

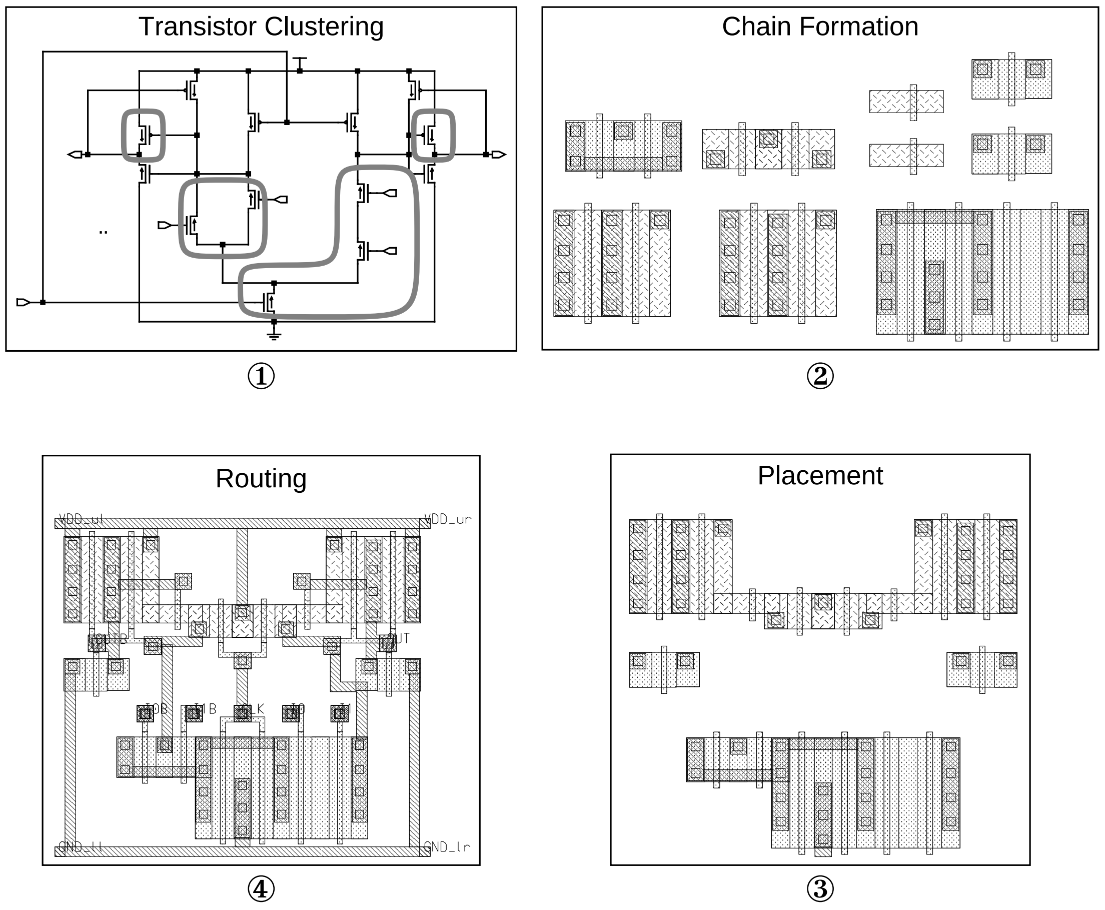


Figure 4: Basic steps in the proposed cell synthesis flow

we take inspiration from its flexibility. An important contribution in this work will be an examination of methods which permit the placement step to dynamically create transistor chains during placement.

As our top level framework we adopt the four-step process, diagrammed in Figure 5, which was first proposed by Basaran [5]. In the first step clusters of transistors are formed, each representing a set of channel-connected transistors to be composed into a single merged structure called a *transistor chain*. Clusters of cardinality one represent degenerate chains, or individual transistors. In the second step an ordering for the transistors in each chain is selected, and the geometry for each chain is generated. In the third step a placement of the chains is found. The fourth and final step generates the routing geometry to make the remaining electrical connections which were not made through direct connection by abutment.

In general, it is not required that the four steps of this methodology be performed sequen-

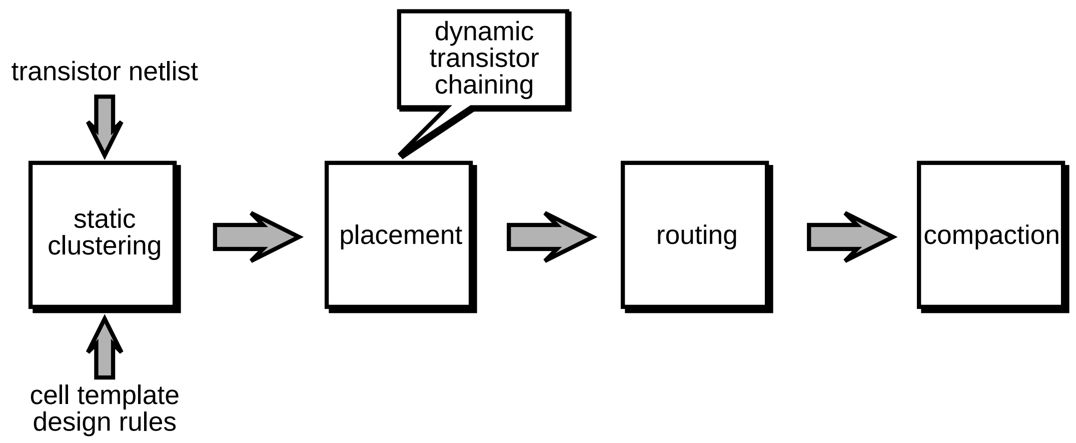


Figure 5: Proposed methodology for two-dimensional cell synthesis

tially. A sequential process would imply that optimal transistor clusters, and the resulting transistor chains, must be formed in the absence of placement and routing information. Similarly, an optimal placement of these chains must be located in the absence of detailed routing information. In the search for an optimal solution, an implementation may merge two or more of these steps, or choose to iterate between them. Our implementation, in particular, merges some aspects of transistor chain formation into the placement step, allowing transistor chaining to be explored dynamically in the presence of placement information. However, the placement and routing step remain serialized.

A flowchart showing the major steps in our proposed methodology is shown in Figure 5. We begin with a sized transistor netlist as well as a database containing the relevant design rules and cell template information. A static clustering step is performed, followed by a placement step with integrated dynamic transistor chaining. After placement, the cell is routed and a final compaction step is used to remove any remaining empty space. We discuss each of these steps in more detail below.

1.3.1 Static Transistor Clustering

Prior to placement we perform static clustering. However, our goal is to defer to the dynamic placement step as much of the transistor chain formation task as possible, so our static clustering retains as much flexibility as possible in this regard.

At this stage we use a netlist partitioning algorithm to break the netlist apart into a number of channel-connected components with a bounded upper size. Each of these components represents a structure which we call a **cluster**. If a particular linear ordering and source-drain orientation is assigned to the transistors in a cluster we obtain a unique transistor **chain**, and the resulting geometry can be generated. One or more **diffusion-breaks** may exist at points where neighboring transistor terminals represent different electrical nets and cannot be merged. We use the locations of the diffusion-breaks to partition each chain into a number of objects which we call **sub-chains**. The sub-chains are single strips of merged transistors which are free of diffusion breaks, and they represent the atomic objects which are placed during the placement step.

The ordering of the transistors in the chains is performed dynamically during the placement step, and different orderings will result in chains with diffusion breaks in different locations (however, we will show that the number of diffusion breaks stays the same.) Thus, the sub-chain to which a particular transistor is assigned, and the sizes of the resulting sub-chains, vary dynamically during placement, providing a limited degree of dynamic chaining flexibility.

1.3.2 Transistor Placement

It is the task of the placement step to assign optimal locations and orientations to each transistor and transistor sub-chain in the cell. As discussed above, the placement step is also charged with locating an optimal ordering for the transistors in each chain, which has the effect of determining the transistors assigned to each sub-chain, as well as their ordering. In this way the structure of the sub-chains can be optimized concurrently with the placement, and specific chaining decisions do not need to be made in the absence of placement information.

A major theme of this work is the development of techniques for the modeling and optimization of transistor geometry sharing—the formation of electrical connections through overlapped geometry instead of metal wires. It is our observation that the formation of transistor chains uncovers the majority of potential geometry sharing in the circuit. We therefore refer to geometry sharing as obtained through chain formation as **first-order** sharing. We also encourage less regular merged structures to form by allowing electrically compatible ports on adjacent transistors to overlap during placement. We refer to arbitrary merged objects formed in this manner as **second-order** shared structures.

The final placement must satisfy the design rules and template constraints, and it should be optimized with respect to some cost function. In order to encourage placements which are easily routeable, this cost function must include the cost of the routing as well. However, since we are performing the cell routing step serially after the placement is complete, the placement cost function must make use of simplified models for the routing. In addition, in order to guarantee that the routing will be feasible, the placement step must leave the correct amount of empty space between the placed objects. We will address our solutions for these routing cost and routing space estimation problems.

1.3.3 Cell Routing

After the placement has been completed, the remaining electrical connections which were not made by the merging of adjacent objects must be completed by the routing step. Since our placement step may have given the transistor chains a non-trivial two-dimensional arrangement, we cannot make use of simple stylized channel routers as is common with one or 1-1/2 dimensional methodologies. We make use of a third-party analog area router to perform our cell routing.

1.3.4 Cell Compaction

As one of the tasks of the placement step we have listed routing-space estimation. This task is required in order to guarantee a feasible routing. However, since the placement step is only using an approximation to estimate the area required for routing, it is inevitable that we can only guarantee feasibility by over-estimating the space that will be required. In order to remove this empty space our methodology specifies an optional compaction step following the final routing. For this step we make use of a third party non-symbolic cell-level compactor.

1.4 Contribution

This dissertation, which has been introduced above, will extend the work of previous authors in a number of areas. In order to provide the reader with a guide to the significance of what follows, we outline those areas briefly below.

- **Problem formulation.** As discussed above, our treatment of complex non-complementary logic families such as CVSL, PTL, and domino-CMOS represents a new branch of research in the classic cell synthesis problem. Circuits such as these require the use of highly unconstrained 2-dimensional layout topologies, defining a placement and routing problem for which little previous work exists.

- **Problem methodology.** In order to address the 2-dimensional placement and routing problem discussed above, we have assembled a complete end-to-end methodology. This system demonstrates new techniques for transistor chain clustering and 2D transistor placement, and makes use of existing third-party tools for detailed routing and compaction.
- **Dynamic transistor chaining.** A highlight of our 2D placement formulation is a technique which allows transistor chaining to be optimized dynamically during placement at a time when detailed routing cost information is known. All existing 2D transistor placement systems of which we are aware perform this optimization statically before placement, and must be satisfied with locating a chain ordering which is optimized only with respect to its internal routing cost.
- **Second order geometry sharing.** In addition to the “first order” geometry sharing obtained through explicit transistor sharing, we also allow more complex “second order” shared structures to form opportunistically during placement when it is determined that the ports on adjacent objects are electrically compatible.
- **Placement modeling and optimization.** While we make use of a previously known model for 2D placement, the 2D compaction constraints of Schlag et al [100] and the symbolic sequence pair representation due to Murata et al [77], we believe that we are the first to apply them at the transistor level. We make several theoretical contributions in this area, including a new algorithm for the incremental solution of the DAG SSLP problem and the formulation of two optimal sequence pair exploration algorithms, one based on integer linear programming, and one based on a branch-and-bound technique.
- **Prototype tool.** In conjunction with our modeling and optimization research we have assembled our proposed methodology into a complete prototype implementation. This implementation consists of custom software to perform the transistor clustering and 2D placement stages, along with third-party applications to perform the routing and compactions stages.
- **Experiments and benchmark circuits.** Using our prototype tool, we have conducted a series of detailed experiments to verify the utility of the ideas developed in this thesis. These experiments demonstrate the strength and flexibility of our approach, as well as several of its apparent weaknesses. In conjunction with these experiments we

assembled a set of 22 benchmark circuits which contain representative examples from a wide variety of high-performance and low-power circuit families such as PTL, CVSL, and domino CMOS. These benchmarks should prove useful to researchers pursuing CMOS cell synthesis research, as well those studying related problems such as transistor sizing and cell characterization, and they represent a significant contribution on their own.

1.5 Summary

In this chapter we have outlined the need for productivity-enhancing automatic cell synthesis tools targeted at complex non-complementary digital logic circuits. We have introduced a general methodology based on transistor-level placement and routing for the synthesis of true two-dimensional cell topologies. Our methodology attempts to form transistors into rows, or “chains”, of merged transistors to take advantage of any regularity which is present within the cells. However, unlike traditional one-dimensional synthesis techniques, these chains are not required to be complementary, and they can be placed in arbitrary two-dimensional arrangements. Unlike existing two-dimensional cell synthesis techniques, we do not form these transistor chains during a static pre-processing step, but instead allow chain formation to be optimized dynamically during the placement step. We also allow arbitrary second-order shared structures to form by allowing electrically compatible adjacent objects to merge dynamically during placement. While our emphasis is on the chain formation and placement algorithms, we will present a complete end-to-end methodology which makes use of existing third-party routing and compaction tools.

1.6 Dissertation Outline

In the remainder of this dissertation we will go into much more detail concerning the models which we use to represent the different stages of our problem, and the algorithms which we use for their optimization. One can characterize our problem, in the most general way possible, as the study of techniques for the detailed transistor-level placement and routing of digital circuits. We have found that existing techniques for transistor-level detailed routing, borrowed from the analog synthesis literature, can be applied almost unchanged to digital circuits. The placement problem, however, resembles neither the high-level block placement problem nor the analog placement problem. Consequently, we direct most of our attention to this latter aspect of the problem.

In Chapter 2 we present a detailed discussion of the modeling of the general placement problem. Here we do not concern ourselves with the details of transistor-level placement, but instead we take a higher-level viewpoint intended to deal with the placement problem in the abstract. We demonstrate the degrees of freedom presented by the placement problem and treat them as variables in a decision tree. A general symbolic modeling approach based on two-dimensional compaction constraints is adopted, and we discuss the algorithmic techniques which are required to solve this problem, as well as their computational complexity.

Chapter 3 extends the discussion of Chapter 2 with the use of a recently introduced model for symbolic placement named the *sequence pair*. We show how the sequence pair representation simplifies the placement model, and present three different formulations of the related sequence pair placement optimization problem: an integer linear programming formulation, a branch and bound formulation, and a simulated annealing formulation. In this context we present a new algorithm for the incremental solution of the DAG single-source longest path (SSLP) problem, which is integral to the solution of sequence pair constraint graphs. We conclude with some experiments demonstrating the efficiency and practicality of the different optimization formulations.

Chapter 4 carries the discussion to the specific details of transistor-level placement and our own cell synthesis methodology. Each of the steps outlined in Section 1.3 is discussed at length. Chapter 5 presents an implementation of our methodology in a prototype tool named *TEMPO*[1], and a set of experiments demonstrating the effectiveness of our two dimensional approach when compared to an industrial 1-1/2 dimensional synthesis tool. In conjunction with these experiments we assembled a new set of benchmark circuits which we also discuss in some detail. We conclude in Chapter 6 with a summary of our contribution and some thoughts on future work.

The appendices contain supplementary information which is intended to support the content of the main thesis body. Appendix A contains a detailed description of our *SPICE*-based transistor sizing tool, which makes use of a new heuristic iterative gradient descent algorithm. This tool was used to obtain realistic sizes for the transistors in the benchmark circuits. Appendix B provides the detailed numerical benchmark data which is provided in graphical form in Chapter 5. Finally, Appendices C–E show the final full-page layout plots for all 22 benchmark circuits as supplied by the commercial reference tool, our tool prior to the final compaction step, and our tool

1. Transistor Enabled Micro-Placement Optimization

after the final compaction step, respectively. Only a sampling of these plots, at greatly reduced size, are displayed in the experimental results section in Chapter 5.

CHAPTER 2

Placement Modeling Issues

In the formulation of the placement problem, the most fundamental issue which must be addressed is the selection of an appropriate modeling environment. In this chapter we examine several alternative models for the placement problem: the *direct* model and several *indirect*, or *symbolic*, models. We formulate the problem using a symbolic model which we initially write in terms of two-dimensional compaction constraints. We also review the standard methods used for the modeling of routing cost within placement.

2.1 Introduction

The choice of an appropriate placement model is critical in determining the flexibility and performance of the resulting implementations. We base our model selection on the following loose set of requirements. First, the model should be amenable to an efficient algorithmic traversal of the search space. Calculating the value of the cost function for each location in the search space should not be computationally expensive. Second, the model should make it easy to identify, or eliminate a priori, identical and infeasible placements. We would prefer not to waste time evaluating equivalent placements multiple times, or evaluating placements that do not map to a design rule correct layout.

In order to keep our discussion at a general level, we will not concern ourselves with the detailed issues involved in transistor-level placement. Those issues will be the subject of Chapter 4. The techniques in this chapter are applicable to placement problems at any level of hierarchy, and we normally restrict our discussion and examples to the placement or arbitrary rectangular blocks. In fact, we do not even consider routing issues until the last section, Section 2.4.6, when we discuss the placement cost function. The general problem of rectangle placement in the absence of routing is often called *rectangle packing* in computer science theory, a term which we will use when the discussion is in this context. In spite of our general treatment in this chapter, the choices which we make are strongly motivated by the nature of our application, and the model which we present may not be appropriate for all placement problems in general.

2.2 Direct Versus Indirect Placement

The most straightforward model for placement would be as follows: the lower-left corner of every atomic object is to be assigned an integer (x, y) coordinate on the euclidean plane[1]. In addition, each object must be assigned one of several possible orientations, corresponding to the different ways in which it may be rotated or mirrored. Finally, in the case of objects with multiple configurations, such as our sub-chains with multiple chainings, one configuration must be selected from the family of possibilities. The solution space is traversed by selecting different configurations, rotation/mirrorings, and integer grid coordinates for each placeable object. The cost function is then some quantitative measure of the resulting placement, such as its area and/or performance. Rutenbar [91] refers to this as a **direct** model for the placement problem, and it is the representation adopted by many simulated annealing placement tools such as *Timberwolf-3.2* [101,102] and *KOAN* [17,18], and by analytic placement tools such as *Gordian* [52].

The alternative to the direct placement model is referred to by Rutenbar as **indirect** placement, an example of which is the symbolic slicing-tree model which is popular in block-level floorplanning [47,120,123]. Indirect placement does not attempt to model the entire physical coordinate space, but instead constructs some form of abstract or symbolic representation for that same space. The search space is explored by manipulating this abstract model. The cost function may be directly computable from the abstract representation, or it may be necessary to map the symbolic placement back into the corresponding physical coordinate space in order to evaluate the cost function.

We believe that indirect models for placement more closely match our modeling requirements. As one traverses the placement space in a direct model, many placements will be infeasible. For example, in *KOAN* at high annealing temperatures, objects may be assigned physical coordinates such that objects overlap in ways which are design rule incorrect (see Figure 6.) *KOAN* relies on such overlaps to identify geometry overlaps which may be beneficial, and all illegal overlaps will be eliminated as the solution cools. In Chapter 4 we show how object overlaps can be handled

1. Real number coordinates may be used, but in most cases it will be adequate to restrict the object coordinates to the integers, as we will most often be working within a grid whose resolution is set by the design rules.

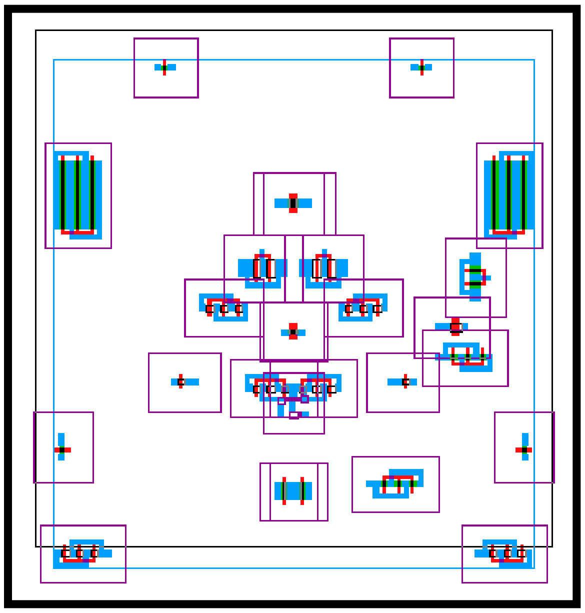


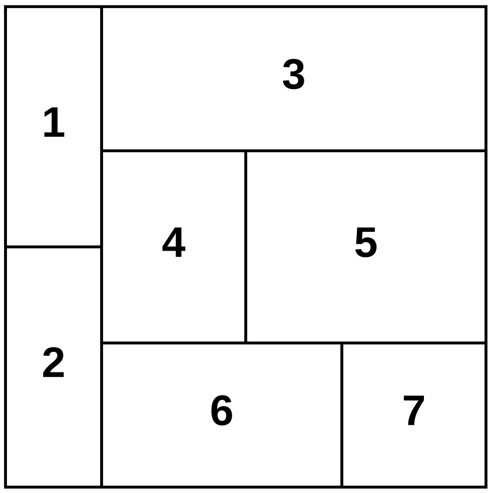
Figure 6: A screen shot of the *KOAN* tool taken at an early stage of placement showing many object overlaps

explicitly in an indirect model. In addition, many placements in a direct model may be equivalent in some sense. By equivalent we mean that, if two different layouts obtained through a direct model are compacted, they may map to the same design-rule-correct layout. It would be more efficient to traverse the search space if an indirect model can be found that eliminates all equivalent solutions a priori.

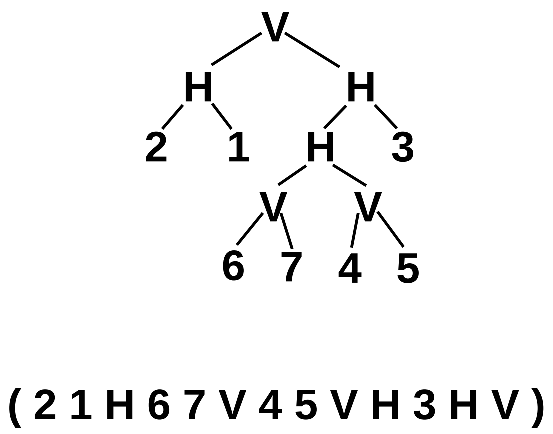
We can now state our placement modeling goal as follows: to locate a symbolic representation for object placement in which every candidate placement is 1) design-rule-correct and 2) unique in relation to all other symbolic placements. The model must allow objects full freedom to make connections through geometry sharing, and it must not exclude any placements that could be obtained through a direct model.

2.3 Indirect Placement: The Slicing Tree

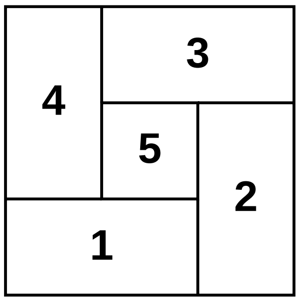
The most popular example of an indirect model has been the **slicing tree**. A good review of the slicing tree model is provided in [96, chap. 3]. We give a brief description here which is intended to serve as a counter example to the model which we do finally adopt. As shown in Figure 7, a slicing floorplan is derived by recursively slicing the layout space either horizontally or



(a) A slicing floorplan



(b) Slicing tree and post-fix symbolic representation of the floorplan in (a).



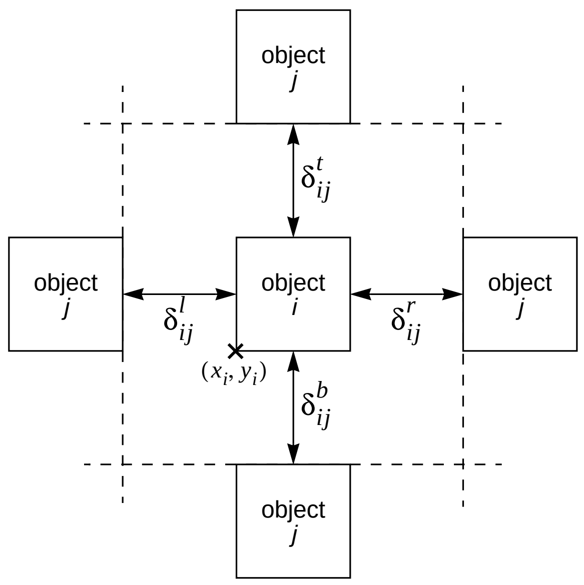
(c) A non-slicing "wheel" floorplan

Figure 7: Slicing floorplans

vertically until each object to be placed is in its own "compartment". There is a convenient symbolic representation of a slicing floorplan, which is the "Polish" expression derived from a post-fix traversal of a binary tree representing the sequence of recursive cuts. The leaves of the tree represent the operands, or placement objects, and the internal nodes are operators in the set (H, V) which represent, respectively, horizontal or vertical cuts. This representation is not implicitly canonical, multiple Polish expressions can represent the same placement[1], but a fairly straightforward convention can be adopted which leads to canonicity.

The most fundamental limitation to the slicing tree model is that some floorplans are *non-slicing*, in other words they cannot be represented by a slicing tree. The most trivial example is the so-called "wheel" floorplan shown in Figure 7(c). This limitation violates one of the fundamental

1. this occurs because two consecutive cuts in the same direction, such as the two "H" cuts separating the blocks in Figure 7(a) into the groups (3), (4,5), and (6,7) can be made in an arbitrary order, resulting in different trees.



c^r_{ij} (j is to the right of i) : $x_i + w_i + \delta^r_{ij} \le x_j$

c^l_{ij} (j is to the left of i) : $x_j + w_j + \delta^l_{ij} \le x_i$

c^b_{ij} (j is below i) : $y_j + h_j + \delta^b_{ij} \le y_i$

c^t_{ij} (j is above i) : $y_i + h_i + \delta^t_{ij} \le y_j$

Figure 8: The geometric "compaction" constraints generated between two objects i and j.

requirements which we desire in our model, namely that all placements representable in a direct model can also be represented symbolically in the indirect model.

The main disadvantage of the slicing tree model from our perspective is that it is intended primarily for use in floorplanning and not placement. In floorplanning, some or all of the objects being placed have flexible aspect ratios, while the term "placement" is generally used when the objects have fixed geometry. It is not possible to control the final shape or aspect ratio of the final compartments that result from a particular slicing, and so large amounts of dead space may remain if the objects cannot adjust their aspect ratios to match.

2.4 Indirect Placement: Two-Dimensional Compaction Constraints

The indirect placement model which we have adopted was inspired by the two-dimensional compaction research of Schlag et al [100]. In the context of block placement, this model makes the following observation: in order to maintain a legal separation δ_{ij} between two objects *i* and *j*, which have widths w_i, w_j and heights h_i, h_j, the object *j* is constrained to remain outside of a box surrounding the object *i* as shown in Figure 8. This constraint can be linearized by imposing the four separate difference constraints shown. At least one of these four constraints must be satisfied in order to obtain a design-rule-correct placement for *i* and *j*. By selecting one of the four constraints one is selecting one of four different *relative* positions for *i* and *j*. Each of these four

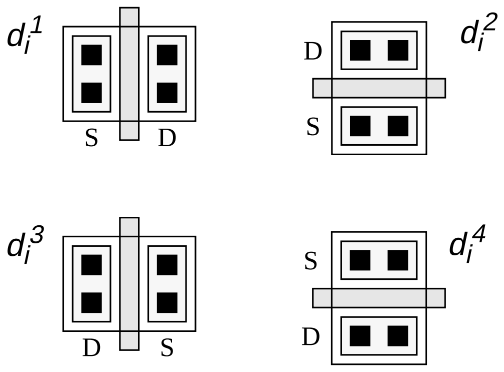


Figure 9: The four possible orientations of a FET and the corresponding decision variable.

choices can potentially (but not always) result in a different placement. A complete placement of *n* objects is obtained by selecting one of the four constraints

$$c_{ij} \in \left\{ c_{ij}^{r}, c_{ij}^{l}, c_{ij}^{b}, c_{ij}^{t} \right\} \tag{2}$$

for all $i, j \in 1 \ldots n$. The reason why different selections for c_{ij} may not always result in different placements is that, depending on the relative placements of the other $n - 2$ objects in relation to *i* and *j*, the constraint c_{ij} may be redundant. In such a situation the position of object *j* in relation to object *i* is determined by other intervening objects, and two of the c_{ij}^{α} constraints that are adjacent in Figure 8(a), for example c_{ij}^{r} and c_{ij}^{t}, can both be satisfied by the same placement (in this example object *j* would be forced to lie *both* above and to the right of object *i* by other, unspecified, intervening objects.)

In order to extend the two-dimensional compaction model discussed above into a complete model for placement, we must allow objects to be rotated/mirrored and reconfigured as well. Our complete model for transistor-level placement thus consists of four separate steps. First, if a placement object possesses multiple configurations, one configuration must be selected for placement. Second, each object must be assigned an orientation in the layout plane. Third, each object must be assigned a placement relative to the others. Finally, the absolute placement of each object must be determined—the symbolic placement must be mapped into its corresponding direct physical placement—so that the cost function can be evaluated. We discuss each of these steps in more detail below.

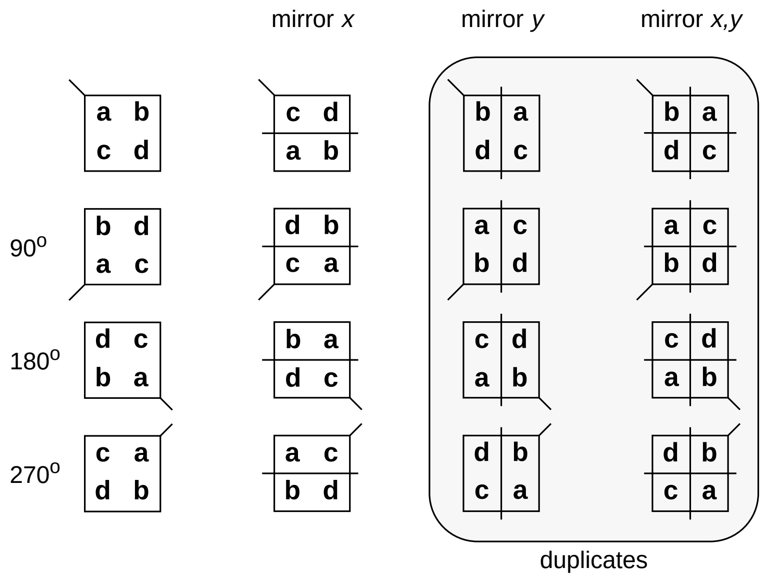


Figure 10: The possible orientations of an arbitrary block which can be obtained through mirroring and rotating. Eight of the sixteen resulting orientations are duplicates.

2.4.1 Configuration and Orientation Selection

As shown in Figure 10, an arbitrary planar object can have one of eight possible **orientations** if the object is allowed to be rotated and mirrored about its two axes. In addition, if the object has more than one **configuration** (such as a bipolar transistor with more than one aspect ratio), each of these configurations can take on up to eight orientations. In Figure 9 we show that one configuration of a CMOS FET, due to symmetry across its axis perpendicular to the gate, can take on one of only four orientations. The first and second aspects of the placement problem listed above, the selection of one of the possible configurations, and the assignment of an orientation to that configuration, can be modeled by appropriate decision variables. We define d_i to be a decision variable representing object *i* that takes on a value from the set

$$\left\{ d_i \middle| d_i \in (d_i^1, d_i^2, \ldots, d_i^n) \right\} \qquad (3)$$

where *n* is the number of members in the cross product of the set of possible orientations and possible configurations. This aspect of the problem requires that an optimal assignment be made to the decision variable representing each *individual* object in the design.

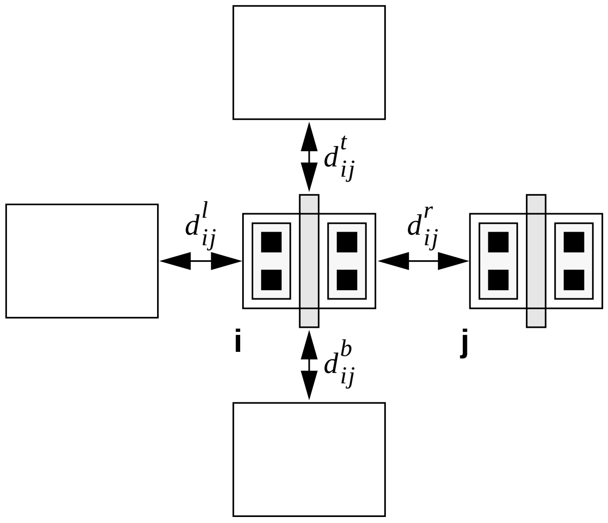


Figure 11: The four possible relative placement of two FET objects i and j and the corresponding decision variable.

2.4.2 Relative Object Placement

The third aspect of the floorplanning problem, assigning relative placements to the selected objects, can also be modeled as a decision problem. Given any pair of objects from the layout, these objects can take one of four positions relative to each other as shown in Figure 11: object *j* is constrained to be either above, below, to the left, or right of object *i*. In order to maintain the correct design rule distance between the two objects, at least one of these constraints must be enforced. We consider d_{ij} to be a decision variable representing the pair of objects *i* and *j* which can take on a value from the set

$$\left\{ d_{ij} \middle| d_{ij} \in (d_{ij}^t, d_{ij}^l, d_{ij}^b, d_{ij}^r) \right\} \tag{4}$$

This aspect of the problem requires that an optimal assignment be made to the decision variable representing each *pair* of objects in the design.

2.4.3 The Pairwise Relative Placement Problem

All three aspects of the placement problem discussed above—object configuration selection, orientation selection, and relative position assignment—can be modeled as the assignment of integer values to a set of decision variables. Thus, all three of these steps can be approached as a unified search problem, which we refer to as the **pairwise relative placement** problem. In this formulation it is easy to see that for a problem with *n* placable objects, there will be a total of

$$\binom{n}{2} + n = n^2 \tag{5}$$

integer decision variables. In the simplest case (no alternate object configurations) each of these decision variables can take on one of 4 possible values, yielding a search space whose size is

$$4^{n^2} \tag{6}$$

The assignment of a value to each of the decision variables has the effect of selecting a design rule that will be enforced between each pair of objects. The design rules can be written in the traditional form of layout compaction constraints [115] as follows

$$\begin{aligned} x_j - x_i &\geq \delta_{ij}^r - \overline{(d_i^\alpha \wedge d_j^\beta \wedge d_{ij}^r)} \cdot K \\ x_i - x_j &\geq \delta_{ij}^l - \overline{(d_i^\alpha \wedge d_j^\beta \wedge d_{ij}^l)} \cdot K \\ y_j - y_i &\geq \delta_{ij}^t - \overline{(d_i^\alpha \wedge d_j^\beta \wedge d_{ij}^t)} \cdot K \\ y_i - y_j &\geq \delta_{ij}^b - \overline{(d_i^\alpha \wedge d_j^\beta \wedge d_{ij}^b)} \cdot K \end{aligned} \tag{7}$$

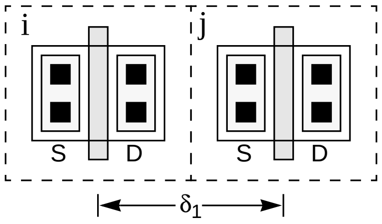
for all $i, j \in 1 \ldots n$, in which $d_i^\alpha \in (d_i^1, d_i^2, \ldots, d_i^n)$, $d_j^\beta \in (d_i^1, d_i^2, \ldots, d_i^n)$, $\delta_{ij} \geq 0$ is an integer constant representing the appropriate design rule for the specified object configurations, and *K* is an integer constant chosen to be larger than the largest δ_{ij} found in the set of design rules. The term in parenthesis which is multiplied by *K* forms a logical predicate that serves to select exactly one of the constraints for each pair of objects.

In Figure 12 we show several of the relative placements that are possible for a pair of MOSFET transistors, along with the design rule constraint which results from each placement. As in Figure 9, each transistor has one of four possible configurations, so d_i, d_j, and d_{ij} can each be represented by a two-bit decision variable. There are therefore $2^6 = 64$ different pairwise placements for two objects.

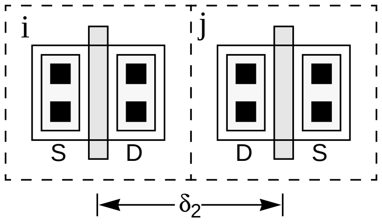
2.4.4 Solving the Compaction Constraint Graphs

In order to evaluate the cost function (discussed in Section 2.4.6) for each candidate placement, absolute physical coordinates for each object must be found, meaning that the indirect pairwise placement constraints must be solved. This is itself an optimization problem which involves

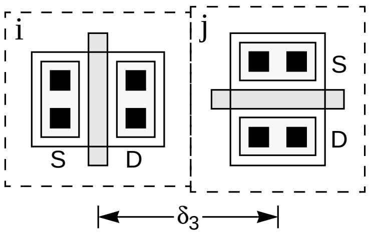
$$x_j - x_i \geq \delta_1 - \overline{(d_i^1 \wedge d_j^1 \wedge d_{ij}^r)} \cdot K$$



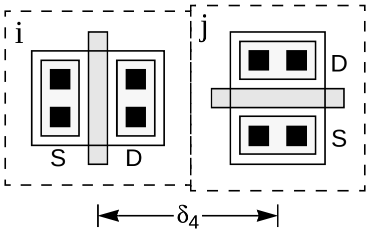
$$x_j - x_i \geq \delta_2 - \overline{(d_i^1 \wedge d_j^2 \wedge d_{ij}^r)} \cdot K$$



$$x_j - x_i \geq \delta_3 - \overline{(d_i^1 \wedge d_j^3 \wedge d_{ij}^r)} \cdot K$$



$$x_j - x_i \geq \delta_4 - \overline{(d_i^1 \wedge d_j^4 \wedge d_{ij}^r)} \cdot K$$



⋮ etc.

$$y_i - y_j \geq \delta_{64} - \overline{(d_i^4 \wedge d_j^4 \wedge d_{ij}^t)} \cdot K$$

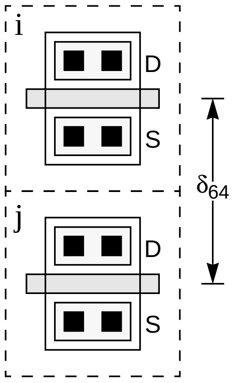


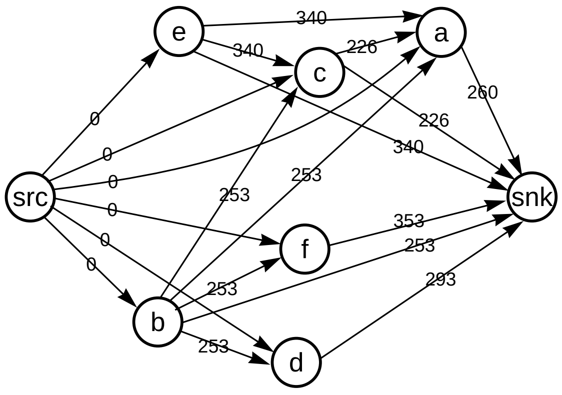
Figure 12: The 64 possible placements for a pair of CMOS transistors (right) and the corresponding design rule constraints (left). *K* is a suitably large number.

locating values for all x_i and y_i variables which satisfy all of the constraints given by Equation 7, given a unique assignment to all of the decision variables d_i and d_{ij}, while minimizing the placement area. However, these constraints are linear inequalities of a special form—they are all two-variable difference constraints. Optimization problems of this type are referred to as **network flow problems**, and they can be solved using straightforward graph techniques.

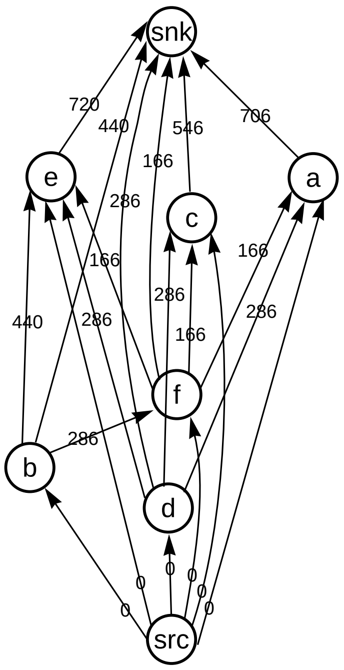
To solve the constraint graph network flow problem we construct two directed acyclic graphs (DAGs), called **constraint graphs,** as follows: One graph represents all of the constraints involving the *x* variables, and the second graph represents all of the constraints involving the *y* variables. The vertices represent the object coordinate variables (either *x* or *y*), and the edges represent a constraint involving the variables at the two endpoints. By convention, we define the object location as the coordinate of its lower-left corner. Each edge is given a positive weight corresponding to the sum of the design rule, δ_{ij}, involved in the constraint and the width or height of the source vertex object. Each graph is given two artificial vertices, named **source** and **sink**. An edge of zero weight is drawn from the source to every other vertex. An edge from each object is drawn to the sink vertex with a weight corresponding to the width/height of the object.

Solving the constraints in each direction is performed by finding the length of the longest path, $D[v]$, from the source to every other vertex v, the so-called single-source longest-paths (SSLP) problem. The longest path from the source vertex to the sink vertex in each graph is called the **critical path**, and represents the minimum size of the layout in that dimension (either *x* or *y*.) Since it is possible to encounter positive weight cycles in these graphs (we will explain why in Section 2.4.5) the well known Bellman Ford algorithm [20, p. 532] provides an appropriate method for finding the longest path. An example of two constraint graphs, and their corresponding minimal rectangle packing, is shown in Figure 13.

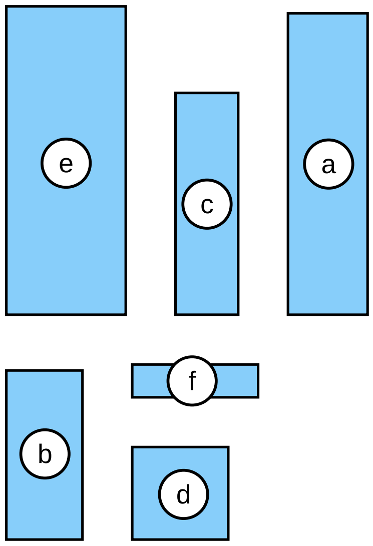
The computational complexity of the Bellman Ford algorithm is $O(ev)$ where *e* is the number of edges and *v* is the number of vertices in the graph *G*. In our case $e \cong v^2/2$ (there will be an edge between every pair of vertices in one of the two constraint graphs) so the complexity is $O(v^3)$. For special cases of the input graph, more efficient algorithms are possible. Gabow and Tarjan [29] have developed an algorithm that runs in $O(\sqrt{V} \cdot E \cdot \log(VW))$ where W is the largest-magnitude weight of any edge in the graph. This latter algorithm will be more efficient that Bellman-Ford when the variation in the edge weights is relatively small (as is often the case in design rule constraints).



(a) X-axis constraint graph



(b) Y-axis constraint graph



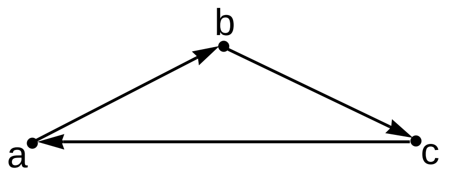
(c) the resulting physical placement

Figure 13: An example of two constraint graphs and their associated physical packing

If we run the Bellman-Ford algorithm twice on a graph, first starting at the source node and a second time from the sink node and with the directed edges reversed, we gather additional useful information about the longest-path solution. For all nodes v_i along the critical path, the values of $D[v_i]$ resulting from the two invocations, call them $D[v_i]_1$ and $D[v_i]_2$, will be identical. For all nodes not on the critical path we can define a value called $\text{slack}[v_i] = D[v_i]_1 - D[v_i]_2$

$$(d^r_{ab} \wedge d^r_{bc} \wedge d^l_{ac})$$

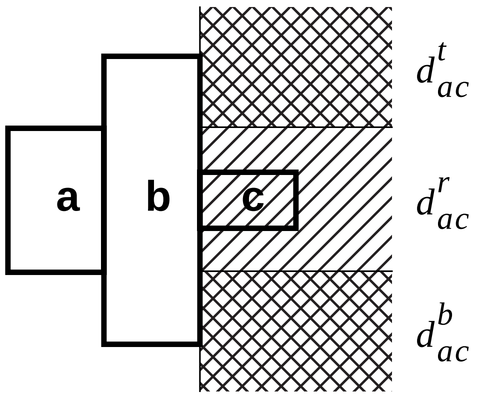
(a) a relative position assignment for three objects



(b) the resulting constraint graph

$$(d^r_{ab} \wedge d^r_{bc} \rightarrow d^r_{ac})$$

(c) the transitivity relationship expressed as an implication



(d) a physical justification for the implication

Figure 14: An example of the transitivity relationship between design rule constraints.

which represents the total range of motion that each object has in the minimum sized layout. Each layout object could be positioned anywhere along this continuum of positions and the graph solution will remain minimal. These slack values are often used to optimize second-order criteria in the placement. For example, objects can be placed so as to minimize the total length of wiring, or the lengths of wires between certain critical nodes, in the cell [39,55,99].

2.4.5 Constraint Graph Cycles and Infeasibility

As we alluded in Section 2.4.4, there is one property of the two-dimensional compaction based placement model which we feel is undesirable. It is possible for a particular assignment to the d_{ij} decision variables to result in an infeasible placement. This will manifest itself as a cycle in one of the two constraint graphs, and is a result of an assignment that violates a transitivity relationship between two or more constraints.

An example of this is shown in Figure 14(a). This example shows an assignment to the relative position decision variables of three objects that results in a constraint graph with a cycle, as shown in Figure 14(b). It is impossible to locate the longest path through this graph (it is infinite) and the Bellman-Ford algorithm would terminate unsuccessfully.

In Figure 14(c) we show that for this example of three objects, given assignments to the first two relative position decision variables of d_{ij}^{r}, we can make the statement that the third decision variable is **implied** to be d_{ij}^{r} as well. As pointed out in [84], one can make the weaker argument that the selection of the assignment d_{ab}^{r} and d_{bc}^{r} implies simply that $d_{ac} \neq d_{ac}^{l}$, however, as shown in Figure 14(d) the stronger implication $d_{ac} = d_{ac}^{r}$ can be made. This is because the assignments

$$d_{ac} = \left\{ d_{ac}^{t}, d_{ac}^{b} \right\} \tag{8}$$

while they are valid assignments, only serve to limit the feasible positions of object C to *either* the upper or lower crosshatched regions, respectively. With the assignment $d_{ac} = d_{ac}^{r}$, *both* of these regions result in feasible solutions, as well as the center crosshatched region as well.

This implication defines a **transitivity** relationship among the constraint assignments, and which can be extended to any sequence of decision assignments which we call a **closed constraint cycle**:

$$d_{12}^{\alpha} \wedge d_{23}^{\alpha} \wedge \ldots \wedge d_{nm}^{\alpha} \rightarrow d_{1m}^{\alpha} \tag{9}$$

where $\alpha \in \{r, l, t, b\}$ and $n \geq 2$.

One can choose to detect transitivity violations, and therefore infeasible solutions, using the Bellman-Ford algorithm at the time when the constraint graphs are solved, or, as in [84], one can detect the presence of closed constraint cycles and use them to make assignment *implications*. The latter approach will result in a smaller search space, so it may be desirable if the detection of closed constraint cycles is computationally efficient. However, a recently developed symbolic representation of the two-dimensional compaction constraints, called the sequence pair [77] representation, solves this problem much more neatly. The sequence pair will be the subject of Chapter 3.

2.4.6 The Cost function

After the constraint graph has been solved we can compute the cost of the resulting placement. It is desirable that the cost function be relatively easy to compute, as its solution must be found in the inner loop of the optimization routine. Up until this point we have been discussing the general problem of rectilinear block placement without regard to routing effects. Essentially, we

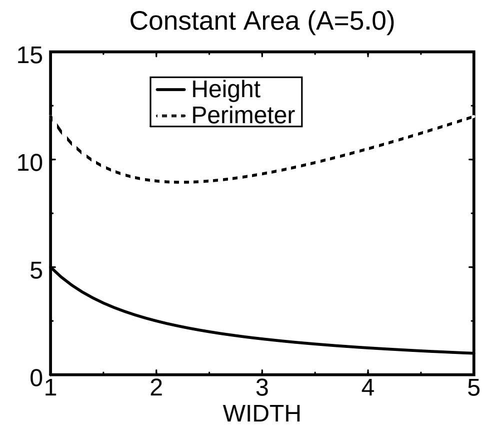


Figure 15: The cell height and perimeter as a function of its width for a constant area

have been solving the rectangle packing problem. In problems related to integrated circuit placement it is most often the case that, because of the complexity of the routing problem itself, detailed routing is performed as a separate step after the placement has been completed. Thus, the placement problem must be solved using, at best, relatively simple estimates for the effects of routing. We most often see these routing estimates appearing in the placement model as a component of the cost function. Therefore, the placement cost function can be written as some function of both the cost of the placement itself and an estimate of the routing cost:

$$\text{cost} = f\,(\text{placement, routing}) \tag{10}$$

In the remainder of this section we will discuss the more common methods of estimating these two terms of the cost function.

Placement Cost

The two most common placement cost functions are probably the area of the bounding box of the cell

$$\text{area} = \text{height} \times \text{width} \tag{11}$$

and half of the cell perimeter.

$$\frac{\text{perimeter}}{2} = \text{height} + \text{width} \tag{12}$$

Both are non-decreasing functions, so they are essentially equivalent. However, for sets of alternative cells with the same area, the use of the perimeter function tends to favor cells with a more square aspect ratio. As can be seen in Figure 15, there are an infinite number of minimum area rectangles over a continuous range of aspect ratios. However, the rectangle with the minimum perimeter corresponds to the single minimum area rectangle with an aspect ration of 1.0. Also, note that the perimeter function is linear while the area function is quadratic. In our linear programming formulation of the problem, described in Section 3.3.1, this property will be important.

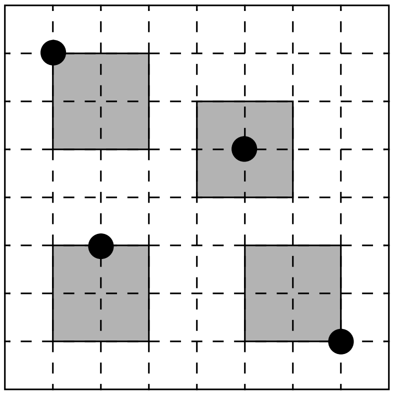
Routing Cost

As we mentioned above, the detailed routing problem itself is extremely difficult to solve and it is impractical to embed routing within the inner loop of the placement step. Therefore we must make do with an estimate for the routing component of the placement cost function. This estimate must be computationally efficient to evaluate, but it must also be accurate enough to act as a reasonable predictor for the actual routing cost of the current placement.

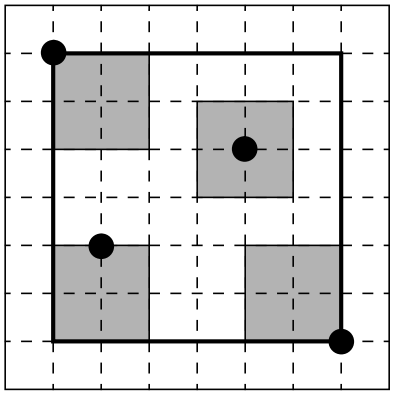
For the moment we will make no assumptions about the nature of the blocks being placed (i.e. whether they are rectangles, macro-blocks or transistors.) We will simply assume that the problem being solved contains a number of electrical nets, possibly multi-terminal, which must be connected at the conclusion of the placement step. The blocks may contain terminals, either on their periphery or internal to the block, which specify the connection points for the nets. The problem is to estimate the actual minimum cost of connecting together, in a design-rule-correct manner, all of the terminals of each net.

In Figure 16 we show an example of a simple multi-terminal net routing problem. Figure 16(a) shows a set of four blocks, each containing one terminal of a multi-terminal net. Perhaps the simplest estimate of the routing cost is to find 1/2 of the perimeter of the smallest rectangle which encloses all of the sinks, the so-called **bounding box** of the net, as shown in Figure 16(b). The net bounding box is simple to compute: it can be found for all nets in $O(n)$ time, where n is the total number of net terminals in the problem. However, this estimate is also fairly crude. It gives a lower bound on the routing cost which is exact for two-terminal nets, but as the number of terminals increases the quality of this bound becomes extremely poor.

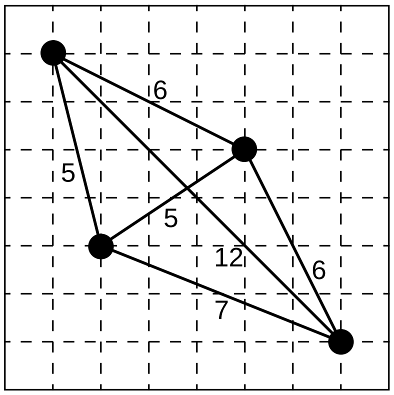
The most accurate routing estimates will be obtained with a model which more closely matches the morphology used by the actual detailed router. Most integrated circuit detailed routers will attempt to connect multi-terminal nets using a tree-like structure, and therefore the routing estimate should make the same assumption. We will discuss two classic models for routing estimation based on tree structures: Minimum Spanning Trees and Rectilinear Steiner Minimum Trees.



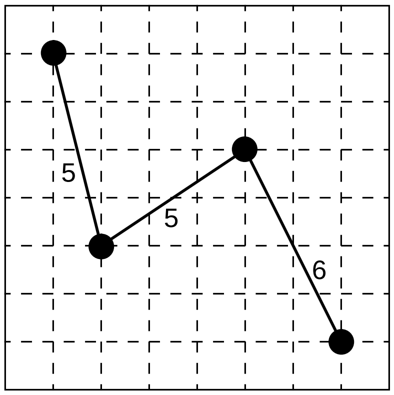
(a) a set of four blocks, and the sinks of a multi-terminal net



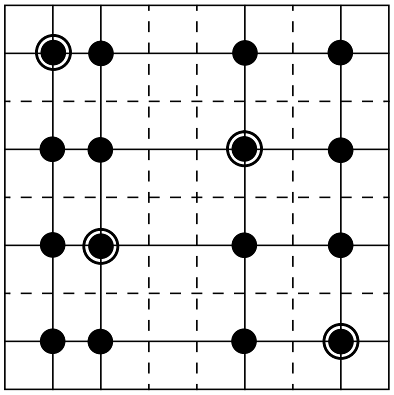
(b) the perimeter of the smallest enclosing rectangle (cost = 12)



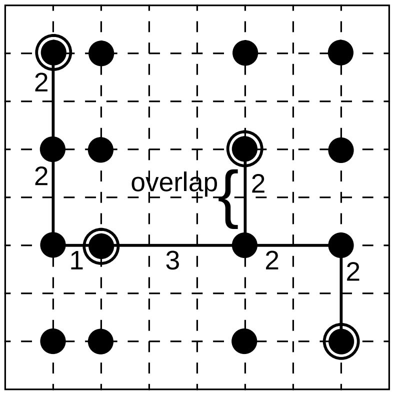
(c) a clique representing the multi-terminal net



(d) a Minimum Spanning Tree on the graph (cost = 16)



(e) the grid graph with steiner points and demand points



(f) a Rectilinear Steiner Tree on the grid graph (cost = 14)

Figure 16: An example illustrating several different methods for routing cost estimation.

Both are derived from classic problems in graph theory.

In general, the **Minimum Spanning Tree** (MST) problem is defined as the following selection problem [106 p. 76]. Given a graph $G = (V, E)$ with an edge weight function $w : E \rightarrow \mathbf{R}$, select a subset of the edges $E' \subseteq E$ such that E' induces a tree on G and the total cost over all edges in E'

$$\text{cost} = \sum_{e \in E'} w(e)$$

is minimum over all such trees.

To model the integrated circuit routing problem we construct a graph G_i to represent each net *i*. There is one vertex in graph G_i for each terminal of net *i*, and an edge between each pair of vertices which is weighted by the rectilinear distance between them (thus the graph is a clique.) The MST is therefore the minimum-weight set of rectilinear net segments which can be used to ensure that the graph is connected. Figure 16(c) shows the weighted graph used to represent the problem of Figure 16(a), while Figure 16(d) shows the MST solution for the net.

Conveniently, the MST problem can be solved optimally using a simple greedy algorithm. There are two well-known algorithms for its solution, Kruskal’s algorithm [54] and Prim’s algorithm [88]. Below we repeat the pseudo-code for Kruskal’s algorithm due to Corman et al [20, p. 505].

```
MST-KRUSKAL(G, w)
    $A \leftarrow \emptyset$
    for each vertex $v \in V[G]$
        MAKE-SET(v)
    sort the edges of E by non-decreasing weight w
    for each edge $(u, v) \in E$, in order
        if FIND-SET(u) ≠ FIND-SET(v)
            $A \leftarrow A \cup \{(u, v)\}$
            UNION(u, v)
    return A
```

Kruskal’s algorithm operates by greedily traversing the edges in order of non-decreasing weight. If the current edge connects two disjoint subsets of the vertices the edge is retained in the MST, otherwise it is discarded. The run time of Kruskal’s algorithm depends on the implementation of the set union operation. Corman et al discuss an implementation with an asymptotic run time of $O(E \cdot \log E)$. Prim’s algorithm, on the other hand, uses a relaxation approach based on a priority queue. The run time of Prim's algorithm depends on the implementation of the priority queue, and Corman et al [20, p. 509] discuss an implementation with an asymptotic run time of $O(E \cdot \log V)$.

The Rectilinear Steiner Minimum Tree problem is based on the more general Steiner Minimum Tree (SMT) problem of graph theory. The most general definition of the SMT problem is as follows [106, p. 81]. Given a graph $G = (V, E)$ with an edge weight function $w : E \rightarrow \mathbf{R}$ and a subset of the vertices $D \subseteq V$, select a subset of the vertices $V' \subseteq V$ such that $D \subseteq V'$ and V' induces an MST of minimum cost over all such trees. The vertices in the subset D are referred to as the **demand points**, while the vertices in the subset $V' - D$ are referred to as the **Steiner points**. If $D = V$ then the problem is equivalent to the MST problem, but if $D \subset V$ then we are given a problem with some optional vertices which can be used to locate lower cost solutions.

The RSMT problem is a subset of the SMT problem in which the edges are restricted to rectilinear shapes, and provides the correct model for manhattan detailed routing. In Figure 16(e) we show the application of RSMT optimization to our routing problem from Figure 16(a). The vertices are embedded in the cartesian plane so that each vertex's coordinates correspond to its location in the placement. We project horizontal and vertical grid lines from each vertex. This graph is called the **grid-graph**—the original terminals represent the *demand* points, while *Steiner* points are placed at every remaining coordinate at which these grid lines intersect. It is easy to see how the RSMT formulation provides the opportunity to reduce the routing cost from the solution of the MST formulation. Route segments are no longer required to terminate at the terminals (demand points): they can now terminate when they intersect an intermediate point (Steiner point) on a pre-existing route segment.

Figure 16(f) shows the RSMT embedded in the grid graph of Figure 16(e). It is apparent that the RSMT represents an exact lower bound on the routing cost for this multi-terminal net as long as blockages in the placement are ignored. The RSMT solution has a cost of 14 while the MST was slightly more pessimistic with a cost of 16. The bounding box solution gave an estimate of 12, which is clearly optimistic. Clearly, among this set of options, the RSMT model provides the most accurate lower bound on the routing cost. Unfortunately, the RSMT optimization problem is NP-complete [30], and for this reason it is most common to use and MST routing mode, or to make use of an RSMT formulation with a heuristic optimization algorithm.

The most common heuristic algorithms for RSMT optimization operate through greedy modifications to an initial MST routing solution. Because the MST cost metric is evaluated using rectilinear coordinates, it is trivial to map an MST solution onto an equivalent RSMT solution. Local modifications can then be used which may reduce the cost of the RSMT solution.

An examination of Figure 16(f) will demonstrate that it is equivalent to the MST solution in Figure 16(d). The reduction of two from the manhattan routing cost estimate of the MST solution was obtained because the RSMT recognized that two routing segments overlap at the position marked in the figure. Hwang [45] used a geometrical argument based on this observation to prove the following relationship

$$\frac{\text{Cost(RSMT)}}{\text{Cost(MST)}} \le \frac{2}{3} \tag{13}$$

This relationship can be used to provide tight bounds on the quality of any RSMT solutions obtained with the use of these heuristic algorithms.

One final technique for routing cost estimation is worth mentioning. As part of a sophisticated supply vs. demand routing estimation scheme named RISA, which is intended for large standard-cell and sea-of-gates designs, Cheng [14] introduced an extremely fast statistical method for the estimation of the RSMT cost of a multi-terminal net. By analyzing the optimal RSMT solutions for a large number of randomly generated routing problems, Cheng was able to make a statistical argument concerning the average number of horizontal and vertical tracks required by a RSMT as a function of the number of pins on the net.

2.5 Summary

The subject of this chapter was the modeling of the abstract placement problem. The specific nature of the objects being placed was temporarily ignored, and the problem formulated as the placement of abstract rectangles. We discussed the distinction between direct and indirect placement models, providing examples of both. Our main goals in selecting a placement modeling environment—efficient algorithmic traversal of the search space and inexpensive cost function calculation—appear to favor the use of indirect *symbolic* models. The primary motivation behind this choice is the fact that there is a many-to-one mapping between the direct placement search space and the set of feasible symbolic placements: When compacted and legalized, many direct placements will map to the same symbolic placement. It is our opinion that working in the symbolic placement space will lead to more efficient search of the legal placements.

In order to avoid the primary problem with the popular Slicing Tree symbolic model we have outlined an alternative symbolic placement model based on two-dimensional compaction constraints. A unique placement is defined by the assignment of an *orientation* to each object in the problem, and the assignment of a *relative placement* to each pair of objects in the layout. We refer to this assignment problem as the **pairwise relative placement** problem. After a complete assignment is made to all problem variables, the absolute placement of each object can be determined from the solution of a related pair of constraint graphs. This graph problem can be solved with the use of the well known Bellman-Ford algorithm.

One problem with the pairwise relative placement model is the fact that some assignments result in infeasible placements. These occur when an assignment violates the requirement for transitivity along chains of object pair assignments. The Bellman-Ford algorithm is capable of detecting infeasibility as it results in positive-weight cycles in the constraint graphs. However, we would prefer to avoid making these infeasible assignments in the first place. This shortcoming will be addressed in Chapter 3 which reviews a recently developed notation, named the sequence pair, for the representation of the two-dimensional compaction constraints.

The subject of routing was introduced in this chapter in the context of calculating the placement cost. Routing is modeled by including abstract multi-terminal nets, represented as sets of pins on their associated objects, which must be connected with minimum cost. Several models for routing cost were reviewed, including the area of the minimum bounding box, minimum spanning trees, and Steiner trees. Of these models the net bounding box cost is the most inexpensive to calculate, but it provides only a very loose lower bound on the true routing cost. The Rectilinear Minimum Steiner Tree routing model provides an exact lower bound and therefore the greatest accuracy when modeling multi-terminal manhattan style routing nets. However, the general RMST routing problem is NP-complete. Minimum Spanning Trees, and approximate RMST solutions heuristically derived from MSTs, often provide an acceptable compromise.

CHAPTER 3
Placement Using the Sequence Pair Representation

In this chapter we discuss a relatively new symbolic placement model called the sequence pair, first introduced by Murata et al [77]. We show how this representation simplifies the two dimensional placement model introduced in Chapter 2, and present three alternative formulations of this model: an integer linear programming formulation, a branch-and-bound formulation, and a simulated annealing formulation. Several experiments are presented which demonstrate the relative merits of the branch-and-bound and simulated annealing implementations.

3.1 The Sequence Pair Representation

The sequence pair is a symbolic representation of two-dimensional compaction constraints introduced by Murata et al [77]. This notation provides a number of advantages over representing the constraints in their raw form. sequence pairs are convenient to manipulate, lending themselves naturally to move-based stochastic optimization algorithms, such as Simulated Annealing and Genetic Algorithms, as well as exhaustive enumeration algorithms such as Branch-and-Bound. Most importantly, however, the sequence pair model implicitly enforces the transitivity relationship between the design constraints, preventing a priori the enumeration of constraint combinations that are infeasible, and thus neatly solving the constraint graph infeasibility problem discussed in Section 2.4.5. Through the elimination of all infeasible placements the size of the search space is reduced significantly. In addition, the lack of cycles in the resulting constraint graphs allow more efficient algorithms to be used for their solution.

In the remainder of this section we review the sequence pair notation and elaborate its advantages. We then discuss several optimization algorithms based on the sequence pair notation. Two formulations for obtaining the exact solution are presented. In Section 3.3.1 we propose an optimization algorithm based on an Integer Linear Programming formulation of the sequence pair notation. In Section 3.3.2 we propose an exhaustive enumeration search algorithm based on the branch-and-bound technique. A heuristic algorithm based on simulated annealing is discussed in Section 3.3.3.

In [77] Murata et al introduced the sequence pair notation to represent the two-dimensional rectangle packing problem, defined as follows:

Definition 3-1: Rectangle Packing Problem **RP.** Let *M* be a set of *m* rectangular objects whose height and width are given in real numbers. (Orientation is fixed.) A packing of *M* is a non-overlapping placement of the objects. The minimum bounding rectangle of a packing is called the chip. Find a packing of *M* in a chip of minimum area.

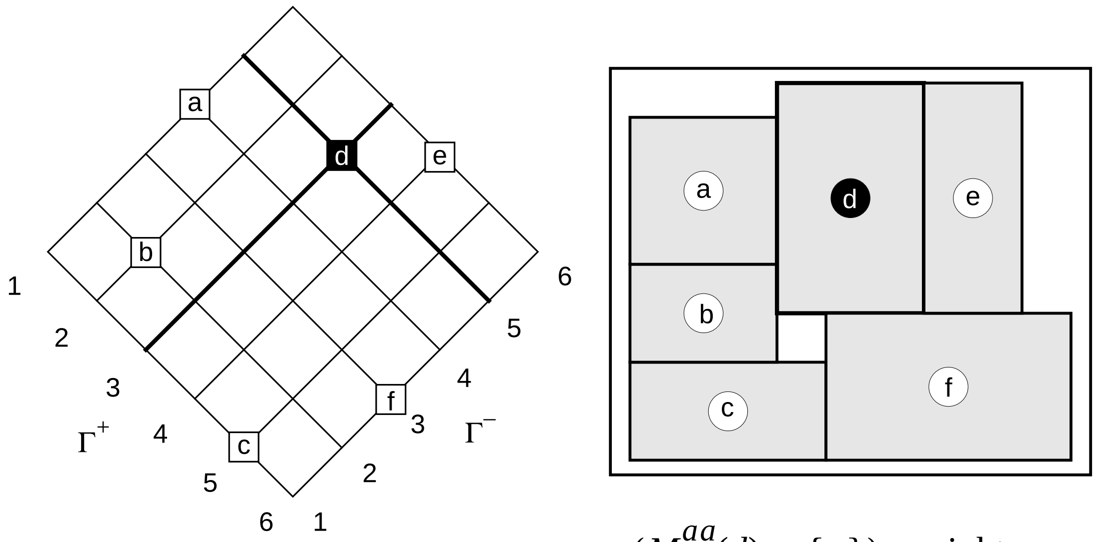
The authors point out that if the location of a rectangle is defined to be a real number, the number of possible placements is infinite. Even if the location coordinates, width, and height of the objects are restricted to the integers, and even if the solution space is restricted to *feasible* solutions in which the rectangles do not overlap, the space is still infinite. They seek a way of expressing the solution space in a symbolic manner so that the size of the solution space is finite, but it is still *guaranteed to contain the optimal solution*. Put more formally, they seek a means of grouping the infinite solution space into a finite number of codes, each code representing a family of solutions which are in some sense equivalent (i.e. an equivalence class.) A solution space which meets this property is called **P-admissible**, and defined as follows:

1. The solution space is finite
2. Every solution is feasible
3. Evaluation of each code is possible in polynomial time and so is the realization of the corresponding packing
4. The packing corresponding to the best evaluated code in the space coincides with an optimal solution of **RP**.

They propose a P-admissible solution space based on the two-dimensional compaction constraints introduced by Schlag et al [100], which we discussed in Section 2.4. A sequence pair is a symbolic representation of this solution space which specifies which of the four relative position constraints is enforced for each pair of objects.

A sequence pair is a one-to-one mapping from a pair of sequences, (Γ^+, Γ^-),

$$\begin{aligned} \Gamma^+ &= (\rho_1 \rho_2 \ldots \rho_m) \\ \Gamma^- &= (\eta_1 \eta_2 \ldots \eta_m) \end{aligned} \tag{14}$$



$$(\Gamma^+, \Gamma^-) = (\text{abdecf, cbfade})$$

$$(M^{aa}(d) = \{e\}) \rightarrow \text{right}$$
$$(M^{bb}(d) = \{a, b\}) \rightarrow \text{left}$$
$$(M^{ba}(d) = \varnothing) \rightarrow \text{above}$$
$$(M^{ab}(d) = \{c, f\}) \rightarrow \text{below}$$

Figure 17: The symbolic sequence pair placement representation

to a rectangle packing, where $\rho_i, \eta_i \in \{1 \ldots m\}$. The correspondence between a sequence pair and its implied packing is best visualized with an example. Figure 17(a) shows an abstract representation for the sequence pair $(\Gamma^+, \Gamma^-) = (\text{abdecf, cbfade})$. It consists of an oblique grid whose axes are labeled with positions in Γ^+ increasing left-to-right on the positive slope grid lines, and positions in Γ^- increasing left-to-right on the negative slope grid lines. Since each of the sequences define a total ordering on the objects, each object is placed on the grid at a unique positive grid line and a unique negative grid line. The location of an object on this grid should not be confused with an object's physical location in the packing; it is only a symbolic representation of the relative positions of two objects, defined as follows.

For an object x, the sequence pair admits the definition of the following four sets

$$\begin{aligned} M^{aa}(x) &= \{x' \mid x' \text{ is after } x \text{ in both } \Gamma^+ \text{ and } \Gamma^-\} \\ M^{bb}(x) &= \{x' \mid x' \text{ is before } x \text{ in both } \Gamma^+ \text{ and } \Gamma^-\} \\ M^{ba}(x) &= \{x' \mid x' \text{ is before } x \text{ in } \Gamma^+ \text{ and after } x' \text{ in } \Gamma^-\} \\ M^{ab}(x) &= \{x' \mid x' \text{ is after } x \text{ in } \Gamma^+ \text{ and before } x' \text{ in } \Gamma^-\} \end{aligned} \qquad (15)$$

Membership in each of these four sets determines the relative positions for each pair of objects which, in turn, identifies the Schlag-style compaction constraint that must be enforced between them. Specifically:

- If $x' \in M^{aa}$ then x' is *right* of x in the packing. In our notation, $d_{xx'} = d^r_{xx'}$.
- If $x' \in M^{bb}$ then x' is *left* of x in the packing. In our notation, $d_{xx'} = d^l_{xx'}$.
- If $x' \in M^{ba}$ then x' is *above* x in the packing. In our notation, $d_{xx'} = d^t_{xx'}$.
- If $x' \in M^{ab}$ then x' is *below* x in the packing. In our notation, $d_{xx'} = d^b_{xx'}$.

On the abstract grid of Figure 17(a) these four rules can be visualized as follows. The two grid lines that intersect at the location of a given object x divide the abstract plane into four disjoint regions, corresponding to the four sets in Equation 15. Every other object x' must be in one of these four regions. The packing which results from this sequence pair, after the solution of the resulting *x*-axis and *y*-axis constraint graphs, and taking into account the dimensions of each object, is shown in Figure 17(b).

The derivation of the reverse mapping process, which provides a unique sequence pair representation for a particular rectangle packing, is somewhat involved, and is described by the authors [77, page 473]. The authors provide detailed proofs that both mappings are one-to-one. The fact that both of these two mappings are one-to-one demonstrates that there is a unique sequence pair representation for any valid packing, and in turn a unique packing for every sequence pair. This, along with the fact that the resulting packing can be found in polynomial time, permits the authors to prove that the sequence pair notation describes a P-admissible solution space.

The solution space described by a sequence pair of *m* objects consists of all possible permutations of the two sequences Γ^+ and Γ^-. Thus the size of the solution space is $(m!)^2$. Clearly this problem is non-polynomial in complexity, and it is straightforward to show that the resulting

$$(\mathrm{ab}\boxed{\mathrm{de}}\mathrm{cf},\ \mathrm{cbfade}) \rightarrow (\mathrm{ab}\boxed{\mathrm{ed}}\mathrm{cf},\ \mathrm{cbfade})$$

$$\mathrm{e} \in M^{aa}(\mathrm{d}) \rightarrow \mathrm{e} \in M^{ba}(\mathrm{d})$$

$$d^{r}_{de} \rightarrow d^{t}_{de}$$

Figure 18: The effect of a single nearest-neighbor pair swap in a sequence pair

optimization problem will be NP-complete [30]. Nevertheless, this is an improvement over the size of the solution space implied by the explicit two-dimensional compaction constraints, which we showed is 4^{m^2} (see Equation 6 on Page 36.) This reduction in the size of the search space is a direct result of the elimination of the infeasible placements.

Starting with the sequence pair representation there are many ways to formulate the problem of finding the optimum rectangle packing. In the following sections we describe several alternate formulations of this optimization problem: a simulated annealing formulation, an integer linear programming formulation, and a branch-and-bound formulation.

There is one property of the sequence pair which we will find especially useful in all of these formulations. As can be seen in Figure 18, a simple “move” in the sequence pair solution space can be performed to change from one packing to another. If a pair of objects are selected such that they are adjacent in one of the two sequences, and their order in that sequence is reversed, only a single constraint changes in the two resulting constraint graphs. It is straightforward to show that a single constraint will be deleted from one constraint graph (either the *x*-axis graph or the *y*-axis graph) and added to the other graph. As we will show in Section 3.2, this change can be propagated through the two constraint graphs in an incremental manner, allowing the longest-path problem solution to be performed slightly more quickly than in the worst case.

In the various formulations of the rectangle packing optimization problem we will traverse the search space using the following basic set of moves:

1. the swapping of two neighboring objects in one of the sequences, resulting in the deletion and addition of an edge in the constraint graphs (as discussed above)
2. the reconfiguration of an object, resulting in the modification of the weights of some of the edges in the constraint graphs
3. the addition or deletion of an object in the sequence pairs, and the addition or deletion of a vertex and its corresponding edges in the constraint graphs

3.2 Solving the Sequence Pair Constraint Graphs

It is possible to solve the *x*-axis and *y*-axis constraint graphs that are implied by a particular sequence pair, and thus obtain the resulting packing, using a single-source longest-path algorithm as descried in Section 2.4.4. As was discussed, the Bellman-Ford algorithm, which is $O(ev)$ in complexity, can be used for this task. Since there is one constraint for every pair of variables, and this constraint may appear in either the *x* or *y* constraint graphs, $e \cong v^2 / 2$ and the algorithm will have $O(n^3)$ complexity. However, since the sequence pair guarantees that every packing is feasible, the graphs are guaranteed to be acyclic, so more efficient algorithms can be used. As we will show in Section 3.2.1, a simple levelized traversal of the constraint graph is sufficient. This traversal can be performed in $O(e)$, or $O(n^2)$, time. This reduction in complexity validates our previous claim that, when compared with the use of explicit two-dimensional compaction constraints, the sequence pair provides not only a reduction in the size of the search space, but also the opportunity to make use of more efficient longest-path algorithms. In this section we review the standard algorithm for performing the single-source longest-path computation and present our own modification which allows for incremental updates (edge/vertex additions/deletions) to the graph.

3.2.1 Single Source Longest Path in Directed Acyclic Graphs

The single-source longest-path (SSLP) problem, finding the longest path in a weighted directed acyclic graph (DAG) from a given source vertex to every other vertex in the graph, is well known in the field of graph theory. In order to introduce our own modifications to this algorithm for the processing of incremental graph updates, we will review the standard algorithm here. We will be using a version of the algorithm from Cormen, Leiserson and Rivest [20][1].

A weighted directed acyclic graph *G* is represented as a set of vertices *V* and a set of edges *E*, $G = (V, E)$. Each edge $e \in E$ is an ordered pair (u, v) which indicates that there is an edge from vertex $u \in V$ (the source vertex) to vertex $v \in V$ (the sink vertex). There exists a weight function $w: E \to \Re$ mapping edges to real-valued weights. A path in the graph corresponds to an ordered sequence of edges, and the weight of a path corresponds to the sum of the weights along the edges of the path. The algorithms presented here assume an adjacency list representation of the

1. As their source these authors cite Lawler [59], who considers this algorithm to be part of the folklore.

graph. Each vertex $u \in V$ has a list $Adj[u]$ which contains pointers to all of the vertices $v \in V$ such that there is an edge (u, v) from *u* to *v*. The operators $V[G]$ and $E[G]$ return a list of the vertices and edges, respectively, in the graph. The operator $w(u, v)$ returns the weight of the edge between vertices *u* and *v*.

The algorithm `DAG-LONGEST-PATH()` is shown below. It takes three parameters, the graph *G*, the edge weights *w*, and the source vertex *s*.

```
DAG-LONGEST-PATHS($G$,$w$,$s$)
1    $finish \leftarrow$ TOPOLOGICAL-SORT($G$)
2    INITIALIZE-SINGLE-SOURCE($G$,$s$)
3    while $u \leftarrow$ LIST-POP-FRONT($finish$)
4        for each vertex $v \in Adj[u]$
5            RELAX($u$,$v$,$w$)
```

The algorithm begins by topologically sorting the vertices through the use of a depth-first traversal of the graph. It then steps through the vertices in topologically-sorted order and *relaxes* each vertex in turn. We maintain two variables for each vertex, $\Pi[v]$ containing the predecessor of vertex *v* in the longest path from the source vertex s, and $d[v]$ containing the distance from *s* to *v* along the longest path. At the conclusion of the algorithm the values of $\Pi[v]$ can be viewed as forming a longest path tree, a rooted tree containing the longest path from *s* to every other vertex that is reachable from *s*. We discuss each of the sub-procedures below.

The process of topological sorting is performed with the use of a depth-first search of *G*, performed by the procedure `DFS()`. `DFS()` is a recursive procedure, and the recursion returns from the vertices in topological order, so the central action of `DFS()` is to push the vertices onto the front of the *finish* list in the order that they return. It then returns this topologically sorted list for use by the calling procedure. The purpose of the test on line 6 of `DFS()` is ensure that disconnected graphs are fully processed.

The depth-first search begins by initializing all of the vertices. The variable *color[v]* can take on two values: WHITE indicates that the vertex has never been visited, while BLACK indicates that it has. A global variable *time* is maintained which will be used to keep track of the time step at which each vertex is visited. Two time values are maintained for each vertex, *b[v]* records the beginning time at which a vertex is first discovered, and *f[v]* returns the time that the vertex is finished. The algorithm then steps through each vertex and calls the recursive procedure `DFS-VISIT()` for each vertex that has not yet been visited. `DFS()` maintains its own predecessor

```
TOPOLOGICAL-SORT(G)
1    call DFS(G)
2    return(finish)

DFS(G)
1    for each vertex $u \in V[G]$
2        color[u] ← WHITE
3        $\pi[u] \leftarrow$ NIL
4    time ← 0
5    for each vertex $u \in V[G]$
6        if color[u] = WHITE
7            DFS-VISIT(u)

DFS-VISIT(u)
1    color[u] ← BLACK
2    b[u] ← time ← time + 1
3    for each $v \in Adj[u]$
4        if color[v] = WHITE
5            $\pi[v] \leftarrow u$
6            DFS-VISIT(v)
7    f[u] ← time ← time + 1
8    LIST-PUSH_FRONT(finish,u)
```

variable, $\pi[v]$, for every node. This keeps track of the predecessor of the vertex in the DFS traversal, and should not be confused with $\Pi[v]$, the vertex's predecessor in the longest path algorithm.

The `DFS-VISIT()` procedure begins by marking the vertex BLACK and setting its discovery time. It then recursively visits all of the vertex's unvisited neighbors, and completes by pushing the vertex onto the *finish* list and setting its finishing time. Notice that the time variable is incremented at each step, so no two starting or finishing times will have the same value.

After the graph is placed in topological order we can proceed with the calculation of the longest path, which is shown below.

During relaxation, the value of *d[v]* can be thought of as lower bound on the weight of the longest path from the source vertex to *v*. Each time that `RELAX()` is called on a vertex the lower bound is tested to see if it can be increased. Because the vertices are visited in topologically sorted order, all predecessors of a vertex will have been evaluated before the vertex itself. Therefore, at the conclusion of the procedure, *d[v]* will have been evaluated for every predecessor of v, and this lower bound will be exact. See [20] for detailed proofs of the correctness of this procedure.

```
INITIALIZE-SINGLE-SOURCE(G,s)
1    for each vertex $v \in V[G]$
2        $\Pi[v] \leftarrow \text{NIL}$
3        $d[v] \leftarrow -1$
4    $d[s] \leftarrow 0$
RELAX(u,v,w)
1    if $d[v] < d[u] + w(u,v)$
2            $d[v] \leftarrow d[u] + w(u,v)$
3            $\Pi[v] \leftarrow u$
```

The asymptotic run time behavior of the SSLP algorithm presented here is straightforward to analyze. The loops at lines 1 and 5 of DFS() are $O(V)$. The recursive procedure `DFS-VISIT()` is called exactly one for each vertex, and its inner loop at line 3 is called $|Adj[u]|$ times. Since

$$\sum_{v \in V} |Adj[u]| = E \tag{16}$$

`DFS()` is $O(V+E)$. A similar argument shows that the cost of the loop at line 3 in `DAG-LONGEST-PATHS()` is also $O(V+E)$. Since line 1 of `DAG-LONGEST-PATHS()` is $O(V+E)$, and line 2 is $O(V)$, the total running time is $O(V+E)$.

3.2.2 Single Source Longest Path with Incremental DAG Updates

As discussed in Section 3.1, the movement of objects in a sequence pair placement representation results in the insertion and deletion of a relatively small number of edges in the corresponding x-axis and y-axis constraint graphs. Reconfigurations (i.e. rotations, chainings, etc.) of objects in the placement result only in the modification of edges weights. As we will see in the branch-and-bound sequence pair optimization algorithm discussed in Section 3.3.2, there are occasions where constraint graph vertices may be inserted and deleted as well. In this section we discuss algorithms which support the incremental update of the SSLP calculations which take advantage of the fact that these edge/vertex insertions/deletions do not cause major disruption to the structure of the graph.

Recall that the `DAG-LONGEST-PATH()` algorithm presented in Section 3.2.1 consists of two major steps: a $O(V+E)$ topological sorting step followed by a $O(V+E)$ vertex relaxation step. As we will show, our incremental SSLP algorithm makes use of the fact that the addition/

deletion of edges in the constraint graphs will only invalidate the topological sorting order of vertices which are “downstream” of this change. The algorithm propagates the extent of these changes, minimizing the amount of work required to update the topological sorting solution. However, as we will show, the vertex relaxation step must still be executed in its entirety, so the algorithm still exhibits asymptotically $O(V + E)$ complexity. In fact, in the worst case it must still recompute the entire topological sorting, incurring an additional $O(V)$ time bookkeeping penalty. However, we will provide some empirical data which demonstrates an average case runtime reduction of approximately 13%.

Below we show modified versions of the `DFS()` and `DFS-VISIT()` procedures which are used to support our new incremental algorithms. The remaining procedures discussed in Section 3.2.1 can be used without modification. These two new procedures are called under the assumption that the state of some of the graph vertices will have been reset to reflect a depth first search traversal which is only partially complete. We will then discuss several new procedures which perform this state resetting, and which must be called on a graph when edges are deleted or inserted, vertices are inserted or deleted, or edge weights are changed.

There are two major changes to `DFS()` and `DFS-VISIT()`. First, a new vertex color, `GRAY`, has been added to reflect the state of a vertex which has already been discovered but has not yet been completely processed. In addition, we have added a second list, *`start`*, which records the order in which the vertices are discovered, in the same way that *`finish`* records the order in which the vertices are completed. When a change is made to the graph, the state of the vertex colors are changed to indicate which ones must be re-processed. In addition, all `GRAY` vertices are removed from the *`finish`* list, and all `WHITE` vertices are removed from both lists.

The most complex aspect of the new `DFS()` code is the new calculation of the current value of the *`time`* variable. This must pick up the time value at which the current partial traversal “left off”. Line 8 reflects the situation in which the entire graph has been reset. Line 10 reflects the condition that the finish list in empty, in which case the current time becomes equal to the time of the `GRAY` vertex which had most recently been started. Finally, lines 12 and 13 indicate the condition in which some BLACK vertices exist. In this case the maximum valid time stamp in the graph may either be on the vertex that most recently started or the one which most recently completed, so the maximum of these two is used.

Note that the test at line 15 of `DFS-INCR()` does not call DFS-VISIT-INCR() for verti-

```
DFS-INCR(G)
    for each vertex $u \in V[G]$
        if color[u] = WHITE
            $\pi[u] \leftarrow$ NIL
            b[u] $\leftarrow$ -1
        if color[u] $\neq$ BLACK
            f[u] $\leftarrow$ -1
    if LIST-EMPTY(start)
        time $\leftarrow$ 0
    else if LIST-EMPTY(finish)
        time $\leftarrow$ b[LIST-FRONT(start)]
    else
        time $\leftarrow$ MAX( f[LIST-FRONT(finish)],
                     b[LIST-FRONT(start)] )
    for each vertex $u \in V[G]$
        if color[u] $\neq$ BLACK
            DFS-VISIT-INCR(u)
```

```
DFS-VISIT-INCR(u)
    if color[u] = WHITE
        color[u] $\leftarrow$ GRAY
        b[u] $\leftarrow$ time $\leftarrow$ time + 1
        LIST-PUSH-FRONT(start,v)
    for each $v \in Adj[u]$
        if color[v] $\neq$ BLACK
            if color[v] = WHITE
                $\pi[v] \leftarrow u$
            DFS-VISIT(v)
    color[u] $\leftarrow$ BLACK
    f[u] $\leftarrow$ time $\leftarrow$ time + 1
    LIST-PUSH-FRONT(finish,u)
```

ces which have already finished. It is this pruning of the depth first traversal which benefits from the incremental update. If an incremental change to the graph leaves most of the vertices colored BLACK, a significant amount of time will be saved. However, this time savings must exceed the extra processing which must be performed in order to reset the state of the affected vertices. At the end of this section we will show some examples of the effectiveness of this approach. Note that the new incremental update code only applies to the DFS traversal and corresponding topological vertex sorting. The SSLP relaxation step must be repeated in its entirety.

Below we show four procedures which must be called before a vertex is added to a graph, an edge weight is changed, an edge is deleted, or an edge in added. We assume that if a vertex is deleted from a graph all edges are deleted first, in which case nothing needs to be done. Hence there is no DELETE-VERTEX-INCR() routine.

```
ADD-VERTEX-INCR(v)
1    color[v] ← WHITE

CHANGE-EDGE-WEIGHT-INCR(src,snk,r)
1    w(src,snk) ← r

DELETE-EDGE-INCR(src,snk)
1    if src ≠ π[snk] or LIST-EMPTY(start)
2        return
3    INCR-RESET(snk)

ADD-EDGE-INCR(src,snk)
1    if b[src] > b[snk] and b[snk] ≥ 0
2        return
3    else if b[src] < 0 and b[snk] < 0
4        return
5    if b[src] < 0
6          targ← snk
7    else
8          targ← src
9    INCR-RESET(targ)
```

We assume that when a vertex is added, it is added before any of its edges. The processing of a vertex addition is therefore straightforward, requiring only that its color state be initialized to WHITE. Edge weight changes are also simple. The order in which vertices are visited in the DFS in independent of the edge weights, so nothing must be done. We will discuss the edge insertion and deletion cases individually below.

Several examples of edge deletions from a DAG are shown in Figure 19. Here we show the results of the previous DFS traversal with all edges intact. The *b*[*v*], *f*[*v*] beginning and finished time pairs are shown for each vertex, as well as the final contents of the *start* and *finish* lists. The dashed edges marked “a”, “b” and “c” will then be deleted from the graph in that order.

The case of edge “a” demonstrates the main action of the edge deletion algorithm. This edge is a “forward” edge in the graph, meaning that the source vertex of the edge, B, is also the DFS predecessor of the sink vertex, H. The deletion of this edge will effect the DFS traversal order. In order to put the graph into the proper state we must reset H, and all vertices which were *started* after H, to WHITE because we do not know when they will start or finish. In addition, we must reset vertices B, A and F to GRAY. These are the vertices which originally *finished* after H—their start times will not change, but their new finishing times will be unknown. In addition to resetting the vertex color state, all new GRAY vertices must be removed from the *finish* list, and all new

```
INCR-RESET(targ)
    if color[targ] = WHITE
        targ_white ← TRUE
    else if color[targ] = GRAY
        targ_gray ← TRUE
    if targ_white = FALSE
        while start_done = FALSE
            v ← LIST-POP-FRONT(start)
            if color[v] = GRAY
                v_gray ← TRUE
            color[v] ← WHITE
            if v_gray = FALSE
                if (v = targ)
                    start_done ← TRUE
                    finish_index ← LIST-NEXT(finish,v)
                LIST-ERASE(finish,v)
        if targ_gray = FALSE and targ_white = FALSE
            while finish_done = FALSE
                v ← LIST-POP-FRONT(finish)
                if v = finish_index
                    finish_done ← TRUE
                    LIST-PUSH-FRONT(finish,v)
```

white vertices must be removed from both the `start` and `finish` lists.

The code accomplishes this by erasing all vertices before and including `H` from the `start` list and coloring them `WHITE`. The same vertices are also erased from the `finish` list. It then erases all remaining vertices in the `finish` list which originally came before `H` and colors them `GRAY`. These two positions are marked in the figure with arrows in the two lists. This portion of the algorithm is given in the pseudo-code for the `INCR-RESET()` algorithm.

The case of edge “b” is much simpler. Edge “b” is not a forward edge, and it was never traversed by the original DFS algorithm. Its deletion will not effect the DFS traversal order and nothing needs to be done when it is deleted. This case is tested for at line 1 in `DELETE-EDGE-INCR()`.

The case of edge “c” is special in that the source and sink vertices have already been marked `GRAY` during the deletion of edge “a”. They are present in the `start` list but not the `finish` list. Situations such as this will arise when multiple incremental graph modifications are made before the `DFS()` algorithm is called to re-solve the graph. In such a case, if the sink vertex is already `WHITE`, everything potentially effected by this edge has already been reset so nothing needs to be done. If the sink vertex is `GRAY`, as in this case, mark it `WHITE` and delete it and its

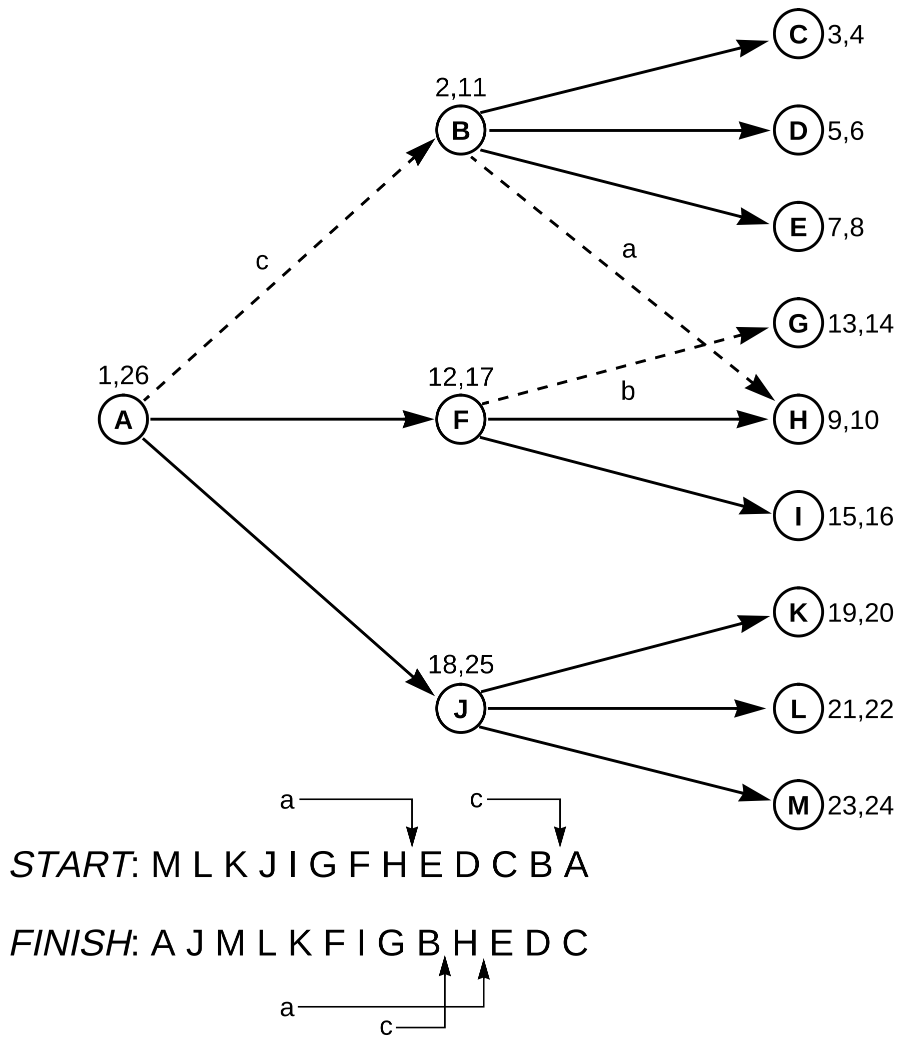


Figure 19: An example of incremental edge deletions from a DAG

predecessors as before from `start` list (the `finish` list is unaffected.) A similar situation can also arise while we are resetting vertices as in case "a". If at any point we begin to encounter `GRAY` vertices as we delete vertices from the `start` list, we know that they are not present in the `finish` list, so we should not try to delete them from the `finish` list again. This is the purpose for all of the tests for the color `GRAY` in the `INCR-RESET()` code. To unify this example, note that a vertex may only be set `WHITE` once, by its predecessor vertex, but it can be set `GRAY` multiple times by other edge additions/deletions to nodes that appeared earlier in the DFS traversal.

This example does not show any vertex additions because we assume that edges involving newly added vertices will not be added before the vertex, and will not deleted again before the

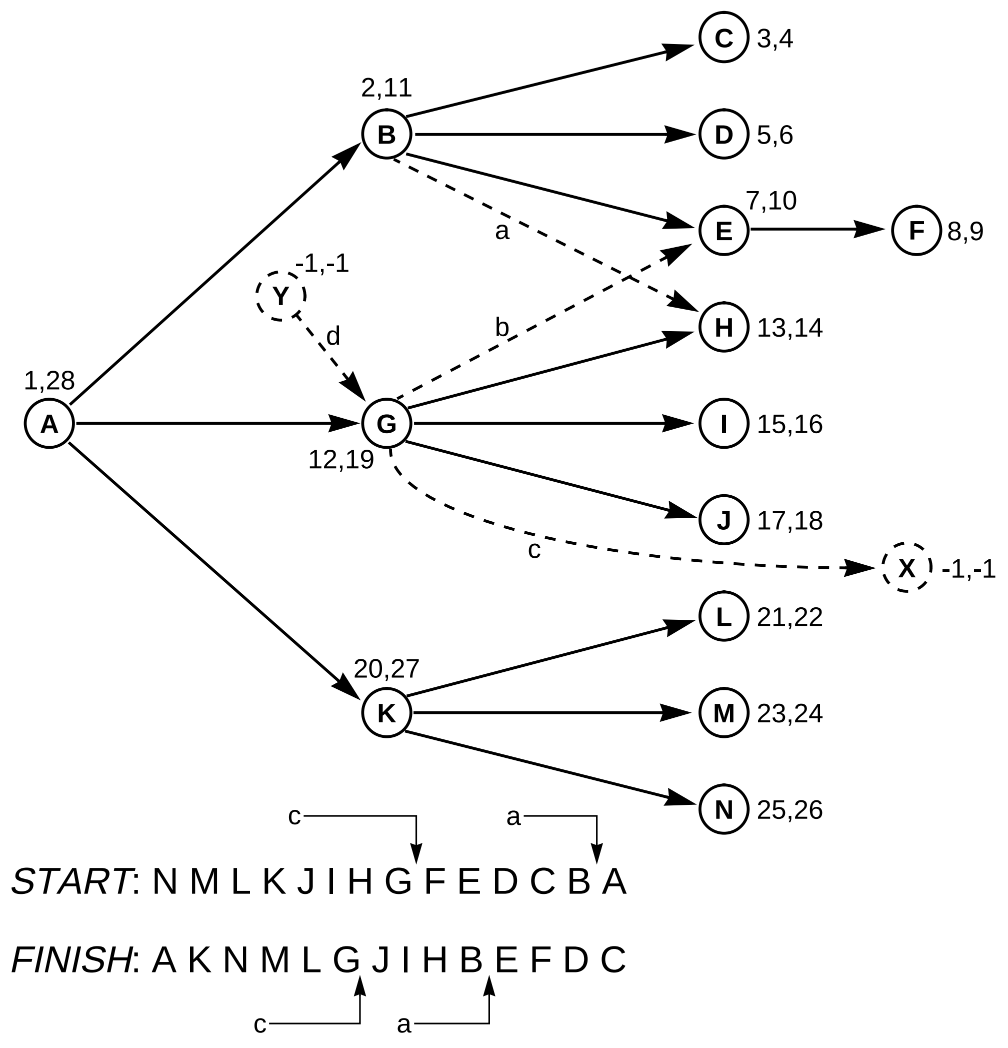


Figure 20: An example of incremental edge and vertex insertions in a DAG

graph is solved.

Examples of several edge addition cases are shown in Figure 20. The dashed edges “a”, “b”, “c” and “d” will be added to the graph in that order. In addition, the dashed vertices “x” and “y” are assumed to have been added to the graph since the last call to `DFS()`. Again we show the original start and finish times of each vertex and the state of the *`start`* and *`finish`* lists at the beginning of the edge addition/deletion process.

Again, case “a” is representative of the ordinary edge addition case. This new edge will be a forward vertex, because the source vertex discovery time is greater than the sink vertex discovery time, and the graph must be partially reset to reflect this change. The reset is performed in much the same was as with edge deletions, except that we must reset vertices to `WHITE` beginning at the source vertex, not the sink vertex. Note that this may be somewhat pessimistic (i.e. reset too many vertices to `WHITE`.) If we assume that new edges are added to the end of the edge list of a vertex, we could reset the source vertex to `GRAY`, and leave all of the source vertex's other sinks BLACK, since we know that the newest sink vertex will always be started last. However, we cannot derive this information from the start and finish lists, and more complex data structures would be needed. The time savings resulting from this optimization are questionable, and we have elected not to pursue it at this time. Instead, the only correct thing to do is to reset the source vertex to WHITE, guaranteeing that all of its sinks will also be reset WHITE, and therefore the order in which the sinks are re-visited is immaterial.

When edge "`b`" is added to the graph it will represents a backward edge, i.e. the discovery time of the source vertex is already greater than the discovery time of the sink vertex. Therefore the DFS traversal order will be unaffected by its addition, and nothing needs to be done. This condition is tested at line 1 of `ADD-EDGE-INCR()` and simply causes the incremental update algorithm to terminate.

Edges "`c`" and "`d`" involve new vertices which have been added since the last DFS traversal. New vertices will be given start and finish times of `-1`, as shown. If the sink vertex is new (case "`c`") we must reset the source vertex because we do not know if the edge will be a forward or a backward edge. If the source vertex is new (case "`d`") we need to reset the sink vertex since it may lie on a new forward path. However if both vertices are new nothing needs to be done as they currently represent a new disconnected sub-graph.

The run time of our incremental version of the SSLP algorithm is more difficult to analyze than that of the non-incremental version. Of course, in the worst case, all vertices will be reset to `WHITE` and the asymptotic run time will still be $O(V + E)$. In this case the actual run time will be higher because of the extra overhead of performing the resetting operations. However, the average case running time of our algorithm has the potential to decrease when the average number of reset vertices is small.

As we noted previously, the top-level `DAG-LONGEST-PATHS()` algorithm is unchanged in the incremental implementation. It must still relax all of the vertices in the *`finished`* list, so its $O(V + E)$ run time remains the same. Only the algorithm for `DFS()` has been modified. In the best case, such as a sequence of edge weight changes and "backward" edge additions and deletions, none of the graph will be reset and the routines `ADD-VERTEX-INCR()`, `CHANGE-EDGE-WEIGHT()`, `DELETE-EDGE-INCR()`, `ADD-EDGE-INCR()`, and `DFS-INCR()` will all execute in $O(1)$, so the actual run time will be approximately $1/2$ of the non-incremental version. In the average case, the number of vertices which are reset is difficult to predict, as this depends on the dynamics of the placement algorithm. The easiest way to evaluate the performance of the incremental graph update algorithms is by experimental means.

Figure 21 shows the dynamic run time behavior of the non-incremental and incremental SSLP algorithms on a problem consisting of 1,000 rectangles with randomly generated sizes. The simulated annealing algorithm described in Section 3.3.3 was run for several thousand iterations using a simple constant-rate cooling schedule, and the CPU time required to solve the *x* and *y* constraint graphs was measured for each iteration. Both experiments were started with the same random number seed, so the sequence of moves were identical. In the two graphs, the solid red line indicates a running average that is computed with a window size of 100 samples. The average run time per iteration was 2.37 seconds for the non-incremental algorithm and 15.6% smaller, 2.00 seconds, for the incremental algorithm. It is not clear why the running average tends to decrease over time in both graphs, or why there are so many outlying points with very high run times. These may be due to operating system effects.

The distribution of solution times for the incremental algorithm is clearly bi-modal. There is one tight band with a mean at 1.14 seconds, and a wider band with a mean at 2.47 seconds. The lower band represents the best-case scenario in which the incremental algorithm does not reset any vertices in the depth-first search, and for the most part is due to object rotation moves which, in this run, accounted for approximately 1/3 of the moves.

This bi-modal distribution can be better understood with the use of the graph shown in Figure 22. Here we measure the effectiveness of our node-resetting strategy. We find the percentage of the edges and the vertices that must be re-visited by the `DFS-INCR()` algorithm and sort them into bins. On the vertical axis we show the percentage of iterations that lies in each bin. The lower two curves show the absolute percentage, and the upper two curves show a cumulative running total which must sum to 100% at the last bin. The data show that 27.07% of the vertices fell into bin 0 (i.e 0% of the vertices needed to be re-visited), and 17.09% fell into bin 100. The remaining 55.84% fall in between with a fairly uniform distribution. The edge data is similar: 27.07% of the edges fell into bin 0, 30.79% fell into bin 100, while 42.14% fell in between.

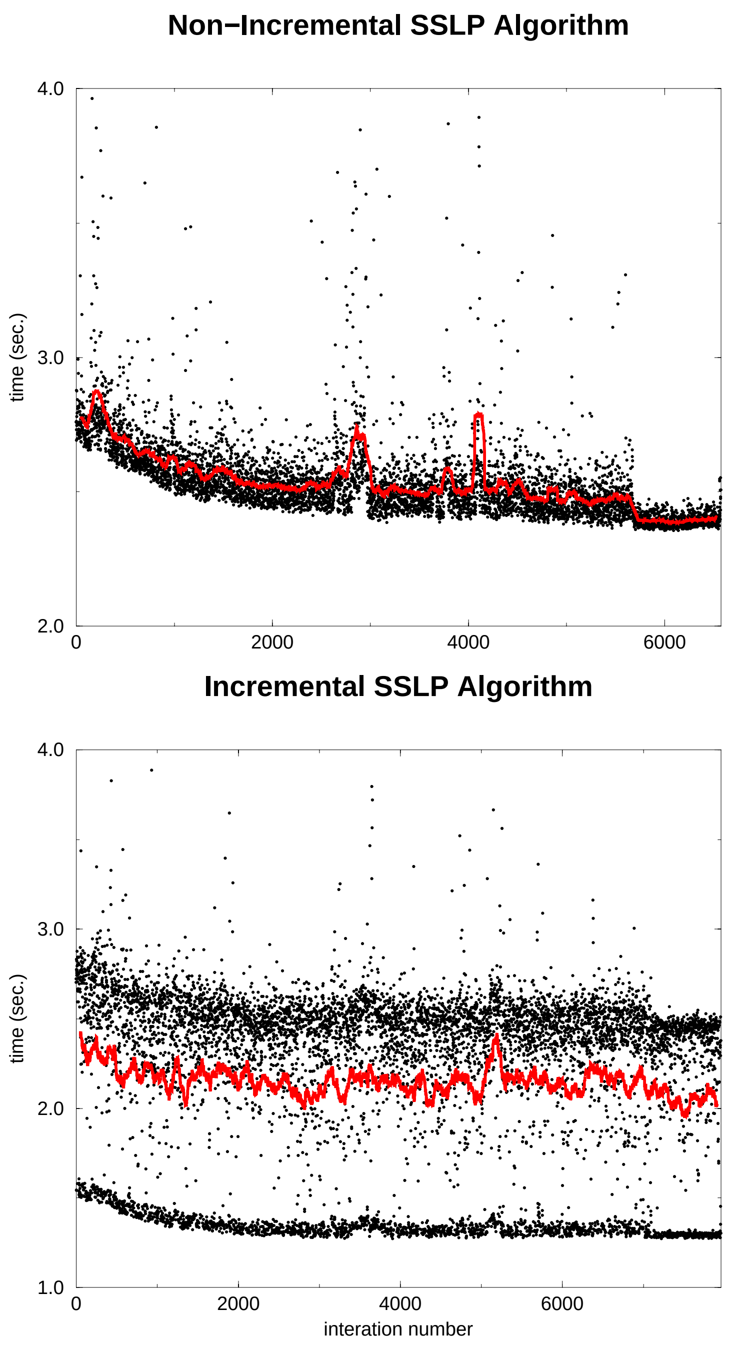


Figure 21: Constraint graph solution times for the incremental and non-incremental SSLP algorithms

Finally, we show how the incremental SSLP algorithm effects the total run time. An example with 100 randomly sized rectangles was run to completion using the same simulated annealing algorithm. The cooling schedule used 50,000 moves to arrive at a solution, and again the same ran-

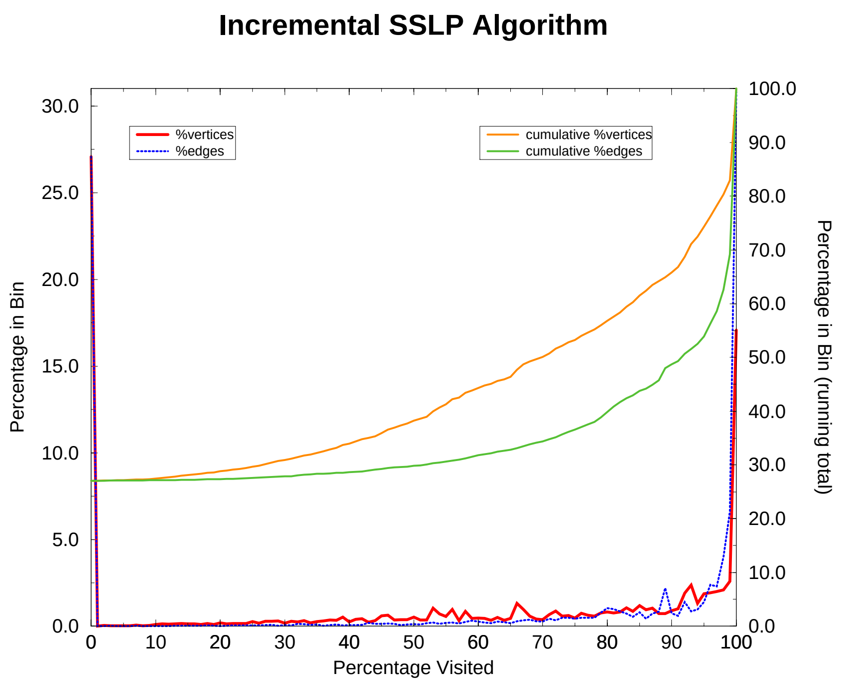


Figure 22: The percentage of edges and vertices reset by the incremental SSLP algorithm

dom number seed was used in both runs to guarantee identical moves. The non-incremental algorithm ran for a total of 2040.7 seconds, while the incremental algorithm ran for 1762.5 seconds, 13.6% faster.

3.3 Optimization Formulations

In this section we discuss three different formulations of the pairwise relative placement optimization problem. We make use of the sequence pair notation for the representation of the relative placement component of the search space. We discuss two exact formulations, one making use of integer linear programming, and one using a branch-and-bound traversal of the search space. We also describe a stochastic simulated annealing algorithm. We then conclude with some experiments demonstrating the performance of the stochastic technique relative to an exact technique.

3.3.1 ILP Optimization

Efficient and powerful optimization packages exist for problems which can formulated as linear programs. A **linear program**, or **LP**, consists of a linear objective function and a set of linear inequality constraints in which the free variables are real numbers:

$$\begin{array}{ll} \text{minimize} & \bar{c}\bar{x} \\ \text{subject to} & \mathbf{A}\bar{x} \le \bar{b},\ \bar{x} \ge 0 \end{array} \tag{17}$$

where $\bar{x}$ is a vector of unknowns, $\bar{c}$ is a vector of cost function coefficients, **A** is a matrix of constraint coefficients, and $\bar{b}$ is a column vector of right-hand-side values.

If the unknown variables x are restricted to the integers, the problems are referred to as **integer linear programs (ILPs)**. Problems in which only a subset of the variables are restricted to be integer are referred to as **mixed ILPs** (**MILPs**). If all of the variables are integer variables and restricted to the domain $\{0, 1\}$ the problems are called **0–1 ILPs** or **pseudo-boolean optimization** problems [3].

In linear programs the constraints form a convex hull, and they can be solved in polynomial time with methods based on the Simplex algorithm or the newer interior point algorithms [94]. ILPs are much more computationally intensive to solve. Common solution methods for ILPs [79] relax the integer constraints on the variables and then solve the corresponding linear program in the conventional manner. A variable is then selected and rounded up or down to the nearest integer, yielding a sub-problem that is solved recursively in the same manner until an all integer solution is obtained. A branch-and-bound search algorithm can be defined on this recursive process to obtain the optimum integer solution. Pseudo-boolean optimization problems can be solved using conventional ILP techniques by simply introducing additional constraints on the variables to restrict their domain. However, recent work by Barth [3] has demonstrated that techniques based on boolean satisfiability show promise as well.

If we can formulate our placement problem as some form of linear program we can make use of existing off-the-shelf optimization codes that have already been fine tuned for high performance. Our formulation is written as an Integer Linear Program (ILP). It is derived from the work of Sutanthavibul et al [115] and makes use of techniques from Devadas [21] and Gupta and Hayes [34,35]. As described in Section 2.4, each pair of objects can be in one of four possible relative positions, and this is enforced by introducing four constraints between every pair of objects (i, j), exactly one of which will be “active” in any given solution. These are written as follows

$$\begin{aligned}
\mathcal{I}_{ij}^{R}&: \quad x_i + z_i h_i + (1 - z_i) w_i \le x_j + K \cdot [(\rho_i > \rho_j) \vee (\eta_i > \eta_j)] \\
\mathcal{I}_{ij}^{L}&: \quad x_i - z_i h_j - (1 - z_j) w_j \ge x_j - K \cdot [(\rho_i < \rho_j) \vee (\eta_i < \eta_j)] \\
\mathcal{I}_{ij}^{T}&: \quad y_i + z_i w_i + (1 - z_i) h_i \le y_j + K \cdot [(\rho_i < \rho_j) \vee (\eta_i > \eta_j)] \\
\mathcal{I}_{ij}^{B}&: \quad y_i - z_j w_j - (1 - z_j) h_j \ge y_j - K \cdot [(\rho_i > \rho_j) \vee (\eta_i > \eta_j)]
\end{aligned} \tag{18}$$

Here $x_i \in \Re$ and $y_i \in \Re$ are the unknown real *x*-axis and *y*-axis coordinates of the lower-left corner of rectangle *i*. The constants $h_i \in \Re$ and $w_i \in \Re$ represent the height and width of rectangle *i*. An integer variable $z_i \in \{0, 1\}$ selects the rotation of rectangle *i*, $z_i = 0$ indicates that it is unrotated and $z_i = 1$ indicates that it is rotated 90 degrees. *K* is a suitably large constant, and the term in brackets which multiplies *K* acts as a **predicate** that selects one of the four constraints. Exactly one of the predicates will be zero in any given solution, in the remaining three constraints the *K* term will cause the constraint to be satisfied automatically.

The free variables $\rho \in \aleph$ and $\eta \in \aleph$ in the predicate expressions are integer variables representing the relationship between the two objects *i* and *j* in the two sequence pairs. The variables ρ_i and ρ_j represent the integer valued "slot", or location, of rectangles *i* and *j* in Γ^+. Similarly η_i and η_j represent their locations in Γ^-. The predicate express the partial ordering between ρ_i and ρ_j, and between η_i and η_j in the two sequence pairs.

We require a set of bounding constraints for all objects *i* that ensure that ρ_i and η_i do not exceed *m*, the number of objects being placed

$$\begin{aligned}
1 \le \rho_i \le m \\
1 \le \eta_i \le m
\end{aligned} \tag{19}$$

Note that we also require $\rho_i \ne \rho_j$ and $\eta_i \ne \eta_j$ for all $i, j \in 1 \ldots m$ and $i \ne j$. However, these will be handled implicitly by the constraints to be introduced next in Equation 20, so no further constraints are introduced here.

As written, Equation 18 does not represent a system of linear equations because of the presence of the logical "$\vee$" operator and the arithmetic ">" and "<" operators in the predicate expression. The equations can be linearized with the following technique. We define two new $\{0, 1\}$ integer variables, p_{ij} and n_{ij}, for each (i, j) pair. We then introduce the following constraints on the ρ and η sequence pair indices:

$$
\begin{aligned}
\rho_i > \rho_j: &\qquad (\rho_i - \rho_j) + K(1 - p_{ij}) > 0 \\
\rho_i < \rho_j: &\qquad (\rho_j - \rho_i) + K p_{ij} > 0 \\
\eta_i > \eta_j: &\qquad (\eta_i - \eta_j) + K(1 - n_{ij}) > 0 \\
\eta_i < \eta_j: &\qquad (\eta_j - \eta_i) + K n_{ij} > 0
\end{aligned}
\tag{20}
$$

in which K is a suitably large constant.

The constraints in Equation 20 operate as follows. Assume that $p_{ij} = 1$. In this case the constraint in Equation 20 labelled $\rho_i > \rho_j$ is enforced, while the constraint labelled $\rho_i < \rho_j$ is automatically satisfied. Notice that the exclusivity constraints $\rho_i \neq \rho_j$ and $\eta_i \neq \eta_j$ are handled implicitly in this formulation. However, the range constraints in Equation 19 are still required.

Using the fact that addition can be used to linearize the logical inclusive-or function, we define four temporary integer variables as follows

$$
\begin{aligned}
P_{ij}^1 &= p_{ij} + n_{ij} \\
P_{ij}^2 &= (1 - p_{ij}) + (1 - n_{ij}) \\
P_{ij}^3 &= (1 - p_{ij}) + n_{ij} \\
P_{ij}^4 &= p_{ij} + (1 - n_{ij})
\end{aligned}
\tag{21}
$$

which allows Equation 18 to be written in its linear form as

$$
\begin{aligned}
d_{ij}^R: &\qquad x_i + z_i h_i + (1 - z_i) w_i \leq x_j + K P_{ij}^1 \\
d_{ij}^L: &\qquad x_i - z_i h_j - (1 - z_j) w_j \geq x_j - K P_{ij}^2 \\
d_{ij}^T: &\qquad y_i + z_i w_i + (1 - z_i) h_i \leq y_j + K P_{ij}^3 \\
d_{ij}^B: &\qquad y_i - z_j w_j - (1 - z_j) h_j \geq y_j - K P_{ij}^4
\end{aligned}
\tag{22}
$$

The cost function for the ILP is defined as follows. Two artificial objects with zero width and height are created, sink_x and sink_y. Constraints are introduced to force all objects to lie to the right of sink_x and below sink_y.

$$
\begin{aligned}
0 \leq x_i \leq (x_{\text{sink}_x} - z_i h_i - (1 - z_i) w_i) \\
0 \leq y_i \leq (y_{\text{sink}_y} - z_i w_i - (1 - z_i) h_i)
\end{aligned}
\tag{23}
$$

for all $i = 1 \ldots m$. The cost function can be written as one-half of the perimeter of the packing, which is simply

$$\text{cost} = x_{\text{sink}_x} + y_{\text{sink}_y} \tag{24}$$

Including routing effects in the ILP is a matter of augmenting the cost function to include some linear measure of the routing cost. One possible metric discussed in Section 2.4.6 is one-half of the perimeter of the bounding box enclosing two objects that must be connected with a net. The cost function thus becomes

$$\text{cost} = x_{\text{sink}_x} + y_{\text{sink}_y} + \sum_{\forall (i,\, j)\text{ connected}} |x_i - x_j| + |y_i - y_j| \tag{25}$$

In Equation 25 the absolute value terms are non-linear. However, they are straightforward to linearize. To represent the absolute value function $|Z|$ we create two new real variables $a \geq 0$ and $b \geq 0$, and impose the constraints $Z = a - b$ and $a \cdot b = 0$. Note that Either a or b must be zero. If $a = 0$ then $Z \leq 0$ and $b = |Z|$. Conversely, if $b = 0$ then $Z \geq 0$ and $a = Z$. Therefore we can write $|Z| = a + b$. The constraint $a \cdot b = 0$ is also nonlinear, but it can be linearized by introducing a $\{0, 1\}$ integer variable s and writing $a(s) + b(1 - s) = 0$. Thus, the cost function in Equation 25 can be written as

$$\text{cost} = x_{\text{sink}_x} + y_{\text{sink}_y} + \sum_{\forall (i,\, j)\text{ connected}} a_{x_{ij}} + b_{x_{ij}} + a_{y_{ij}} + b_{y_{ij}} \tag{26}$$

if we introduce the following new constraints for all pairs of objects (i, j) such that $i \in \{1 \ldots m\}$, $j \in \{1 \ldots m\}$ and (i, j) are connected by a net in the schematic:

$$\begin{gathered} x_i - x_j = a_{x_{ij}} - b_{x_{ij}} \\ y_i - y_j = a_{y_{ij}} - b_{y_{ij}} \\ a_{x_{ij}}(s_{x_{ij}}) + b_{x_{ij}}(1 - s_{x_{ij}}) = 0 \\ a_{y_{ij}}(s_{y_{ij}}) + b_{y_{ij}}(1 - s_{y_{ij}}) = 0 \\ 0 \leq s_{x_{ij}} \leq 1,\ 0 \leq s_{y_{ij}} \leq 1 \\ a_{x_{ij}} \geq 0,\ b_{x_{ij}} \geq 0,\ a_{y_{ij}} \geq 0,\ b_{y_{ij}} \geq 0 \end{gathered} \tag{27}$$

The complexity of an ILP optimization problem is often characterized by its "size" expressed as the number of constraints and variables in the problem. In the formulation described above, the pairwise relative placement problem for m objects requires $2m$ real variables (x_i and y_i), $2m^2$ $\{0, 1\}$ integer variables (z_i, p_{ij}, and n_{ij}), and $2m$ integer variables (ρ_i and η_i.) The problem also specifies $8m^2 + 4m$ constraints (Equations 19, 20, 22, and 23.) If routing cost is included in the cost function, as in Equation 26, an additional $4m^2$ real variables ($a_{x_{ij}}$, $a_{y_{ij}}$, $b_{x_{ij}}$, and $b_{y_{ij}}$) and $2m^2$ $\{0, 1\}$ integer variables ($s_{x_{ij}}$ and $s_{y_{ij}}$), as well as an additional $4m^2$ constraints (Equation 27), are required. These results are summarized in the following table

Table 2: ILP problem size for *m* objects

	no-routing	routing
real variables	$2m$	$4m^2 + 2m$
integer variables	$2m$	$2m$
$\{0, 1\}$ integer variables	$2m^2 + m$	$4m^2 + m$
constraints	$8m^2 + 4m$	$12m^2 + 4m$

In practice, the computational cost of solving a particular ILP problem is most closely related to the number of integer variables, which in this case is $O(m^2)$. This cost results from the combinatorial branch-and-bound search which must be performed in order to obtain an optimal integer solution for these variables. In the worst case, our problem formulation will require $O(2^{m^2})$ variable relaxations in order to obtain an optimal solution. Clearly, the ILP formulation has not allowed us to escape the NP-complete nature of the unbounded sequence pair bin packing problem.

Many large ILP problems have been demonstrated to be solvable in the average case with the use of sophisticated ILP solvers. However. these problems generally have some underlying structure which allows effective bounding during the branch-and-bound variable relaxation step. As we will demonstrate in Section 3.4, the dedicated branch-and-bound algorithm to be presented next in Section 3.3.2, which is tailored specifically to our problem, showed poor bounding behavior. This discouraged us from implementing the ILP formulation of the optimization problem, though the details presented here remain useful as they provide some insight into the nature of the search space.

3.3.2 Branch-and-Bound Optimization

Branch-and-Bound is a general optimization technique for combinatorial optimization through systematic exhaustive search. Its main features are a recursive incremental search through the search space and an ability to prune away parts of the un-searched space that can be shown to be sub-optimal.

The pattern of search through the solution space can be viewed as a recursive search tree,

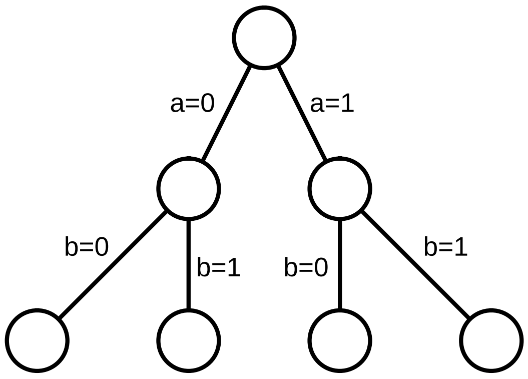


Figure 23: A branch-and-bound search tree

as seen in Figure 23. The search begins at the root of the tree with all of the variable values undecided. The search engine then selects one of the free variables and fixes it to one of its possible values, a process called **branching**. The choice of which variable to select and which of its values to assign are based on some **decision heuristic**. The tree is traversed depth-first until a leaf node, i.e. a tree node corresponding to a complete assignment to the problem variables, is reached; this represents a feasible solution to the optimization problem. The algorithm can then **backtrack** and reverse the most recent decision in order to continue the search for the optimal solution. However, the algorithm need not always explore all the way to a leaf node before backtracking. If it can prove, at some internal non-leaf node, that the current partial solution will not lead to an optimal solution, it can **bound** the search and backtrack immediately, potentially pruning away vast regions of the search space. This exhaustive search is obviously combinatorial in nature, and obviously of exponential complexity. An implementation's ability to bound is critical to its practicality.

Bounding is implemented as follows. In the case of a minimization problem, as it enters a new node in the search tree the algorithm calculates a local **lower-bound** on the value of the cost function for the current partial solution. The algorithm also maintains a global **upper-bound** on the optimum solution, which corresponds to the best solution yet encountered in the search. If at any time the local lower-bound becomes larger than the global upper-bound, the algorithm can backtrack.

For branch-and-bound to be applied to a particular optimization problem it must be possible to calculate this lower-bound on the cost function when given a partial solution with some parameters undefined. The tighter that it is possible to make this lower-bound, the more effective the algorithm will be. The distinguishing features of a branch-and-bound algorithm are thus its branching strategy and its lower-bound calculation, and to a lesser extent its decision heuristic. We will describe one possible branch-and-bound implementation for the rectangle packing problem based on the sequence pair.

Given the sequence pair representation of the rectangle packing problem, the free parameters are the slot numbers in the two sequences (Γ^{+}, Γ^{-}) at which each rectangle is placed and the orientation of each rectangle. The only constraints are that each slot in each of the two sequences must by assigned to exactly one rectangle.

Our branching strategy seeks to build up the sequence pairs incrementally, starting from length zero and building up to length *m*. At each branch in the decision tree, the decision being made is which rectangle to select for placement and at which two slots in the partial sequence pair it will be placed. This strategy is based on the observation that a partially constructed sequence pair must contain the same set of rectangles in both sequences. One can view this process as generating all possible permutations, in a pair of sequences with identical members, in a recursive incremental fashion.

We show an example of our branching strategy for a single sequence in Figure 24. Here we have a set of four objects with the names $\{\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}\}$ and we show the search tree required to construct all possible permutations of this set. At each level in the tree one of the un-placed set members is selected for placement, progressing from left-to-right we select the members in the order **A**, **B**, **C**, **D**. The nodes at any given level are visited in top-to-bottom order. The different branches at each level are distinguished by which "slot" in the partial sequence the new element is placed. One can see that the leaves of the tree indeed contain all possible permutations of the set.

Figure 24 demonstrates how one searches a single permutation. However, the sequence pair requires two permutations to be constructed simultaneously, and at every step in the search tree the two partial sequences must have the same membership. This is easily accomplished by extending the same branching strategy outlined above. Again the decision being made at each level of the tree is which set member is selected for placement. The different branches at each level concern which slot in the sequence each member is placed into, only now we must try all possible combinations of slot assignments in both sequences. One can visualize this in Figure 24 by reproducing a copy of the tree extending into the page in a third dimension. For example, at level-3 in the tree we select **C** for placement. Progressing top-to-bottom we assign the sequence Γ^{+} the value (ABC), and proceeding into the page we assign the sequence Γ^{-} the values (ABC),

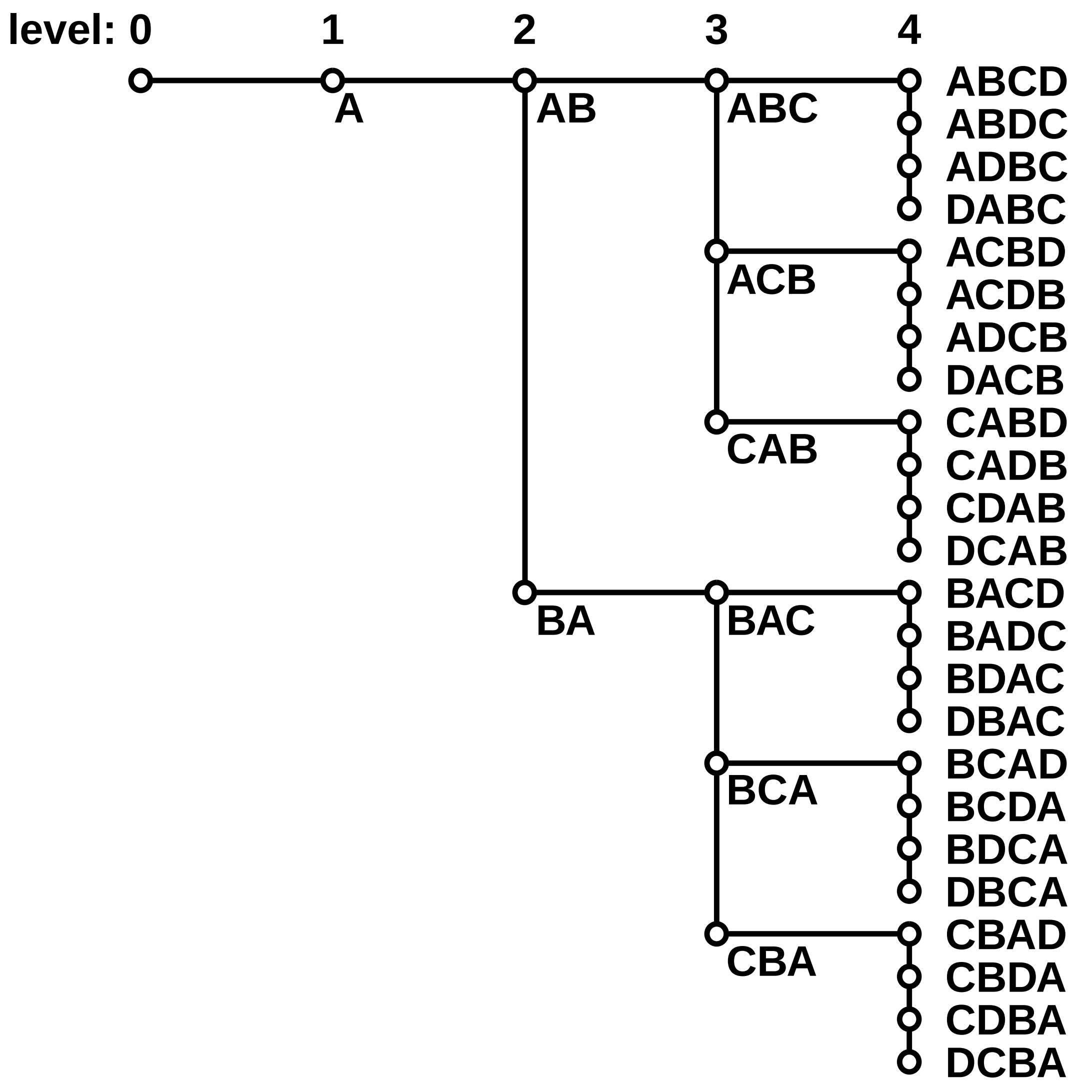


Figure 24: Branch-and-Bound permutation generation

(ACB), and (CAB) in turn. Then we try $\Gamma^{+} = (ACB)$ and $\Gamma^{-} = (ABC), (ACB), (CAB)$ in turn.

In addition to the sequence pair slot location, the decision tree must specify an orientation for each rectangle. In the case of simple rectangles each block has only two orientations: rotated by 0° and 90°. In order to include rectangle orientation, every object must be tried in every slot location in each of the two rotations.

Using the branching strategy described above it is a fairly straightforward process to calculate a lower bound on the value of the cost function at each node in the search tree. Each node represents a packing of the rectangles which have already been assigned, all unassigned rectangles

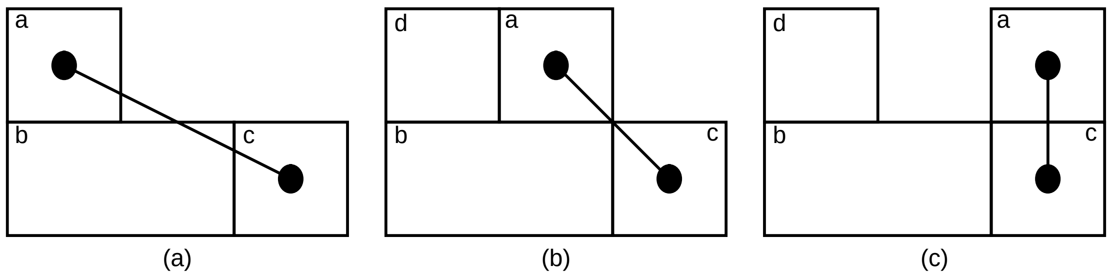


Figure 25: Example of routing effects in branch-and-bound placement

have not yet been placed in the packing. One can view this as a constructive placement process whereby rectangles are selected one by one and placed in every possible relation to the rectangles that have previously been placed. At each branch we solve the constraint graphs implied by the partial sequence pair to obtain the area (or perimeter) of the current partial placement. This is obviously a lower bound on the area of the final placement—as new rectangles are added to the placement the cost function is guaranteed to be non-decreasing. We maintain a global upper bound which is the area of the smallest complete placement encountered so far, and if the lower bound, the area of the current placement, exceeds this upper bound, it is clear that the algorithm can backtrack.

A nice property of this branching strategy is that one can backtrack between branches, for example between the branches $\Gamma^{-} = (ABC), (ACB), (CAB)$ above, with a single pairwise exchange. This leads to a very efficient implementation, as the constraint graph can be modified incrementally as discussed in Section 3.2.

If this branch-and-bound algorithm is to be extended from the general rectangle packing problem to the VLSI placement problem we must include the cost of routing in the cost function. This turns out to be complex. Because the cost function is used to calculate a lower bound on the cost of a partial solution, it must be non-decreasing as new objects are added to the partial placement. However, this will not always be the case if a traditional routing metric is included, as shown in Figure 25. Here, in Figure 25(a), three blocks are shown with a net between the centers of blocks a and b. In Figure 25(b) a fourth block, d, has been added by a branch to a new depth in the search tree. As can be seen, the distance between blocks a and c has decreased, and so will the routing component of the cost function, yet the total area of the packing has not changed. It is therefore possible for the total cost to decrease as new blocks are added to a partial solution, which would imply that the cost function can no longer be used as a lower bound.

We conjecture that this final problem can be solved by introducing wire-length minimization [39, 55, 99] into the constraint graph solution step. As shown in Figure 25(c), block a has some range of motion in the solution over which the placement area doesn't change—i.e. there is some slack in the constraint graph. If block a is placed within its range of slack so as to minimize the total routing length, the introduction of block d does not decrease the total wire length in the circuit. We postulate that the introduction of a new block into a compacted placement with minimum wire length will never decrease the total wirelength, and thus the non-decreasing property of the cost function will be maintained. Hambrusch and Tu [39] have developed an $O(n_h \cdot n \log n)$ algorithm for finding the minimum width layout among all layouts with minimum wire length during horizontal 1-dimensional compaction, where n_h is the number of horizontal wires and n is the number of nodes in the constraint graph.

3.3.3 Simulated Annealing Optimization

Simulated annealing [50, 85, 91] is an optimization technique for combinatorial systems which is based on a physical analogy to the thermodynamics of a solid crystallizing from a high temperature liquid melt. In this physical system, the free parameters describe the positions of the individual molecules, which are represented by spheres in a two-dimensional or three-dimensional bounded space. The cost function, which is being minimized, describes the total energy of the system which can be calculated as a decreasing function of the distance between the individual molecules. In such a physical system a good way to obtain a high quality solid is to heat the material up to a very high temperature and then slowly cool it until the material freezes out into its final structure.

Simulated annealing was first described by Metropolis et al in 1953 [75] for the purpose of describing the state equations of physical systems of interacting particles which could not be found analytically. Kirkpatrick et al [50] later demonstrated that it could be applied to the general problem of combinatorial optimization. The system energy could be abstracted to nearly any cost function, whether described analytically or experimentally.

Simulated annealing proceeds as a chain of modifications to the state of the system being optimized. The set of transformations which take the system from one state to the next are referred to as the **move set**. After each move the value of the cost function, referred to as E, the **energy** of the system, is evaluated. The move is either accepted or rejected based on the change in energy,

ΔE as follows:

$$\begin{array}{l} \text{if } \Delta E \le 0\text{, accept the move} \\ \text{if } \Delta E > 0\text{, accept the move with probability } P(\Delta E) \propto e^{-\Delta E / T} \end{array} \tag{28}$$

where T is a parameter which decreases over the course of the optimization, and is a direct analogy to the **temperature** of the physical system. This criterion for move acceptance is usually referred to as the Metropolis criterion after its original authors [75].

A distinguishing feature of the Metropolis criterion is that it permits the process of “hill climbing”. “Downhill” moves which decrease the cost function are always accepted. However, “uphill” moves which increase the cost function are still accepted with a certain probability in the hope that they will allow the system to escape from a local minimum. The probability function $P(\Delta E)$ decreases exponentially with the increase in ΔE, and has an inverse relation to temperature. At high temperatures nearly every uphill move is still accepted, while at low temperatures only very small increases will be tolerated. This particular probability function relates the system energy to the temperature through the observation that, at a particular temperature, the probability that the system is in a particular state with energy E is proportional to the Boltzmann distribution, $e^{-E/T}$.

The simulated annealing algorithm initializes the system to a particular starting placement at a particular temperature. It then proceeds to select random moves from the move set and evaluate them according to the Metropolis criterion. Periodically, after some criterion (usually called the “inner-loop criterion”) is met, the temperature is reduced by some amount until another criterion (usually called the “outer-loop criterion”) is met, at which time the algorithm terminates. The inner and outer-loop criteria, together with the temperature reduction function, collectively called the **cooling schedule**, can have a profound effect on the quality of the final solution as well as on the required computation time.

In order to tailor this generic simulated annealing strategy to a particular problem one must first characterize the free variables which will be used to model the solution state space. Then a set of moves must be selected that will allow the algorithm to traverse from one state to another. Finally a number of implementation issues must be resolved concerning the detailed control of the algorithm. A method for selecting the initial placement and initial temperature must be specified, as well as the properties of the cooling schedule. We will discuss each of these topics in the following sections.

3.3.3.1 Solution Space Modeling and the Move Set

Our simulated annealing move set, designed for use with the sequence pair model for two dimensional placement, is based on the work of Murata et al. [77]. The free variables are the two *m*-vectors of the sequence pair, (Γ^+, Γ^-), and an additional *m*-vector of $\{0, 1\}$ variables, Θ, called the **orientation vector**, representing rectangle orientation. If $\Theta_i = 0$, rectangle *i* is unrotated in the packing, otherwise it is rotated 90-degrees.

The move set consists of seven different moves:

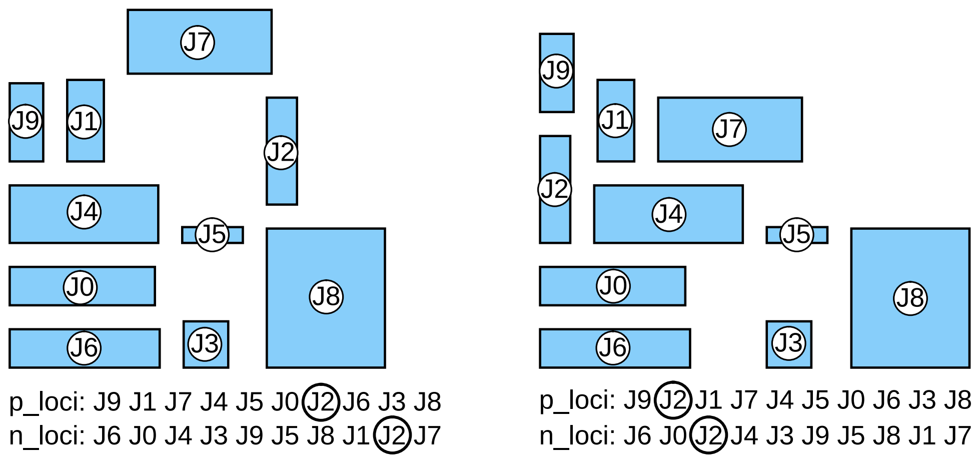
1. a pairwise interchange of two rectangles in Γ^+
2. a pairwise interchange of two rectangles in Γ^-
3. a pairwise interchange of two rectangles in both Γ^+ and Γ^-
4. a translation of one rectangle in Γ^+
5. a translation of one rectangle in Γ^-
6. a translation of one rectangle in both Γ^+ and Γ^-
7. a 90 degree rotation of one of the rectangles in the orientation vector

The translation moves select a random rectangle for translation. In moves 4 and 5, a random distance (positive or negative) is selected and the rectangle is moved that distance in one of the sequences. The rectangles between the old location and the new location are then moved over one position to accommodate the move. If move 6 a different random distance is chosen for each sequence and the selected rectangle is moved in both. The two pairwise interchange moves select two objects at random for interchange.

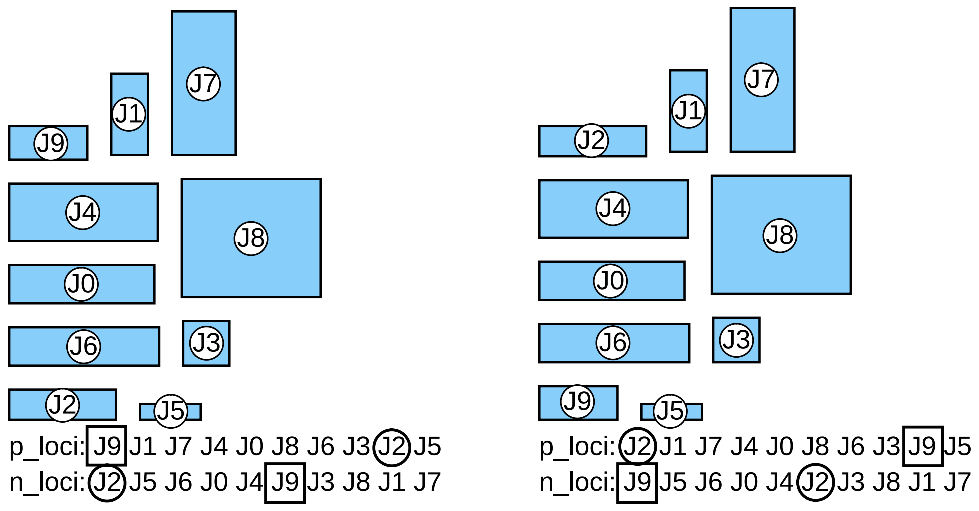
We note that only moves 1,3 and 7 were used by Murata et al They have proven that these three moves are complete, meaning that every possible placement is reachable from every other placement. However, the additional moves make more placements reachable in a single move, which increases the efficacy of the search.

As we will see in Chapter 4, it is trivial to extend the concept of the orientation vector to include arbitrary objects with more than one possible rotation/mirroring orientation, and with more than one possible configuration. The variables of the orientation vector correspond directly to the d_i decision variable discussed in Section 2.4.1.

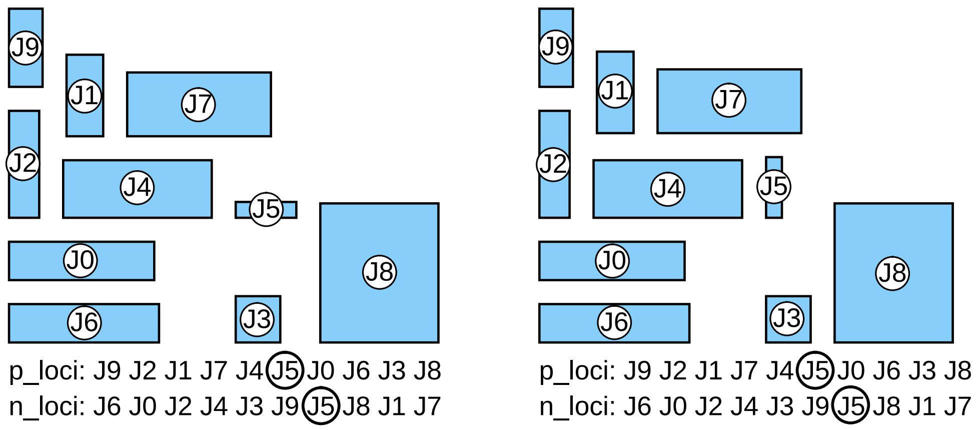
Examples of the three move categories are shown in Figure 26. In Figure 26(a) we show an example of move type 6, rectangle J2 is translated in Γ^+ by a distance of -5 and in Γ^- by a distance of -6. Figure 26(b) demonstrates an example of move type 3, rectangles J2 and J9 are


J7
J9
J1
J2
J4
J5
J0
J8
J6
J3
p_loci: J9 J1 J7 J4 J5 J0 J2 J6 J3 J8
n_loci: J6 J0 J4 J3 J9 J5 J8 J1 J2 J7
p_loci: J9 J2 J1 J7 J4 J5 J0 J6 J3 J8
n_loci: J6 J0 J2 J4 J3 J9 J5 J8 J1 J7

(a) translation move: J2


J7
J1
J9
J2
J4
J8
J0
J6
J3
J5
p_loci: J9 J1 J7 J4 J0 J8 J6 J3 J2 J5
n_loci: J2 J5 J6 J0 J4 J9 J3 J8 J1 J7
p_loci: J2 J1 J7 J4 J0 J8 J6 J3 J9 J5
n_loci: J9 J5 J6 J0 J4 J2 J3 J8 J1 J7

(b) pairwise interchange move: J2,J9


J9
J1
J7
J2
J4
J5
J0
J8
J6
J3
p_loci: J9 J2 J1 J7 J4 J5 J0 J6 J3 J8
n_loci: J6 J0 J2 J4 J3 J9 J5 J8 J1 J7
p_loci: J9 J2 J1 J7 J4 J5 J0 J6 J3 J8
n_loci: J6 J0 J2 J4 J3 J9 J5 J8 J1 J7

(c) rotation move: J5

Figure 26: Examples of the three major annealing move types

interchanged in both sequences. Finally, in Figure 26(c) we show an example of move type 7, rectangle J5 is rotated from a horizontal orientation to a vertical orientation.

3.3.3.2 Cost Function

One of the attractive properties of optimization through simulated annealing is that the algorithm is extremely robust in the presence of almost any type of cost function. Simulated annealing places no requirements on the linearity or convexity of the cost function. The cost function is not even required to be specified in an analytic form—cost functions derived through numerical techniques or simulation are easily supported.

The only significant requirement on a cost function for use in a simulated annealing environment is that the function must be unconstrained. The simulated annealing algorithm makes no provision for the presence of infeasible placements in the search space. Infeasible placements can be avoided by assigning them an infinite cost, in which case they will always be rejected, but this solution has an unfortunate drawback. In the most general terms, the simulated annealing formulation makes no assumptions about the convexity of the solutions space (indeed, one of its strengths is that is requires no such assumptions). In a non-convex constrained space, it may be true that the shortest path from the current placement to the global optimum passes through a region of infeasibility. In fact, the current placement or the global optimum may be completely enclosed by an infeasible region, and the algorithm may be required to take several steps through this region before feasibility is regained. The capability to move through regions of infeasibility can be extremely helpful in escaping from local minima.

The most common method for mapping solution space constraints into an unconstrained simulated annealing formulation is to replace the constraints with a penalty term in the cost function. For example, in the *Timberwolf-3.2* standard cell placement and routing system [101,102], constraints preventing the overlap of adjacent cells within a row are translated into a penalty term in the cost function which is proportional to the amount of overlap. Timberwolf also includes a penalty term which ensures that the lengths of the standard cell rows are reasonably uniform. A second example is provided by the *Koan* analog circuit placement tool [17,18]. As in *Timberwolf-3.2*, *Koan* makes use of a cost function penalty term to manage illegal object overlaps. *Koan* also uses penalty terms to control an aspect ratio constraint on the layout and a maximum object separation constraint between critical components.

By treating the constraints as penalty terms in the cost function, at high temperatures the annealing algorithm is free to traverse through an infeasible region of the original problem space. As the temperature is lowered this becomes more and more difficult as the increase in cost becomes prohibitive. In order to ensure that all of the constraints in the original constrained version of the problem are satisfied in the final solution, it must be guaranteed that all of the penalty terms are driven to zero at the termination of the algorithm. However, it can be very difficult to tune the cost function to ensure that this is guaranteed. It may be true that an infeasible placement provides such a strong gain in terms of one term of the cost function (say area) that it compensates for the cost of the penalty term (say illegal overlap). *Timberwolf-3.2* uses a fixed weight on the penalty terms, but *Timberwolf-SC* [103] and *Koan* use a more sophisticated scheme which gradually increases the cost of the penalty term as a function of temperature.

Our simulated annealing formulation makes almost direct use of the general placement and routing cost function outlined in Chapter 2 Section 2.4.6. The cost is the weighted sum of the cost of the placement and the cost of an estimate for the routing:

$$\text{cost} = w_1 \cdot \text{placement} + w_2 \cdot \text{routing} \tag{29}$$

where $w_i \in \mathbf{R}$ is a real valued weight on term i of the function. Any of the methods for routing cost estimation discussed in Section 2.4.6 may be used here depending on the degree of accuracy required. In addition to the two options for placement cost outlined in Section 2.4.6, placement *area* and placement *perimeter*, we support two other options within the simulated annealing formulation of the sequence pair optimization problem: width (height) minimization with constrained height (width), and area or perimeter minimization within a fixed aspect ratio.

In these latter two cases above we make use of the techniques described above to transform these constraints into penalty terms in the cost function. In the case of a circuit with a fixed dimension, say height, we write the placement cost as the sum of the width and the product of the width and the height constraint violation:

$$\text{Cost}_{Placement} = \begin{cases} w_{1a} \cdot width + w_{1b} \cdot width \cdot \dfrac{height}{target} & \text{if } (height > target) \\ w_{1a} \cdot width & \text{otherwise} \end{cases} \tag{30}$$

where w_{1a} and w_{1b} are separate weight values and *target* is the target cell height. For area minimization with a fixed aspect ration we use a similar function which penalizes aspect ratios which are not equal to the target aspect ratio. We currently use fixed values for the penalty term weights, as the cost function has not proved to be overly dependent on this value and the adaptive weight factor ageing techniques discussed above did not seem necessary.

3.3.3.3 Annealing Control

In an implementation of the simulated annealing algorithm one wishes to obtain a high quality solution which is close to the global optimum, but one also faces a limited quantity of computation resources. There are several algorithmic issues concerning the detailed control of the simulated annealing algorithm which have a direct bearing on the execution time of the algorithm and on the quality of the final result. An initial placement and temperature must be selected, and an appropriate cooling schedule chosen. The cooling schedule determines how many moves to make at each temperature step, and the magnitude of the reduction in temperature at the end of each step. A stopping criterion must be chosen to determine when the algorithm should terminate. Finally, a method of move selection must be specified.

Early implementations of simulated annealing [101,102] used fairly simple heuristics for each of these tasks, and often made use of a large number of user-specified tuning parameters which had to be adjusted for each problem. However, recent work has shown that the link between combinatorial optimization and thermodynamics, based on the original analogy to statistical physics, can be leveraged to provide adaptive problem-independent schemes for the control of the state of the algorithm [56,57,85]. We draw most of the details for our implementation from the review provided by Cohn [17]. One highlight of the techniques outlined here is that manual control of the algorithm has been reduced to two user-tunable parameters.

Viewing the optimization problem as a thermodynamic system, the use of the Boltzmann distribution as the basis for the Metropolis criterion assumes that the system is in thermodynamic equilibrium. We will see that most of the recent techniques developed for the dynamic control of the simulated annealing process adopt this as a constraint. They adjust the control parameters so as to maintain the system in a state of quasi-equilibrium.

Initial Placement and Temperature Selection

It has been observed [102 p. 433] that if the initial temperature is selected to be high enough, the initial placement will have no impact on the final solution quality. Hence it is normally selected at random. We adopt the convention of Murata et al to begin the process with $\Gamma^{+} = \Gamma^{-}$.

This has the effect of beginning with all rectangles lined up in a horizontal row.

We make use of an initial temperature selection scheme due to Huang et al [44], which is based on an analysis by White [125]. They observe that at an infinite temperature the probability of visiting any particular state is equally likely, and make the assumption that the density of states with a particular energy approaches a Gaussian distribution near the average energy. The temperature is judged to be high enough if the distribution of the density of states which are visited approximates this gaussian distribution, and this will occur when $T \gg \sigma$, where σ is the standard deviation of the energy distribution. The initial temperature T_∞ is computed using the equation

$$T_\infty = k\sigma. \tag{31}$$

The constant *k* is selected such that the temperature will be high enough to accept, with probability *P*, a state whose energy is three standard deviations worse than the mean. Thus,

$$k = -\frac{3}{\ln(P)}. \tag{32}$$

If we assume that the distribution is indeed gaussian, $P \cong 85\%$ and $k = 20$. In order to determine the standard deviation of the cost density we begin the simulation at an infinite temperature and run the algorithm until our equilibrium detection criterion, discussed next, is satisfied.

Inner Loop Criterion (Equilibrium Detection)

The “inner loop” criterion in the generic simulated annealing algorithm determines when an adequate number of simulations have been performed at the current temperature, allowing the temperature to be reduced. *TimberWolf-3.2* [102] used a piecewise constant cooling schedule with a fixed number of iterations at each temperature. We make use of an adaptive technique due to Huang et al [44]. Following from the physical analogy, we judge that it is safe to decrease the temperature when the system has reached thermal equilibrium at the current temperature. In thermodynamics, equilibrium represents a steady state condition in which the probability distribution of all reachable states has stabilized. However, in practice this will never occur as the number of reachable states is enormous. Instead, the authors use a more practical definition for equilibrium based on the goal that the probability distribution of the system *energy* will have reached steady state.

Equilibrium is established when the ratio of the number of new states generated with their costs within a certain range $\pm\delta$ of the average cost $\overline{C}$, to the total number of new states, will reach a stable value χ. Here δ is called the **sampling range**. If the state distribution is assumed to be gaussian (strictly true only at high temperatures), $\chi = \mathrm{erf}(\delta/\sigma)$. A typical value for δ is half of the standard deviation of the cost distribution, 0.5σ. Then, $\chi = \mathrm{erf}(0.5) = 0.38$.

Using the notation of Cohn [18], this scheme is implemented by establishing a two-bin counting system. We monitor the average cost of the system over the current temperature step. A successful move is placed in the first bin, called the *within-count*, if its cost falls in the range

$$\overline{C} - \frac{\sigma}{2} \le \text{cost} \le \overline{C} + \frac{\sigma}{2} \tag{33}$$

and it is placed in the second bin, called the *tolerance-count*, if it falls outside of this range. Upper limits are placed on the sizes of these two bins as follows:

$$\begin{aligned} \chi_{\text{inside}} &= \mathrm{erf}(0.5) \cdot \eta D = 0.38\eta D \\ \chi_{\text{outside}} &= (1 - \mathrm{erf}(0.5)) \cdot \eta D = 0.62\eta D \end{aligned} \tag{34}$$

where *D* is a measure of the problem size, in our case the number of rectangles, and η is the first of our manual tuning parameters (a value of 3 is suggested). If the size of the within-count reaches χ_{inside} before the tolerance-count reaches χ_{outside} we judge that equilibrium has been achieved. If the χ_{outside} limit is reached first we assume that equilibrium has not been reached, and we reset all of the counters and begin counting again.

As a practical matter, the authors observed that the maximum number of *attempted* moves must be given a fixed upper limit at low temperatures because equilibrium may never be reached. Cohn [17] notes that this is because the extremely small variation in the cost of accepted moves implies a very small value of σ and thus a very tight sampling range. The authors suggest that this upper limit be set to the number of states *M* which can be reached in one move. In our case $M = m^3$ (there are $m(m-1)$ possible pairwise swaps and *m* rotation moves.)

In addition, in order to guarantee that the collected statistics are valid, the authors also enforce a lower limit on the number of *accepted* moves at each temperature. This limit is also set to *M*. Obviously at low temperatures we may never reach this lower limit on accepted moves while the upper limit on attempted moves is in force, so if this occurs we relax the upper limit to a maximum of $4M$.

The Temperature Decrement

When equilibrium has been established the temperature is reduced so that the system may equilibrate at the new temperature. In order to determine the value of the new temperature we make use of an adaptive technique also due to Huang [44]. Their goal is to ensure that the average cost of the system decreases at a uniform rate. Formally, the slope of the *annealing curve*, a plot of the average cost versus the logarithm of the temperature, is to be held constant. The choice of a target slope is made by requiring that a state of quasi-equilibrium be maintained across temperature changes. For this to hold, they require that the expected decrease in average cost be less than the standard deviation of the cost: $\Delta C = -\lambda\sigma$. Here $\lambda \leq 1$ is the second of our manual tuning parameters (a value of 0.7 is suggested). From these requirements the authors derive the following expression for the new temperature T_{i+1} based on the current temperature T_i

$$T_{i+1} = T_i \cdot \exp\left(-\frac{\lambda T_i}{\sigma}\right) \tag{35}$$

As a practical matter the ratio T_{i+1}/T_i must be given a lower bound, typically 0.5, to prevent drastic cooling at high temperatures where σ is extremely large.

Outer Loop Criterion (Frozen Condition Detection) and Greedy Descent

The "outer loop" criterion in the generic simulated annealing algorithm determines when the algorithm can be terminated. Following from the physical analogy this is often called the **frozen condition** as it marks the temperature at which the energy has reached its apparent minimum. This would imply that a local minimum of the cost function has been found and the temperature is too low for the algorithm to climb out of it.

The frozen condition is commonly detected either by stopping at a fixed temperature [102] or by observing the average or equilibrium cost over three or four consecutive temperature steps and terminating when no change is observed [17]. Huang et al [44] adopt a different approach. At each equilibrium point, they compute the difference between the maximum cost and the minimum cost seen over the current temperature. If this is equal to the change in cost seen by a single accepted move, they observe that the costs of all of the reachable states are of comparable costs and there is no need to use simulated annealing. Instead they terminate the algorithm and switch to a standard "greedy" random selection algorithm.

One could extend the above argument and observe that, sufficiently near a local minimum, the cost function over the reachable states will become convex, meaning that the minimum can be found with simple greedy descent, and hill climbing is no longer necessary. In fact, random greedy search may not be the most efficient method for performing this greedy descent. If one's cost function is differentiable, or at least relatively smooth, it may be better to make use of more sophisticated methods for nonlinear gradient descent [98].

In effect Huang et al are simply lowering the temperature to zero when the system freezes, but otherwise they are still using the same algorithm. Their method is equivalent to the more traditional approach of continuing to lower the temperature as normal until the true local minimum if found. Therefore, we use the simple heuristic from Cohn [17] discussed above. We observe the equilibrium cost over four successive temperature steps and terminate when no change is observed.

After the frozen condition has been met we terminate simulated annealing and begin an optional phase of greedy descent. The previous four temperature steps have demonstrated that random greedy descent has become inefficient. However, to be absolutely certain that we have indeed reached the local minimum, we exhaustively try every possible unit distance move. We try every possible pairwise swap and every possible rectangle rotation. This takes m^3 iterations, the same number as the worst-case upper bound over each temperature step.

Move Selection and Range Limiting

There are many techniques for selecting moves from the move set. Central to these are the observation that some moves are more likely to be accepted than other moves, and this likelihood changes over time. Highly disruptive moves may be accepted readily at high temperatures, and they are very useful for exploring the search space, but they will almost always be rejected at lower temperatures. In addition, different subsets of the move set may have very different cardinalities. For example, in our case, there are $O(m^2)$ different pair-swap moves, but only $O(m)$ different object rotation moves. In this case it makes sense to select pair swap moves much more often.

An intelligent move selection heuristic can have a large impact on the efficiency of the simulated annealing algorithm and on the quality of the results which are obtained. The standard approach [102] has the programmer partition the move set into different classes and assign fixed selection probabilities to each class. To account for temperature dependence, these selection probabilities may be a function of the temperature.

One particularly useful schemes for controlling move selection is through the use of temperature dependent **range limits** [50]. For example, the *TimberWolf-3.2* standard cell placement tool [102] encloses each cell within a window, outside of which it is not permitted to move. Since large moves are much more disruptive than small moves, this window is reduced as the temperature decreases until only nearest neighbor moves within the same row are permitted. The *Koan* analog placement tool [17] discretizes move range into four categories: Extra-Large, Large, Medium, and Small. Each move range can then be assigned a different temperature dependent probability of success.

Static move selection approaches suffer from the disadvantage that they are very problem specific and require a lot of hand tuning. *Koan* [17] makes use of an adaptive scheme attributed to Hustin [46]. Here, each move class is given an initial static probability of success with a uniform distribution. During each temperature step the algorithm monitors the effectiveness and probability of success for each move class and, when equilibrium is reached, the selection probabilities are dynamically modified for the next temperature step. For each move type *t*, the contribution to the total cost is calculated as

$$\Delta\text{Cost}_t = \sum_{k=1}^{N_t} \Delta\text{Cost of } k_{\text{th}} \text{ accepted move of type } t \tag{36}$$

where N_t is the number of accepted moves of type *t* at temperature *T*. We can then calculate a quality factor for move type *t* as

$$Q_t = \frac{\Delta\text{Cost}_t}{N_t}. \tag{37}$$

This quality factor favors moves which were accepted very often, or even if accepted rarely, provided large cost increases. The probability for selection for each move type, P_t, is then calculated as follows:

$$P_t = \frac{Q_t}{\sum_k Q_k}. \tag{38}$$

This adaptive scheme is not easily defined in the presence of range limiting when the range limit has a continuous value. Thus discretized range classes, as used by *Koan*, are required. As in *Koan*, we define four different range classes for our translation and pair-swap moves. However, it is important to understand that the concept of movement range in our symbolic sequence pair model has a slightly less precise definition than in the direct models of *TimberWolf-3.2* and

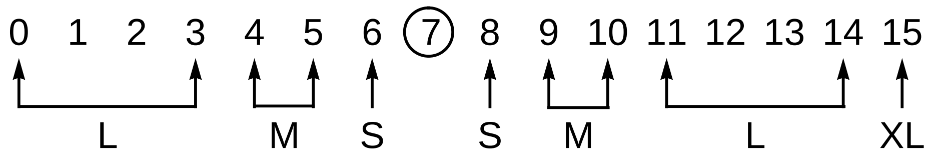


Figure 27: An example of sequence pair move ranges in a sequence of length 16

Koan. Distance values are restricted to the domain of the integer sequence pair slot numbers. Given a single rectangle, rectangles at successively larger index numbers will lie at physical distances which are non-decreasing, but the relationship is not linear as in the direct model. Furthermore, a pair of rectangles separated by a distance of, say, 5 index numbers may in fact lie closer to each other than a different pair of rectangles separated by a smaller symbolic distance of, say, 3 index numbers.

As in Koan we partition the sequence pair translation and pair-swap moves into four range classes. Below we enumerate the definitions for these classes. We assume that we are placing m objects (and thus the two sequences are of length m), and that we are discussing the distance d from an object at index i.

- Small: $d = 1$
- Medium: $2 \le d \le (m/4) - 1$
- Large: $(m/4) \le d \le (m/2) - 1$
- XLarge: $(m/2) \le d \le \max(i, m - i - 1)$

A demonstration of these sequence pair move ranges is shown in Figure 27. Here we show a sequence pair of length 16. The objects with the smallest possible move range are in the center at slots 7 and 8, we show the limits of the four different range classes for the object in slot 7. Note that the XLarge move is only possible in the positive direction.

Object translation moves are relatively straightforward to define in terms of these move ranges. The move generation algorithm randomly selects an object for translation, and randomly selects one of the four move ranges. It notes the current index number of the object and selects a new index that satisfies the move range constraint. Move ranges for pair swap moves are equally straightforward to define, with one complication. For pair-swap moves in a single sequence, we again select a random sequence index and a distance within the move range. The object at the two resulting indices are swapped. However, pair-swap moves which switch the items in both sequences are difficult to define in this manner. It would be somewhat time consuming to choose a pair of objects which obey the distance constraint in both sequences simultaneously. Therefore, simultaneous pair-swap move are only given an extra-large move class.

We have implemented the *Koan* scheme for adaptive move selection making use of our four-class move range for sequence pair translation and pair-swap moves. However, we have found that it can be very difficult to guarantee the numerical stability of this technique. Quality factors tend to become so polarized that the resulting probabilities become either 0% or 100%, strongly favoring one move type over the other. In addition, moves with extremely low probability may not be attempted enough times to allow the algorithm to gather a statistically significant quality factor value for use during the following time step.

To address the statistical significance concern Cohn [17] notes that some (unspecified) facility must be used to guarantee that a minimum number of moves of each type are performed at each temperature. Ochotta [82] makes use of Cohn's technique as well, but instead of re-calculating move probabilities at each temperature decrement, he breaks the annealing process up into fixed intervals, called statistical intervals, and begins each interval with a fixed number of moves of each type. As an alternative, we simply define a minimum allowable probability value for each move type. If any move probability falls below this range, we increase its probability value to this minimum and re-scale the remaining probabilities. For robustness, we also allow a user-selectable mode which simply assigns all move classes a uniform probability throughout the entire optimization.

3.4 Experimental Evaluation

In order to assess the feasibility and effectiveness of our optimal formulations of the sequence pair based placement optimization problem, as well as our heuristic simulated annealing formulation, we have conducted a number of experiments using prototype implementations of the branch-and-bound and simulated annealing algorithms. The objects being placed were simple rectangles without routing. The problems consists of artificial benchmarks constructed from rectangles of a randomly generated size. All experiments were conducted on a 170MHz Sun UltraSparc workstation with 128 megabytes of RAM.

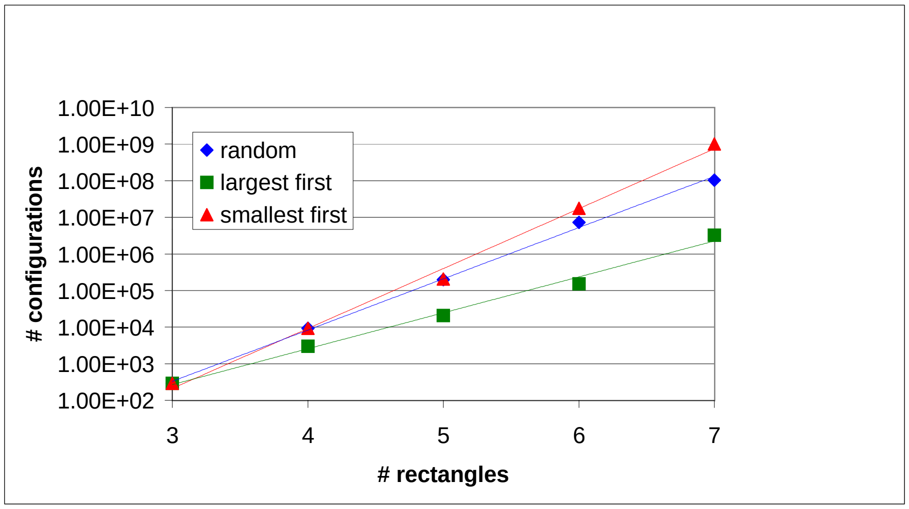


Figure 28: Branch and Bound run time experiments

3.4.1 Branch and Bound Algorithm Experiments

In examining the Branch and Bound algorithm our primary interest was in determining the number of rectangles that can be placed in a reasonable amount of time, and to investigate several different heuristics involved in the algorithm. In the first experiment we produced example problems using rectangles with randomly generated sizes (uniformly distributed between 1 and 100 units on a side). Problems of size 3, 4, 5, 6 and 7 rectangles were produced, with 10 examples of each size. We ran the algorithm using three different branching heuristics: 1) choose rectangles in largest-first order, 2) choose the rectangles in smallest-first order, and 3) choose the rectangles in a random order.

We would expect heuristic #1 to be the most effective, encouraging the algorithm to bound more quickly, and this is indeed what we saw. The data is graphed in Figure 28. Here we show, over the ten examples of each size, the average number of placements that were explored before the optimum solution was found and verified. The graph is plotted on a logarithmic scale and clearly shows the exponential nature of the problem complexity. For problems sizes on the order of 1–10 rectangles the DAG SSLP algorithm required approximately 1.0×10^{-4} seconds to evaluate each placement, so even with heuristic #1, problems with 7 rectangles took on the order of 1,000 seconds of CPU time, and we are doubtful that problems of size 8 could be solved in a reasonable amount of time.

Another question worth asking is how important it is to obtain a good initial upper bound.

If it is critical to begin with a very tight initial upper bound, it may be prudent to run a quick simulated annealing step to generate the bound before the Branch and Bound algorithm is started. We conducted an experiment in which we ran the Branch and Bound algorithm once to obtain the optimum solution. We then fed the cost of the optimum solution as a user-supplied (exact) upper-bound into a second run of the Branch and Bound algorithm and ran the example again. The random decision heuristic was used. We discovered that the Branch and Bound algorithm generally converges fairly rapidly to an upper bound which is close to the optimum, and little run time was saved. The data is shown below:

Table 3: Effect of Exact Upper Bound

# rectangles	no upper bound run time (sec.)	exact upper bound run time (sec.)
4	0.673	0.655
5	30.874	29.344
6	53.478	49.589
7	5,872.3	5,625.7

Our final experiment was intended to examine how effectively the lower bounding criterion is operating. It is interesting to examine the number of times the algorithm bounds itself at each depth in the decision tree. Obviously the bounding operation prunes significantly more of the solution space at smaller tree depths. The experiment in Table 4 was conducted with 7 rectangles and used the random decision heuristic. Recall that the size of the solution space is $2^n(n!)^2$ which, for $n = 7$, equals 3.251404×10^9. The algorithm ultimately explored 18,492,847 of these placements, a reduction of approximately two orders of magnitude, in 7,932.2 CPU seconds.

In Table 4, decision level 0 is the root, and decision level 6 would correspond to the leaves of the tree. The vast majority of pruning occurs at decision level 5, thought some does occur at levels as shallow as 3.

Clearly the upper limit of about 7 on the problem size does not suggest that the Branch and Bound algorithm will be a practical option in its current form. We have targeted designs of approximately 30 to 50 transistors as a reasonable goal, which would require an order of magnitude increase in the capability of the Branch and Bound algorithm. Possible ways of achieving this goal will be a significant topic of future work. As in [84] it is possible to recursively partition larger

Table 4: Bounding by Decision Level

Decision Level	# bounds
0	0
1	0
2	0
3	3,372
4	228,403
5	4,452,480
total	4,684,255

problems into problems of size 6 or 7, though it is unclear how the solution quality will be maintained as the locally optimal sub-problems are recombined.

3.4.2 Simulated Annealing Algorithm Experiments

The results of Section 3.4.1 indicate a lack of success in achieving global optimality for reasonable sized problems when using the branch and bound formulation. These results discouraged us from implementing the Integer Linear Programming formulation of the exact problem. It is our belief that the combinatorial size of the placement problem search space is fundamental to the problem itself, and it is unlikely that the ILP formulation would be much more successful in pruning the space. Therefore we concentrated our efforts on the implementation of the heuristic simulated annealing formulation.

Because of the stochastic nature of the simulated annealing algorithm, it is impossible to obtain exact and repeatable runtime information for a particular placement as with the branch-and-bound algorithm. Rather, one adjusts the tunable parameters of the annealing control algorithms in order to obtain acceptable results in an acceptable amount of time. In Figure 29(a) we show the results of a series of runtime experiments conducted with the annealing control algorithms discussed in Section 3.3.3.3. Benchmarks with increasing size were placed using the default annealing control parameter values of $\eta = 3$ and $\lambda = 0.7$. Ten randomly generated examples of each size were run and the average runtime observed. As shown in the figure, the control algorithms which we have adopted result in runtimes which are somewhat sub-exponential in the problem size. This result indicates that the simulated annealing algorithm does not free us from the expo-

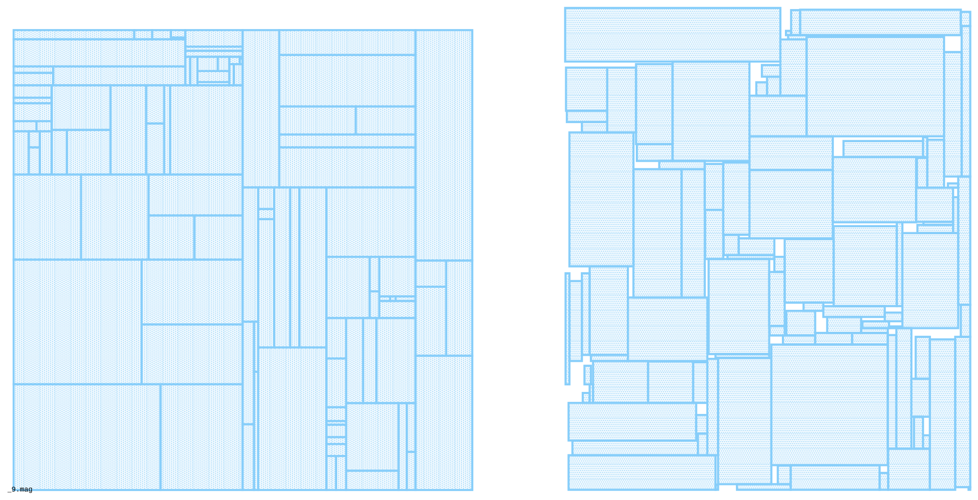
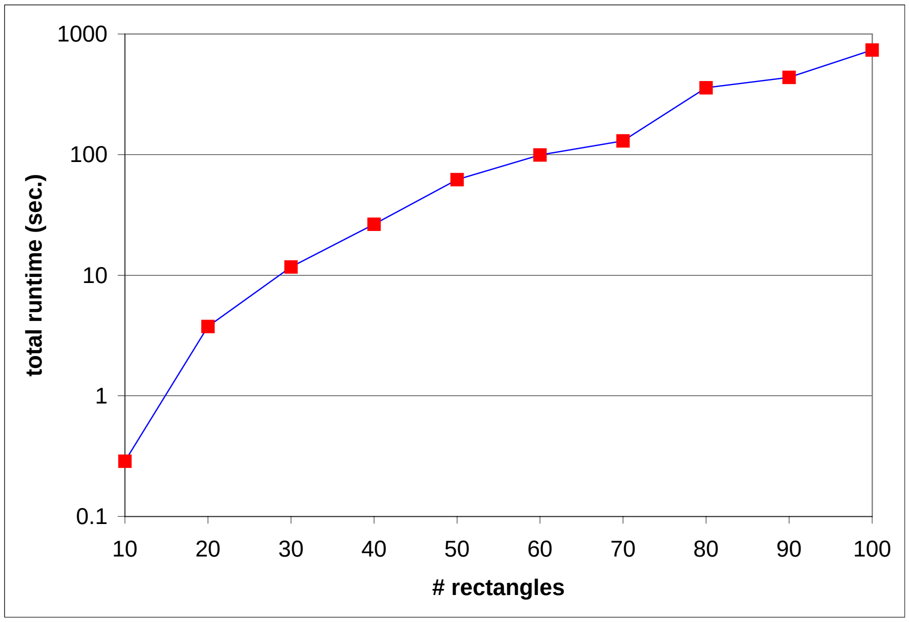

(a) optimal solution, area = 1.00000 (b) annealing solution, area = 1.05144

Figure 30: An example of a 100 rectangle perfect benchmark

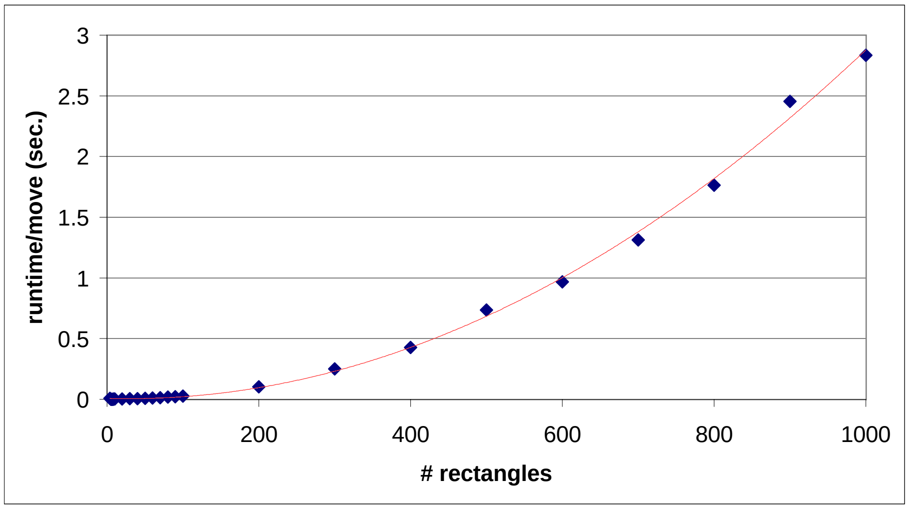
nential nature of the search space. However, the implementation presented here has reduced the rate of exponential growth to an extent that permits problems of several hundred objects to be solved.

It is also interesting to examine the impact of increasing problem size on the runtime of the simulated annealing inner loop, i.e. the execution time of a single move. The most expensive operation required to perform a move is the solution of the sequence pair constraint graph. Figure 29(a) shows a series of experiments conducted on randomly generated examples over a wide range of problem sizes. The CPU time per iteration was measured and averaged over 100 iterations for each example. The data shows an excellent fit to a quadratic curve, verifying the predicted $O(n^2)$ asymptotic complexity of the constraint graph SSLP algorithm.

While the simulated annealing algorithm is delivering reasonable execution times, it is natural to question the quality of the resulting solutions. In order to examine this question we constructed a set of "perfect" artificial benchmark problems with known optimal solutions. To obtain a benchmark consisting of *n* rectangles, an exact square with a normalized area of one is recursively sliced $n - 1$ times. At each step a randomly selected sub-rectangle is given a cut in a random direction (either horizontal or vertical) at a random coordinate on the selected axis. The cut coordinate is biased toward the center with a Gaussian distribution. An example of a perfect benchmark problem with 100 rectangles is shown in Figure 30. Figure 30(a) shows the optimal solution while Figure 30(b) demonstrates a solution obtained with our simulated annealing placement algorithm.



(a)simulated annealing total runtime



(b) simulated annealing runtime per move

Figure 29: Simulated annealing run time experiments

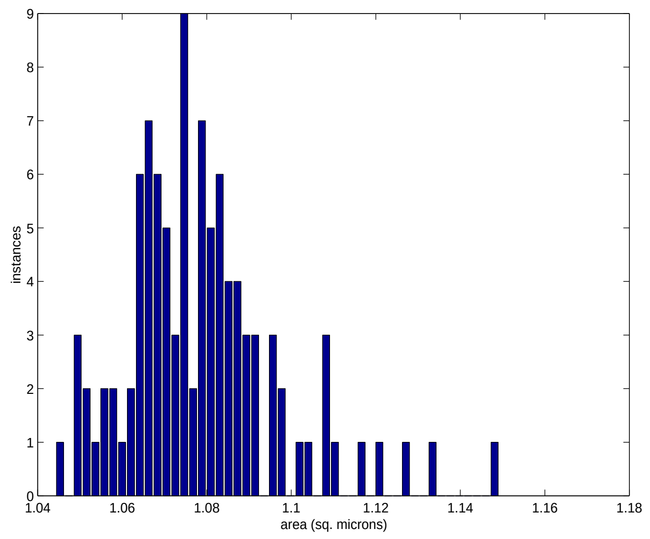


Figure 31: The area distribution of one hundred different solutions to the same randomly generated artificial problem.

Because the simulated annealing algorithm is based on a random stochastic process, it is likely that a different solution will be obtained each time that it is run. Naturally, the quality of the final frozen solution will be different each time as well. One method for judging the quality of the final solution is to study the distribution of solution cost over a number of separate optimization runs. In Figure 31 we show data taken with one hundred optimization runs over the same 100-rectangle perfect benchmark. This histogram is roughly gaussian in shape, indicating a fairly well behaved random process. Some statistics derived from this data are given below in Table 5. We

Table 5: "perfect" benchmark experiment statistics

Smallest Area	1.04422
Largest Area	1.14918
Mean Area	1.07868
Area Std. Dev.	1.84596×10^{-2}

can see that the area of the smallest solution is 1.86 standard deviations below the mean. This result suggests that it is generally advisable to run multiple optimization passes when using simulated annealing in order to obtain the best possible solution quality. However, it has been shown that it is not necessary to start the optimization from the beginning each time. Random restarts [85] from the intermediate temperature regime tend to result in the same statistical spread as restarts

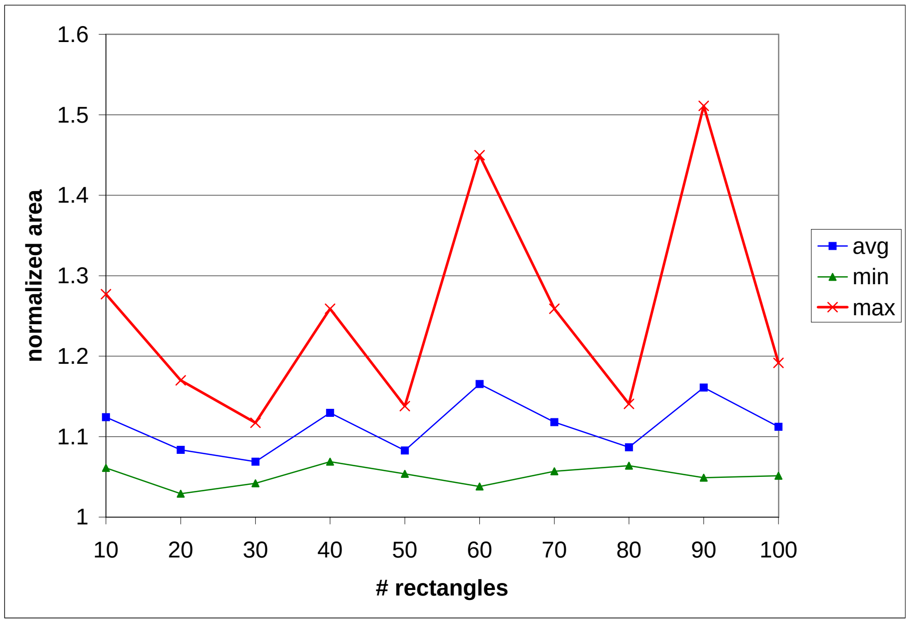


Figure 32: Simulated annealing "perfect" benchmark experiments

from an infinite temperature.

As a final experiment we conducted tests on artificial "perfect" benchmark problems of various sizes. Ten examples of each size, with problems ranging from ten to one hundred rectangles, were optimized using simulated annealing. The results of this experiment are given in Figure 32 where we show the minimum, maximum, and average placement area for each problem size. It is interesting to note that the area of the selected minimum solution appears to be relatively independent of the size of the problem. This result indicates that our annealing control mechanisms are performing reasonably well at adapting the execution time to the problem size. However, the wide variability in the area of the maximum solution does indicate a somewhat unstable stochastic process and emphasizes the need for multiple optimization restarts.

3.5 Summary

The subject of Chapter 3 was the symbolic sequence pair notation for the representation of the pairwise relative placement model. In the first half of this chapter we reviewed the sequence pair notation, which was developed by Murata et al [77], and elaborated at length on its implementation in a complete placement environment. We demonstrated how the sequence pair notation eliminates the infeasible placements from the search space and how we can take advantage of this when designing more efficient algorithms for the solution of the resulting constraint graph. In this context we introduced a new algorithm for the incremental solution of the Single Source Longest Path problem. The incremental nature of this problem occurs because of the fact that moves within the sequence pair solution space result in small local changes to the constraint graph, and this can be leveraged to avoid re-solving the entire graph from scratch.

The second half of this chapter was concerned with the implementation of a placement optimization framework built around the sequence pair notation. We developed two provably optimal approaches based on integer linear programming and branch-and-bound, respectively. We also discussed a simulated annealing approach which is based on previous work by Murata et al [77]. A set of pilot studies conducted with the branch and bound algorithm indicated that it would only be possible to obtain optimal solutions for problems of extremely small size, perhaps six to eight objects. For this reason we chose not to pursue the optimal approaches and instead concentrated on the simulated annealing implementation.

A sophisticated adaptive annealing control environment, used to control system cooling and move selection, we developed based on work by Cohn [17]. An extensive set of pilot studies verified the power of the simulated annealing implementation and its ability to obtain near optimal solutions for problems of one hundred objects or more. However, it was observed that statistical variance across multiple annealing runs suggests that multiple optimization trials will often be required in order to assure the location of a good local minimum.

CHAPTER 4
Transistor Level Micro-Placement and Routing

4.1 Methodology

In Section 1.3 of Chapter 1 we outlined the basic flow of our proposed methodology for two-dimensional digital cell synthesis. This flowchart, originally shown in Figure 5, is repeated in Figure 33. In a significant departure from traditional row-based cell synthesis techniques, our methodology adopts a very general model based on unconstrained placement and routing. Given a sized transistor netlist as input, in addition to the design rule database and a cell template, we proceed through four steps: static clustering, placement, routing, and optionally compaction.

In this chapter we present each of these steps in more detail. Section 4.2 reviews the theoretical framework behind transistor chaining, a concept which is critical in achieving dense layouts for digital circuits and a central issue in all stages of the methodology. Section 4.3 presents the details behind our transistor clustering algorithm. Section 4.4 discusses the problems which make detailed transistor-level placement, which we refer to as *micro-placement*, distinct from ordinary block level macro-placement. This section can be viewed as an extension to the generic placement algorithms developed in Chapter 3. We discuss our technique for dynamic transistor chaining, as well as our support for arbitrary geometry sharing outside of the transistor chains. We also address the details of the cell template, the placement cost function, and the routing model which is used in placement. The routing model addresses two separate issues: estimation of routing length, and estimation of routing area. We conclude with a brief discussion of the placement and compaction stages, both of which are performed with third-party tools.

4.2 Transistor Chaining

As can be seen in the manually designed mux-flipflop circuit shown in Figure 3 on Page 16, transistors are rarely placed in the layout as individual isolated objects. In order to achieve high

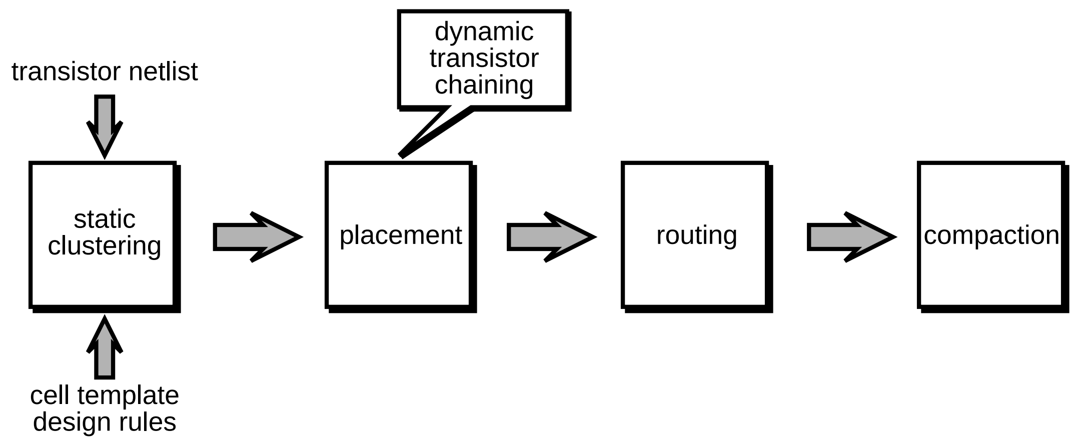


Figure 33: Proposed methodology for two-dimensional cell synthesis

density, most transistors are grouped into rows which share a single continuous strip of diffusion. Transistor chaining improves circuit density in two respects. First, because the sources or drains of adjacent transistors are overlapped and merged, the total size of the transistor group is reduced. Second, the need for a wire is eliminated as the electrical connection between adjacent transistors is formed through their shared diffusion contacts. This observation forms the basis of the row-based one and 1-1/2 dimensional placement techniques which we reviewed in Section 1.2.

The traditional row-based cell synthesis techniques place a highly stylized restriction on the appearance of the cell layout. All transistors must be placed in chains, and these chains must be placed in one or more parallel rows. (Recall that one-dimensional, or “dual-row” techniques allow only two rows, one consisting of all p-channel transistors and one consisting of all n-channel transistors, and these two rows must be duals of each other. The 1-1/2 dimensional techniques allow more than two, possibly non-dual, rows.) In order to achieve dense layouts for complex non-dual non-ratioed digital circuits, we wish to remove the restrictions imposed by these highly constrained layout styles. However, as we see in the mux-flipflop layout, we cannot abandon the concept of transistor chaining altogether.

In this section we review the theoretical concepts behind transistor chain optimization. We discuss an algorithm that finds a transistor ordering that minimizes the chain width. We also examine the issue of routing within the chains and how this effects the chain height.

4.2.1 Overview

Given a set of transistors that all share the same well (i.e. they are all of the same polarity and they all share the same body voltage), these transistors can be arranged in a single transistor chain. Without loss of generality, in this discussion we will assume that a transistor chain is oriented horizontally with the transistor gate terminals oriented vertically. Each transistor within a chain has two degrees of freedom:

1. the left-to-right orientation of the transistor within the chain (i.e. either source-gate-drain or drain-gate-source)
2. the relative position, or index, of the transistor within the chain

The central problem in the optimization of a transistor chain is the selection of an orientation for, and a linear ordering among, the transistors within the chain. We refer to a particular assignment to these variables as a transistor **chaining**. Both the width and height of a chain is dependent on the chaining which is selected.

A particular ordering affects the width of a chain in two respects. First, we observe that if two adjacent transistors share the same electrical node the two shared terminals can be merged so that their geometry is shared. If they do not share the same electrical node the two terminals may not overlap, and in addition they must be separated by the proper diffusion separation design rule, thus increasing the width of the cell. This increase in width is commonly referred to as a **diffusion break**. Second, we observe that if two diffusion shared terminals must be connected to a third terminal, either inside or outside of the chain, the adjacent polysilicon lines must be separated by a diffusion contact, but if no other connection is necessary the contact may be removed, thus decreasing the width of the chain.

The chaining selection also has the potential to affect the height of a transistor chain. The height of a chain is commonly defined as the number of routing tracks required to complete all electrical connections within the chain that are not made through transistor abutment. These intra-chain routes are commonly made using a channel routing model and different chainings may require different numbers of routing tracks.

As an example, in Figure 34 we show two different chainings for the same transistor schematic [73]. Both chains have one diffusion break, but chain 2 requires one fewer horizontal routing track than chain 1. Chain 2 is also slightly narrower because the shared node between transistors c and b can be made without a contact. We should point out that, in this case, chain 2 was actually obtained by modifying the schematic—the top-level series chain was reordered. The two circuits will have the same logical behavior, but may not be electrically equivalent. Chainings which

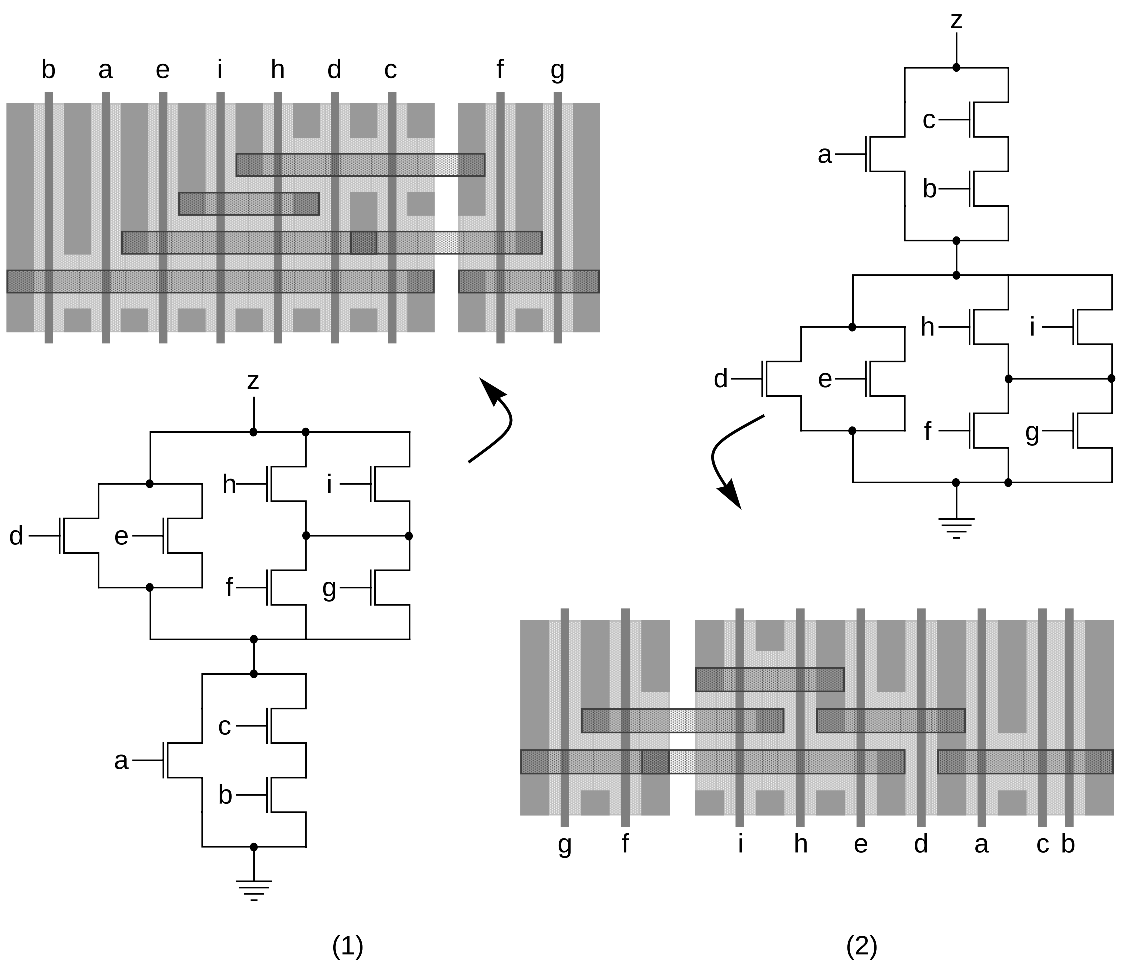


Figure 34: Two different chainings corresponding to the same logically equivalent circuit.

change the schematic should only be explored if permitted by the designer.

4.2.2 Terminology

Given the discussion of the above section, we now have the language to define more precisely several terms which will become important in the remainder of this chapter.

Definition 4-1: **Transistor Cluster**. A transistor cluster is an unordered set of transistors, all of the same polarity and body potential, which can be reached through an unbroken sequence of source or drain terminal connections. These are also often referred to as channel-connected components (CCCs). The schematics in Figure 34 are both transistor clusters.

Definition 4-2: **Transistor Chaining**. A transistor chaining is a mapping which assigns a unique ordering to the source-drain terminals of the transistors in a cluster. One can also view this as assigning a unique ordering and a source-drain orientation to the transistors themselves.

Definition 4-3: **Transistor Chain**. A transistor chain is a transistor cluster which has been assigned a chaining. This is a well defined physical structure for which a design-rule correct layout can be generated. Figure 34 shows two examples of transistor chains.

Definition 4-4: **Diffusion Break**. A diffusion break is a position within a transistor chain at which two neighboring transistors do not share a common electrical terminal. Therefore these two terminals will not be able to share geometry and must be separated by a diffusion-separation design rule distance.

Definition 4-5: **Transistor Sub-Chain**. A transistor sub-chain is a subset of a transistor chain which is free of diffusion breaks. Both chains in Figure 34 have a single diffusion break and can be decomposed into two sub-chains.

4.2.3 Transistor Chain Width Minimization

The traditional optimization problem associated with transistor chain width minimization seeks to find a chaining that minimizes the number of diffusion breaks within a chain. It is assumed that the chain width is proportional to the number of diffusion breaks, and we must therefore maximize the number of adjacent transistors which share a diffusion connection. This problem is easily posed as a linear-time graph optimization problem as follows:

We construct the so-called **diffusion graph** $G(V, E)$ by associating a graph vertex $v \in V$ with every electrical net which connects to a transistor source or drain terminal. An edge $e \in E$ is placed between two vertices (v_1, v_2) if there is a transistor spanning the two associated nets. The vertices are labelled with their corresponding net names, and the edges are labelled with the name of their associated transistor gate terminals. The diffusion graph is classified as a multigraph as there may exist multiple edges connecting the same two vertices. An example of a transistor schematic and its associated diffusion graph are shown in Figure 35.

Ideally, the optimal chaining solution would contain no diffusion breaks, and thus every transistor can merge with both of its neighbors (except the two transistors on the ends of the chain which have only a single neighbor.) In the corresponding diffusion graph, such a chaining would

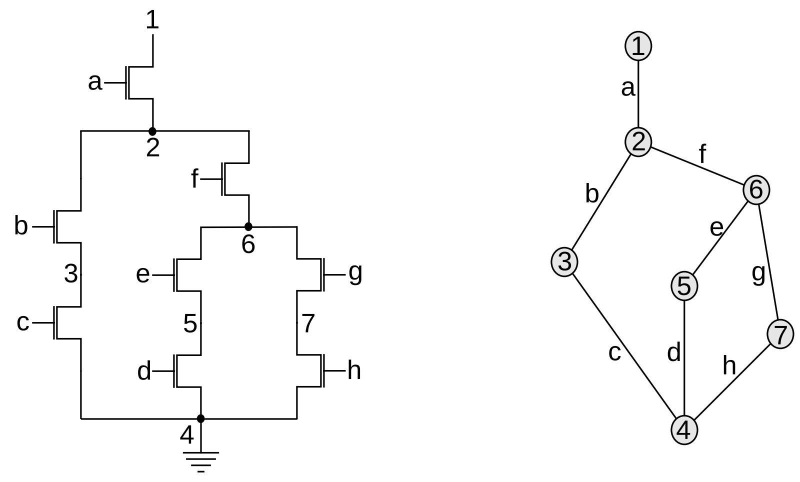


Figure 35: The diffusion graph corresponding to a diffusion connected set of transistors

correspond to the sequence of edges $(e_1, e_2, \ldots, e_n)$ visited in an *open Eulerian walk* through the graph. An *open* Eulerian walk is related to a *closed* Eulerian walk, which is defined as a *closed* walk which covers every edge exactly once and every vertex at least once. A multigraph for which a closed Eulerian walk exists is known as an *Eulerian* graph, and the term Eulerian walk is usually taken to mean a *closed* Eulerian walk. It can be shown [24] that a multigraph *G* is Eulerian if and only if

1. *G* is connected, and
2. all vertices in *G* have even degree.

An *open* Eulerian walk relaxes the restriction that the walk be closed, and the necessary and sufficient conditions for the existence of an open Eulerian walk are somewhat relaxed as well:

1. *G* is connected, and
2. Exactly two of the vertices in *G* have odd degree.

If zero vertices in *G* have odd degree a closed Eulerian walk exists, and the walk may start at any vertex. However, if two vertices have odd degree, an open walk still exists, but the walk must begin at one of the odd degree vertices and end at the other. Since there is no restriction that the two terminals at the ends of a transistor chain share the same electrical net, the existence of an open Eulerian walk on the chain's diffusion graph is sufficient to ensure that there will be no diffusion breaks.

A simple recursive algorithm exists which will locate a closed Eulerian trail in a multigraph, if such a trail exists, in $O(V + E)$ time. The following pseudo-code is taken from Papadimitriou & Steiglitz [86]. It locates a closed Eulerian walk in the graph G beginning and ending at the vertex v_α.

```
EULER($v_\alpha$)
1    **if** $v_\alpha$ has no edges
2        **return** $v_\alpha$
3    starting from $v_\alpha$ create a walk of $G$, never
4    visiting the same edge twice, until $v_\alpha$ is reached
5    again; let [$v_\alpha$,$v_1$],...,[$v_n$,$v_\alpha$] be this walk;
6    delete [$v_\alpha$,$v_1$],...,[$v_n$,$v_\alpha$] from $G$;
7    **return** (EULER($v_\alpha$),EULER($v_1$),...,EULER($v_n$),$v_\alpha$)
```

We can see that, if a particular vertex has degree greater than two, the algorithm has to make an arbitrary choice at lines 3–4 concerning which edge to follow away from a vertex. Thus, unless every vertex has degree two, there are several possible Eulerian walks, and therefore several possible minimum width transistor chains, which cover the same Eulerian graph. This choice is usually made randomly, as it has no significance unless we are trying to optimize some secondary condition related to the graph. In the problem of transistor chain optimization we usually seek to find the set of minimum width solutions which also minimize the height. We will discuss this in the following section, Section 4.2.4.

If an open Eulerian walk does not exist in the diffusion graph, one or more diffusion breaks will be required. In such a situation, the chain will correspond to the sequences of edges in an *ordered set* of open Eulerian walks,

$$(e_1, e_2, \dots e_n), (e_p, e_{p+1}, \dots, e_{p+q}), \dots, (e_r, e_{r+1}, \dots, e_{r+s}) \tag{39}$$

in which $e_n \neq e_p$ etc., which together cover every edge exactly once and every vertex at least once. A diffusion break exists in the transistor chain at each discontinuity in the covering set of open Eulerian walks.

The traditional method used to locate the set of covering open Eulerian walks is to transform the non-Eulerian diffusion graph into an Eulerian graph by adding extra *artificial* edges. The classic approach taken by Uehara & VanCleemput [119] makes use of the observation that there will always be an even number of odd-degree vertices, and thus an Eulerian graph can be constructed by adding $n/2$ artificial edges connecting random pairs of the n odd-degree vertices so as to make them all of even-degree. The traversal of one of the artificial edges in the resulting closed Eulerian walk would then corresponds to a diffusion break in the corresponding transistor

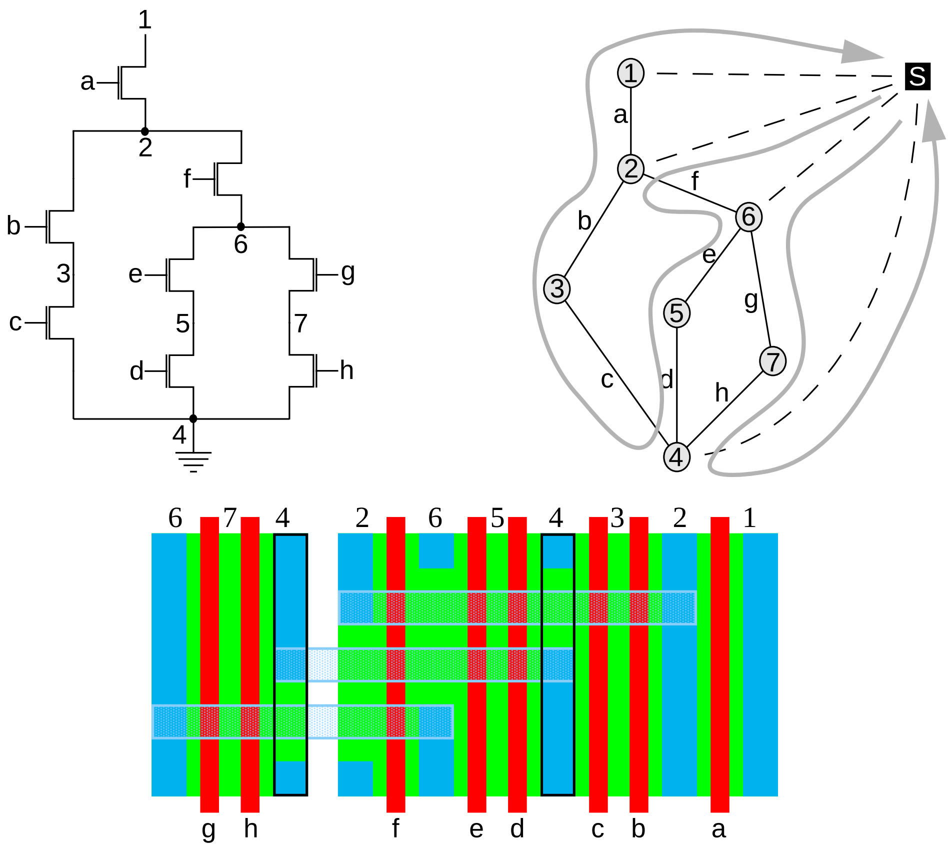


Figure 36: A minimum width solution located using Basaran's super-vertex technique

chain. It is easy to see that any set of covering open Euler paths will have the same number of diffusion breaks, and that this number is equal to two less than the number of odd-degree vertices. This is the minimum number of diffusion breaks in the transistor chain, and thus the set of all minimum width layouts correspond to the set of all possible covering open Euler paths.

However, it has been shown by Basaran [5] that the method of Uehara & VanCleemput does not allow the procedure `EULER()` to generate every possible transistor chaining. The choice of which odd-degree vertices are connected in the new artificial Eulerian graph will limit the set of chainings which can be produced. As an alternative Basaran introduced a new technique which we have adopted. This technique is capable of generating every possible minimum width transistor chain. In order to make the graph Eulerian an artificial vertex, the **super-vertex**, is added and artificial edges are drawn connecting it to all odd degree vertices (recall that there must be an even number of these). All minimum width walks must begin and end at the super-vertex, though any internal vertex may be used if the input graph was already Eulerian. Every visit to the super-vertex (except the first and last) translates into a diffusion break in the corresponding transistor chain.

An example of Basaran’s technique is shown in Figure 36. Here the diffusion graph from Figure 35 is augmented with the super-vertex (S) and a set of artificial edges (shown with dashed lines). A minimum width solution with one diffusion break is located by applying algorithm `EULER()` to the diffusion graph. The algorithm first traced a closed Euler walk

$$S \rightarrow 6 \rightarrow 7 \rightarrow 4 \rightarrow S$$

followed by a second closed Euler walk

$$S \rightarrow 2 \rightarrow 6 \rightarrow 5 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1 \rightarrow S$$

This walk results in the transistor chain shown.

4.2.4 Transistor Chain Height Minimization

In formulating the transistor chain optimization problem, most authors express the problem as one of locating the minimum “height” solution from among the set of minimum “width” solutions. In this context, the “width” of the chain is proportional to the number of diffusion breaks, and the “height” is proportional to the number of routing tracks which are required to complete the intra-chain routing. Figure 37 shows two different minimum width transistor chains which have different “heights”. The upper chain requires three routing tracks to complete the routing, while the lower chain requires only two tracks.

In the optimization of complementary transistor circuits[1] Maziasz & Hayes [73] show a polynomial time algorithm for width minimization, but Chakravarty et al [11] prove that the corresponding height minimization problem is NP-Complete. Lefebvre & Chan [60] provide an example of a circuit for which the minimum height solution is not of minimum width, and discuss an algorithm which allows one to trade chain width for height. As we gave seen in the previous section, width minimization for non-complementary transistor chains can be done in linear time. The corresponding height minimization problem is unfortunately still non-polynomial—all minimum width solutions must be enumerated and the one with minimum height selected. Basaran [5] developed an optimal height minimization algorithm based on integer linear programming, and a heuristic solution using a simulated annealing algorithm.

It is traditional in the literature to assume that chain height is proportional to the number of

1. Recall that complementary circuits consist of two dual transistor chains, and their optimization requires the location of a pair of dual Euler walks in the N-FET and N-FET diffusion graphs.

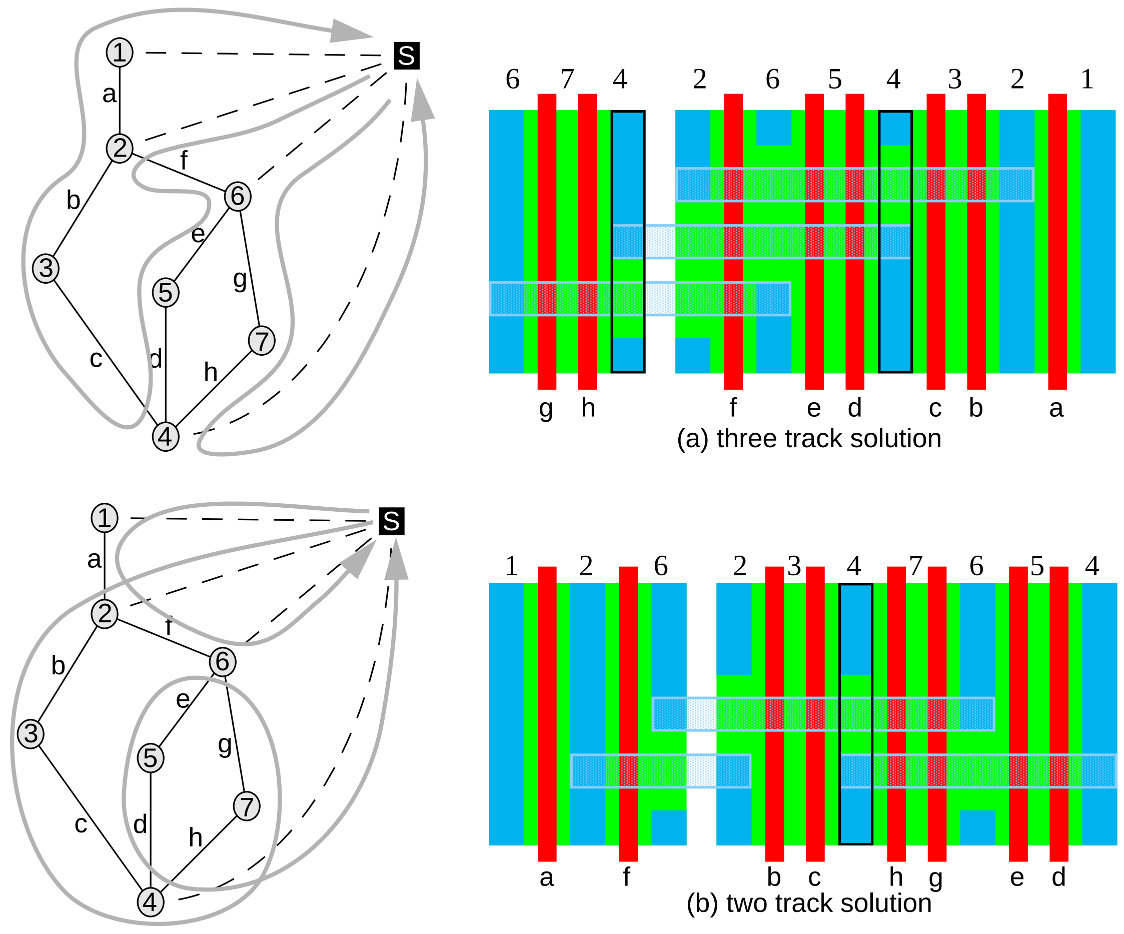
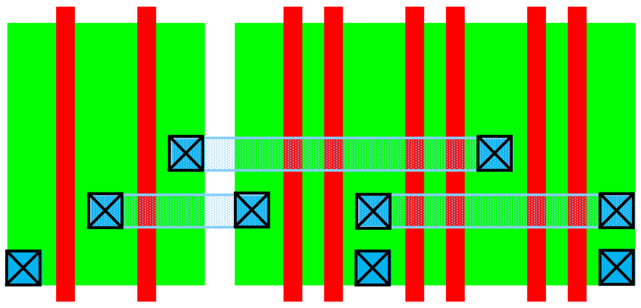
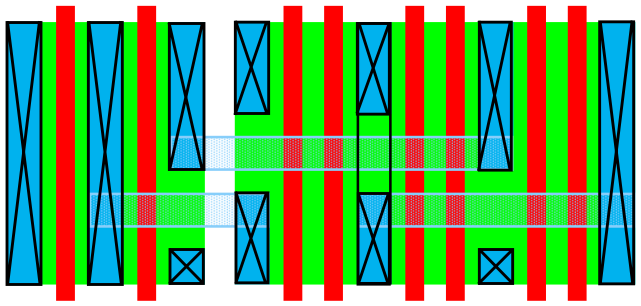


Figure 37: Two different chainings with different maximum route density

required routing tracks. Uehara & Vancleemput [119] and Maziasz & Hayes [73], among others, size the width of the diffusion strip to equal the routing requirements. We would like to point out, however, that the transistor chain routing problem can be formulated in a number of different ways, and the proportionality between track density and chain height is not always so clear-cut. For example, in Figure 37 we show two solutions which require a different maximum track density. However, we show the two diffusion blocks with the same height. In a realistic definition of the problem, the height of the diffusion region is determined by the transistor channel widths as specified in the schematic. In this example routing is performed over the diffusion region, and the chain height will only increase if the routing cannot be accomplished in the maximum number of available tracks and some routes must be completed outside of the diffusion region.

Of primary concern in the formulation of the transistor chain routing problem is the fact


(a) no strapping


(b) partial strapping

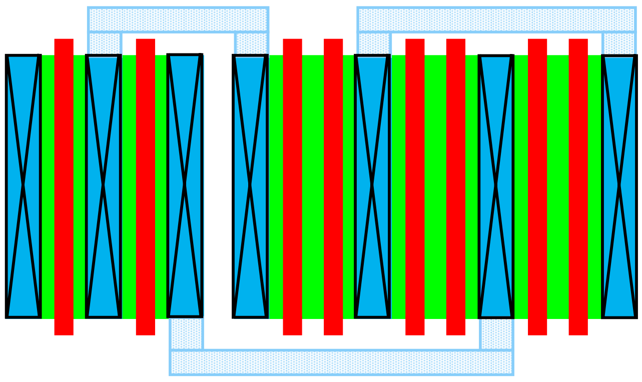
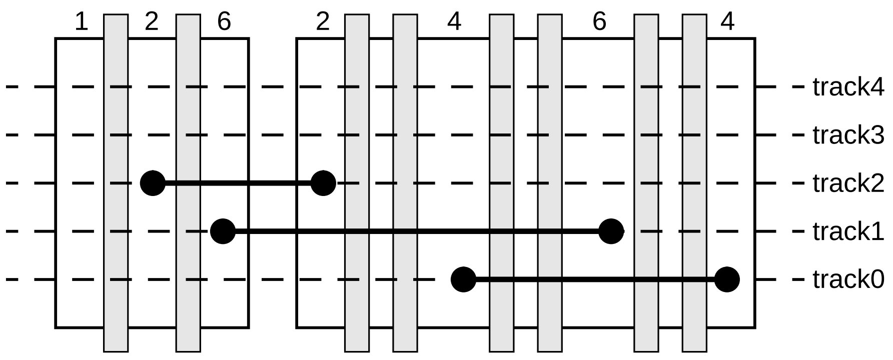
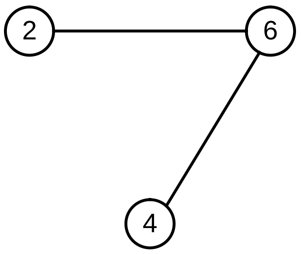

(c) full strapping

Figure 38: Transistor source/drain contact strapping

that the fabrication technology may dictate constraints on the style used for transistor source-drain connections. Figure 38 demonstrates three different routing models which differ by the degree of source-drain contact strapping required. If the fabrication technology provides a self-aligned silicide layer (*salicide*), or some other form of *local interconnect* (LI), the source-drain terminals will have very low resistivity, and a single minimum sized diffusion contact is all that is required. In this case no strapping is needed, as in Figure 38(a), and local routing may take place over the cell.



(a) overcell routing model



(b) interval graph or "track graph"

Figure 39: Routing model for unstrapped and partially strapped routing styles

If no low resistivity local interconnect is available, full strapping may be required and the chain routing must be pushed external to the chain, as in Figure 38(c). However, in this latter case, it is often still acceptable to route over the chain as long as partial strapping is done to guarantee some maximum uncontacted distance, as in Figure 38(b). We will deal with each of these cases below.

No Strapping

When diffusion strapping is not required, transistor chain routing can be treated as a special case of channel routing in which no vertical constraints exist. The problem is modeled using an interval graph defined as follows. As in Figure 39, the diffusion area is divided into separate horizontal tracks defined by the wiring pitch. The span of each net defines a horizontal interval over one of these tracks. We construct a graph $G = (V, E)$ named the **track graph** in which the vertices $v \in V$ represent the nets to be routed. An edge $e \in E$ exists between two vertices if the intervals of the two nets overlap. Chakravarty et al [11] pose the corresponding height minimization problem as one of assigning a valid coloring $c(v)$ to each vertex such that if the edge $(u, v) \in E$ then $c(u) \neq c(v)$. The minimum number of distinct colors which are required to color G is the chromatic number of the graph. If we take the track number to be vertex color, then the minimum number of tracks is equal to the chromatic number. According to Sherwani [106, pp. 112–113], since interval graphs are perfect, their chromatic number is the same as the cardinality of the maximum clique in the interval graph. The algorithm below can be used to find the cardinality of the maximum clique in $O(V + E)$ time.

```
MAX-CLIQUE(I)
1     SORT-INTERVAL(I,A)
2     clique ← 0
3     max-clique ← 0
4     for i ← 1 to 2n
5         if A[i] = L
6             clique ← clique + 1
7             if clique > max-clique
8                 max-clique ← clique
9         else
10            clique ← clique - 1
11    return max-clique
```

Algorithm MAX-CLIQUE() takes as input $I = (i_1, i_2, \ldots, i_n)$ which is a set of intervals. The routine SORT-INTERVAL() builds an array A of dimension $2n$ which sorts the intervals according to their endpoint coordinates. $A[i]$ is equal to “L” if the corresponding endpoint is the left endpoint of an interval, and it is equal to “R” otherwise.

When a transistor chain is routed in the unstrapped style, the chain height is only affected if the chromatic number of the track graph is greater than the number of available tracks. Any routing solution which satisfies this bound is essentially equivalent, except perhaps from the standpoint of yield or signal delay. If the intra-cell routing resources are exceeded, the unrouted nets must be routed outside of the diffusion region, a problem which is equivalent to the fully-strapped case described next.

Full Strapping

The routing problem in the presence of full diffusion strapping is more difficult than the unstrapped case. This is because of the presence of vertical constraints due to the vertical routing segments which are required to bring the nets to the correct track. If single layer routing is desired, this problem is a variant of the classical Single Row Routing Problem (SRRP).

As defined by Sherwani [106, pp. 267–274], an instance of SRRP is described by a set of two-terminal or multi-terminal nets defined on a set of even spaced terminals on a real line called

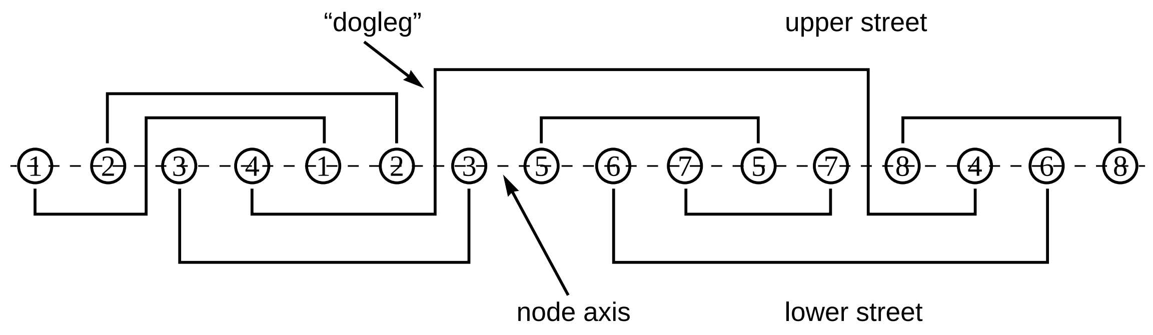


Figure 40: The Single Row Routing Problem (example from Sherwani [106] page 267.)

the *node-axis*. The goal of SRRP is to realize the interconnection of the nets by means of non-crossing paths. Only a single routing layer is used, and no horizontal route is permitted to cross the same terminal more than once (i.e. no backward moves are allowed.)

The example in Figure 40 shows a problem with 8 two-terminal nets. The routing region is divided into two halves, the "upper street" above the node axis, and the "lower street" below the node axis. Obviously, vertical and horizontal routing segments cannot cross. In the general version of the problem, a net may cross from one street to the other by routing between two nodes with the use of a "dogleg" connection. The optimization problem is usually formulated to minimize the maximum or sum of the number of upper street and lower street routing tracks required. We may also wish to minimize the number of dogleg routes as that may increase the width.

The algorithms used for the optimization of SRRP are too complex to review here. The reader is referred to Sherwani [106] for a detailed treatment. In the case of the fully strapped transistor chain routing problem, the SRRP problem is somewhat constrained. Most notably, dogleg connections cannot be permitted, except perhaps between two nodes separated by a diffusion gap. Without the freedom to use doglegs the problem may well turn out to be infeasible without the use of a second routing layer (polysilicon or second-level metal.)

The case of fully strapped routing conforms more closely to the traditional definition of the chain height minimization problem. Each new required routing track contributes to the height of the chain.

Partial Strapping

In the presence of partial strapping, the transistor chain routing problem is equivalent to the unstrapped case, with the addition of constraints on the maximum amount of unstrapped diffu-

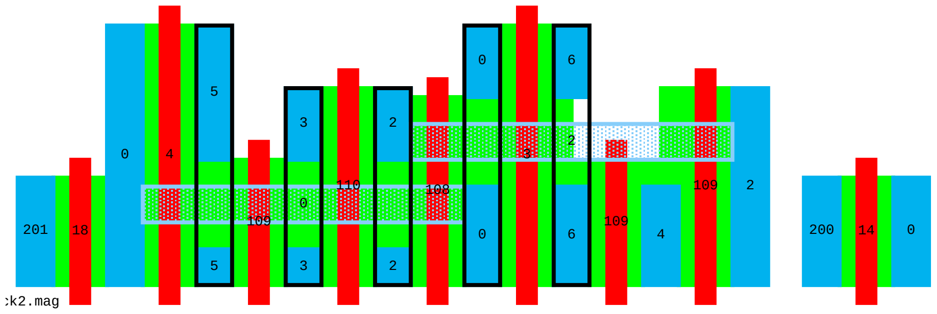


Figure 41: A transistor chain with non-uniform transistor channel widths

sion. Guan & Sechen [33] have developed a heuristic algorithm which is essentially equivalent to the solution of dogleg-free SRRP.

4.2.5 Complications

In this section we discuss a number of factors which may complicate the transistor chain optimization problem. The first of these is the case of transistor chain routing in which the transistor widths (i.e. the chain height) are not uniform. In the class of circuits for which our methodology is targeted, transistors are usually sized to meet specific power and/or speed goals, and are distinctly non uniform in channel width. Figure 41 shows an example of such a chain. Routing problems of this type are related to the problem of Irregular Channel Routing. Lengauer [65, pp. 462–465] provides a short review of this topic. The detailed router in the *LiB* 1-dimensional cell compiler uses a general-purpose maze router with ripup-and-reroute capability to route such irregular regions. We will discuss our solution to this problem in Section 4.2.7 when we discuss our chain geometry generators.

Another issue which may be of concern is the electrical optimization of the transistor chains. The parasitic capacitance on a net is reduced when the connection is made through geometry sharing, so it may be preferable to bias the merging decisions in favor of nets on the critical path. Though the intra-chain routes are short, different chainings will have different routing solutions which may effect routing parasitics as well. This topic has been treated extensively in the analog cell synthesis literature: see for example [5,69]. In addition, as we mentioned previously, some routing solutions may be more preferable from a yield or reliability perspective.

4.2.6 Static vs. Dynamic Transistor Chain optimization

Our methodology, shown in Figure 33, shows transistor chain optimization as a dynamic process embedded within the placement step. This is in contrast to the work of previous authors who have considered the two-dimensional cell synthesis problem. Basaran [5] uses simulated annealing to statically optimize the height of a transistor chain as part of a proposed cell synthesis flow. Fukui et al [28] discussed a tool which also uses simulated annealing to discover good groupings of transistors into diffusion shared chains, and then performs a greedy exploration of a virtual-grid slicing floorplan to locate a 2-dimensional placement. In a more recent work by the same group, Saika & Fukui et al [95] present a second tool which operates by statically grouping transistors into maximally sized series chains, then locating a high quality 1-dimensional solution in order to form more complex chains. A simulated annealing algorithm is then used to modify this linear ordering by placing the diffusion-connected groups onto a 2-dimensional virtual grid.

The difficulty with performing static transistor chain optimization, within a complex circuit, prior to the placement step is that the optimization cost function is unable to account for the effect of external routing. The ordering of the gate input nets can have a profound impact on the quality of polysilicon routing within the cell. In addition, the chain output node(s), and any multi-terminal input nodes and intermediate nodes within the chain, must be routed externally in metal.

In the transistor chain width and height minimization problems, there are often many equivalent solutions. If the optimization of the transistor chains is performed dynamically as part of the placement step, these equivalent solutions can be explored in the context of their impact on the cell-level routing cost function. Our approach provides a simple and elegant method for coupling these two previously decoupled optimization steps. We will discuss our dynamic chain optimization technique in more detail in Section 4.4.2.

4.2.7 Transistor Chain Geometry Generator

Our dynamic approach to the transistor chain optimization problem requires us to be able to generate design-rule correct chain geometries, including routing, quickly and efficiently within the inner loop of the placement step. In this section we describe the specific conventions and solutions which we have adopted in the implementation of our transistor chain geometry generator.

As input, the geometry generator takes the technology design rules, the transistor-level

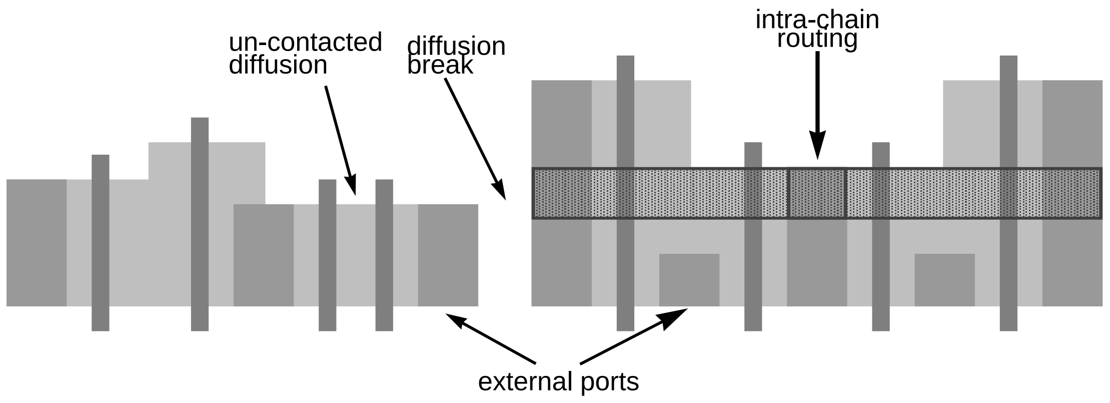


Figure 42: Layout produced by the chain generator

schematic of the chain, and the location and orientation of each transistor within the chain. In Figure 42 we show the output of our chain generator for a fairly complex chain. Our generator currently supports chains with internal diffusion breaks, as well as adjacent uncontacted transistors.

In order to address the problem of chains with a non-uniform width, we adopt the following convention. Oddly sized transistors are aligned along the bottom edge of the chain. All nets which require external routing, either because they must be routed externally to the chain, or because the intra-chain router could not complete the route internally and they spilled into the external channel, are given a port along this bottom edge. In order to enable an escape path in both directions we also place a contact on the top track of each transistor—if that track is not blocked by a net on a higher routing track used by taller transistors. We note that it may be beneficial to relax our restriction that forces all transistors to be aligned along the bottom edge. Better routing solutions may be possible if the transistors were allowed to slide vertically into any position. However, any solution must guarantee at least one escape path for each external net. The efficient optimization of this refined problem poses an interesting unsolved problem.

Intra-chain routing is performed using the `MAX-CLIQUE()` algorithm from Section 4.2.4. which runs in linear time. Routes are assigned from the bottom track toward the top, though the first routing track is only utilized if the route segment will not block any required external ports. Partial diffusion contact strapping is currently supported to a limited extent—all regions of transistor source/drain area which are not overlapped by routing are filled with diffusion contact material in order to reduce their resistance. However, we do not currently support constraints on the percentage of uncontacted diffusion. Any routes which cannot be completed within the intra-cell region are given external contacts and handled by the cell-level router at the completion of placement. As we discuss in Section 4.4.6.1, these spill routes figure into the global placement optimization cost function, and hence the dynamic chain optimization process attempts to minimize their impact along with the true external nets. For the case of full strapping, all routing is handled by the cell-level detail router.

4.3 Transistor Clustering

In the first step of the methodology in Figure 33, transistor clustering, the task is to determine the optimal number of transistor chains and which transistor(s) belongs in each chain. At this stage very little information is available, only transistor sizes and their connectivity, so the choices can only be heuristic in nature. Figure 43 shows the clustering used in the manually designed CGaAs Mux-FlipFlop circuit, our running example from Figure 3 on Page 16.

Authors in previous works have used a wide variety of heuristics at this stage. Fukui et al [28] use simulated annealing. In Saika & Fukui et al [95] only simple chains made up of maximum length series chains are created and larger chains are formed during the 1D optimization step. It is also fairly common, especially in 1-dimensional tools, to approach clustering by performing logic gate recognition, as in Picasso-II [64].

It is our opinion that, because of the heuristic nature of the decisions made at this stage, it makes little sense to devote a large amount of effort to determining, a priori, any sort of “optimal” static clustering. Instead, our goal for this stage is to transfer as much flexibility in the clustering decisions as possible to the dynamic placement stage.

We have implemented a hybrid approach that performs a rough large-scale static clustering of the circuit graph into channel connected components (CCCs) with a user-supplied upper size limit. As we discussed in Section 4.2.2, each **cluster** can be formed into one or more different transistor **chains** by assigning it a **chaining**. Furthermore, these chains may be partitioned at the locations of their **diffusion breaks** into one or more **sub-chains**. A diffusion break represent the location of a gap in the chain’s diffusion strip, any electrical connections which must span this gap must be performed with a strip of metal. There is no reason to constrain the sub-chains to remain aligned in the layout, as a more optimal arrangement may be found if they are allowed to move freely. Therefore it is these sub-chains which form the atomic units that will be placed during the placement step.

Our approach allows the placement step some flexibility to dynamically alter the circuit

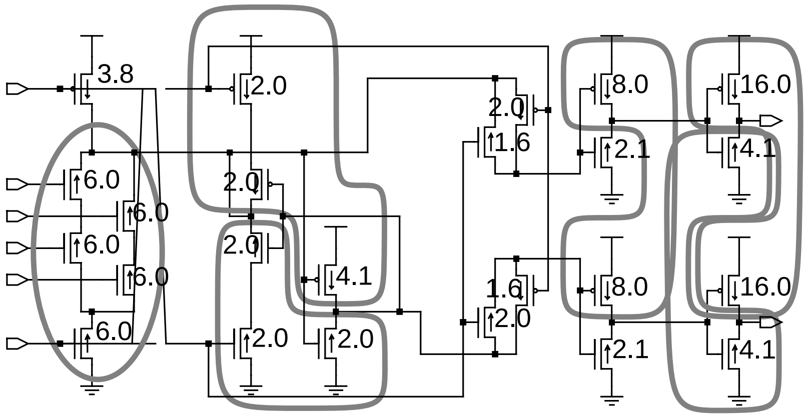


Figure 43: The clustering step. The schematic implements the CGaAs Mux-FlipFlop of Figure 3, and shows the clustering used in the manual design. All transistor lengths are 0.5 microns.

clustering in order to better optimize the placement: different orderings of the transistors in the chains can be used to induce different clusterings in the individual sub-chains. The sizes of the different sub-chains may change as transistors migrate from one sub-chain to another. However, since all minimum width chainings will have the same number of diffusion breaks (see Section 4.2.3), the number of sub-chains in each chain will stay the same. We will elaborate on this further in Section 4.4.2.

We begin the static clustering step by grouping the transistors into two sets, one consisting of all n-channel devices, and one consisting of all p-channel devices. Each of these sets are then partitioned using the function `FORM_MAX_CLUSTERS(`*T*`)`, shown below, into maximum-sized diffusion-connected sub-sets.

```
FORM-MAX-CLUSTERS(T)
1    i ← 1
2    for t1 ← T[1] to T[length[T]]
3        if visited[t1] = FALSE
4            visited[t1] ← TRUE
5            PUSH(t1)
6            j ← 1
7            while t2 ← POP()
8                PROCESS-NEIGHBOR(source[t2])
9                PROCESS-NEIGHBOR(drain[t2])
10               cluster[i][j] ← t2
11               j ← j + 1
12       i ← i + 1
```

This algorithm is called with the argument *T*, a list of transistors to be partitioned. The first transistor is pushed on a chain and marked “visited.” While the chain is not empty we pop the first transistor off the chain and process any transistors which are connected to its source and drain terminals using the routine `PROCESS-NEIGHBOR(n)`, possibly visiting those neighbors and pushing them on the chain as well. As transistors are popped off the chain we add then to the current cluster. When the chain becomes empty we search through the input list for the next unvisited transistor and begin a new cluster.

The routine `PROCESS-NEIGHBOR(n)`, shown below, simply examines each of the transistors which are sinks on the supplied electrical net. If the net connects to the source or drain (i.e. not the gate) of the sink, and that sink has not already been visited, it is marked “visited” and pushed onto the chain.

```
PROCESS-NEIGHBOR(n)
1    s ← sinks[n]
2    for t ← s[1] to s[length[s]]
3        if gate[t] ≠ n and visited[t] = FALSE
4            if CHECK-VARIATION(t)
5                visited[t] ← TRUE
6                PUSH(t)
```

Lines 4–6 in the `PROCESS-NEIGHBOR(n)` routine represents an optional feature which enhances the size uniformity among the transistors in a particular cluster. The function `CHECK-VARIATION(t)` keeps track of the range, median, and variation (defined below in Equation 40) of the transistor widths in a particular cluster. If the width of transistor t falls outside of this range, and if the new width variation falls outside a user-supplied threshold (say 50%), transistor t will not be pushed onto the stack.

$$\begin{aligned} range &= maximum\_width - minimum\_width \\ median &= minimum\_width + range/2 \\ variation &= (range/median) \times 100\% \end{aligned} \tag{40}$$

After the execution of `FORM-MAX-CLUSTERS(T)` we have partitioned the transistors into a set of maximal channel-connected subsets with an optional bound on the size variation within each set. These subsets can still be quite large, so we proceed to iteratively apply a standard Fiduccia-Mattheyses (FM) bipartitioning algorithm [25] to further partition these sets until a user-supplied upper size limit is met. Our goal in this step is to form transistor clusters which minimize the amount of external source-drain routing, so we only count source-drain nets when we calculate the net cut-set of a candidate partition (i.e. nets which connect to transistor gates are not considered.) Since we are primarily interested in minimizing the cut-set, and not in strict bipartitioning, we have found that we generally obtain much better partitionings when the FM balance criterion parameter is set to a relatively loose value.

As a final consideration, we note that it is possible for the two clusters produced by the FM bipartitioning step to lose their channel-connected property. This is especially common in large pass transistor networks. It is necessary to re-execute the `FORM-MAX-CLUSTERS()` algorithm on each cluster after each iteration of the FM algorithm.

In Figure 44 we show the automatic clustering solution for the dec-csa-mux benchmark [53], a carry-save adder with muxed latching inputs. The cluster upper-size limit was set to 10, the coefficient of variation term was set to 50%, and the FM algorithm balance criterion was set to 20%. Figure 45 shows one possible chaining solution for this set of transistor chains. Note the effect of the size variation constraint

4.4 Transistor Level Micro-Placement

At the conclusion of the clustering phase we have determined the set of atomic placeable objects and we can begin the placement phase. The atomic objects consist of individual unchained transistors and transistor sub-chains. The transistors have fixed geometry, and the task is merely to assign them an orientation and a location in the placement. The transistor sub-chains do not have fixed geometry. By manipulating the transistor ordering and orientation within the chains the placement step may alter the sizes and port locations of the individual sub-chains. Thus, in its search for an optimal solution, the placement step is free not only to explore the orientations of placements of the transistors and transistor chains, but it may also make use of the flexibility of the sub-chains in order to located an area efficient packing with minimal routing.

In the implementation of the placement phase it is possible to borrow a great deal from the field of macro-block placement. However at the transistor level, a problem which we refer to as *micro-placement* [92], several differences manifest themselves. We list them here and elaborate on each one in the following sections.

1. Modeling of the placement search space
2. Support for dynamic transistor chaining
3. Support for arbitrary geometry sharing

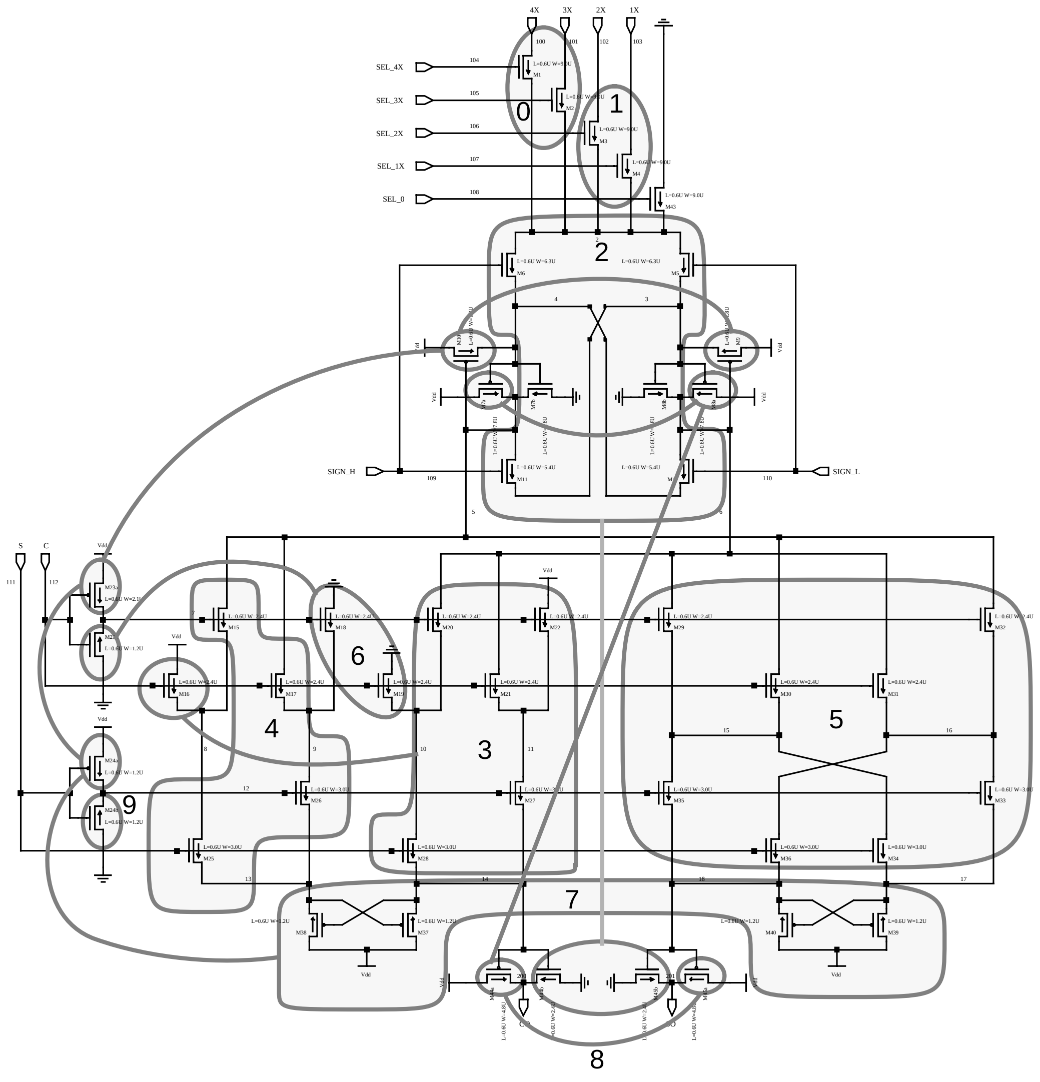


Figure 44: Automatic clustering solution for the dec-csa-mux benchmark [53]

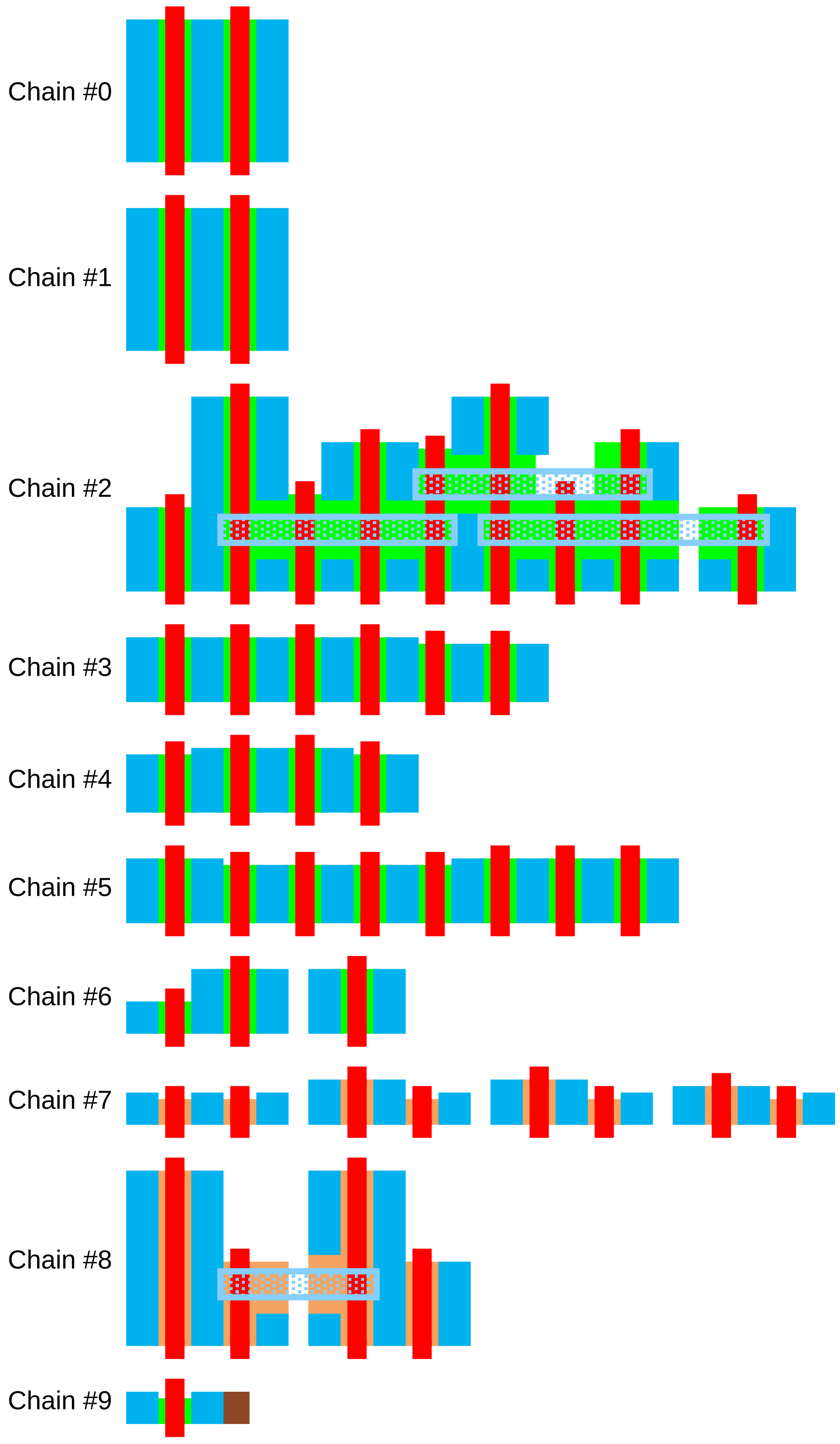


Figure 45: A Transistor chaining solution for the dec-csa-mux [53] benchmark circuit given the clustering solution from Figure 44

4. Presence of cell template constraints
5. Nature of the cost function
6. Nature of the routing model

4.4.1 Modeling of the placement search space

The subject of placement search space modeling is a complex one and was the subject of Chapters 2 and 3. Here we only summarize the conclusions of our previous study. By far the most popular placement models for *macro*-placement (standard-cell placement, block level placement, etc.) are *direct* models, as used in simulated annealing placement tools such as *Timberwolf-3.2* [101,102] and *KOAN* [17,18], as well as analytic placement tools such as *Gordian* [52]. Symbolic slicing-tree models are popular in block-level floorplanning tools [47,120,123].

For the purposes of transistor-level micro-placement, which requires the flexibility of a direct model but the search space efficiency of an indirect model, we developed In Chapter 2 an indirect model based on two-dimensional compaction constraints. Chapter 3 reviewed a symbolic model for these constraints called the sequence pair, which is due to Murata et al [77]. We then developed three optimization formulations for this model based on, respectively, integer linear programming, branch-and-bound optimization, and simulated annealing.

Experiments in Section 3.4 demonstrated that the optimal branch-and-bound algorithm is only practical for problems of small size. In addition, it is not yet clear how to correctly and efficiently model routing cost within this formulation. The integer linear programming formulation, while it has not yet been implemented, is likely to suffer from the same problems. For this reason we have adopted the simulated annealing formulation of the optimization problem. While heuristic in nature, simulated annealing optimization has been shown to achieve excellent results. In addition, the algorithm is extremely flexible and places few limits on the nature of the solution space and cost function models.

4.4.2 Support for Dynamic Transistor Chaining

As we mentioned in Section 4.3, it is very difficult to determine an optimal static clustering of a circuit's transistors into diffusion connected chains. This is because the clustering step has no information about the relative positions of the chains in the final placement. Traditional chain optimization algorithms can locate a chaining solution with minimum width and minimum internal routing cost, but such chainings may not be globally optimal when external area and wiring costs are taken into account. It is thus not clear *a-priori* to which chain a particular transistor should be assigned, and which chain ordering should be chosen.

Instead of investing a great deal of effort making static clustering choices prior to placement, we have adopted a methodology which allows the placement step to dynamically alter the clustering choices and chain orderings in order to find a placement that minimizes *global* area and wiring costs. As described in Section 4.3, the clustering step is used to form **chains**, of single-polarity diffusion connected transistors. The placement step is responsible for exploring different **chaining** solutions for these chains and selecting the globally optimal chainings with respect to the placement cost function.

During placement, the chains are broken at the diffusion breaks and the resulting **sub-chains** are passed to the placement engine as the atomic placeable objects. The diffusion breaks provide a natural point at which to perform this break—the diffusion strip is broken at these points and there is no reason to over-constrain the placement by forcing the resulting sub-chains to remain aligned. The sub-chains are then free to assume any two-dimensional arrangement that optimizes the placement.

In addition, as different chainings are explored for the chains, the sizes of the sub-chains are optimized to obtain the best placement area, and the transistors within the sub-chains are arranged so as to minimize the top-level detailed routing of the circuit. It is easy to see that there may be many sets of minimal covering Euler walks for a given chain, and that all of them will have the same number of diffusion breaks. Thus, the number of sub-chains in the placement will remain constant. When a new Euler walk is found for a chain, individual transistors may move from one sub-chain to another, and the individual sub-chains may grow or shrink in size depending on the number of edges traversed in each sub-trail.

We re-emphasize that we are not attempting to optimize the “height” (i.e. number of routing tracks) of these transistor chains, as is traditional in the literature. The height of the chain is fixed by the transistor widths in the input schematic. Instead, the transistor chain geometry generator reports a list of the internal chain nets which could not be routed over top of the chains, and these are added to the top-level routing cost (seeSection 4.4.6 for more detail on the placement routing model.) Thus, the sub-chains are automatically optimized such that their internal routing cost is taken into account as well as the resulting top-level routing cost.

In order to support dynamic transistor chaining, the list of simulating annealing moves discussed in Section 3.3.3 is augmented with two additional moves. The first move, which we call the **chain modification move**, was described by Basaran[1] [5]. One of the chains is selected at random, and a random sub-trail within the chain's current Euler walk is removed and replaced with a different randomly generated sub-trail. The new sub-trail must contain the same set of edges as the previous sub-trail, and it must have the same two vertices as endpoints.

The chain modification operation is easily performed by creating a new diffusion graph which contains only the set of vertices and edges (including the super-vertex and any artificial edges) in the selected sub-trail. We do *not* draw artificial edges from the two endpoints to the super-vertex. If the two endpoints are the same vertex, they will both have an even number of edges because the selected sub-trail is a *closed* Euler walk in of the selected diffusion sub-graph. The algorithm `EULER()`, defined in Section 4.2.3, is called to find a new closed Euler walk which begins and ends at that vertex. However, if the two endpoints are different, they will both have an odd number of edges as the sub-trail is an *open* Euler walk in the selected diffusion sub-graph. We call the algorithm `EULER_2()`, shown below, to find a new open Euler walk which begins and ends at the same pair of endpoints. Note that this algorithm is identical to algorithm `EULER()` except that v_α is replaced with v_β in line 4 and at the end of the trails in lines 5–7.

```
EULER_2($v_\alpha$,$v_\beta$)
1     if $v_\alpha$ has no edges
2         return $v_\alpha$
3     starting from $v_\alpha$ create a walk of $G$, never
4     visiting the same edge twice, until $v_\beta$ is reached
5     again; let [$v_\alpha$,$v_1$],...,[$v_n$,$v_\beta$] be this walk;
6     delete [$v_\alpha$,$v_1$],...,[$v_n$,$v_\beta$] from $G$;
7     return (EULER($v_\alpha$),EULER($v_1$),...,EULER($v_n$),$v_\beta$)
```

An example of the chain modification operation is shown in Figure 46. Here we use the diffusion graph for chain #2 of the dec-csa-mux benchmark [53] from Figure 44. We operate on the sub-trail $(2, 0, 6, 4, 2, S, 200)$, spanning the randomly selected indices 4–11 in the chain, and replace it with the new sub-trail $(2, S, 200, 0, 6, 4, 2)$. In Figure 47, beginning at the top, we show the geometry corresponding to these two transistor chains, along with a sequence of three additional chain modification moves. This example demonstrates how this move is capable of rearrang-

1. Basaran refers to this move as the "sub-trail modification move".

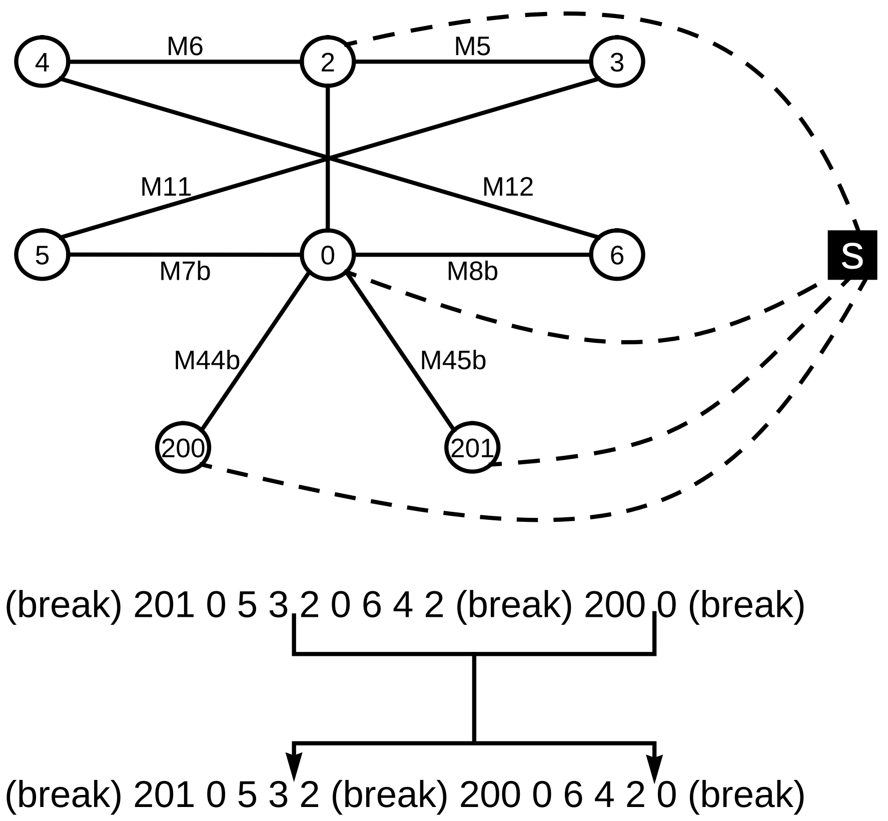
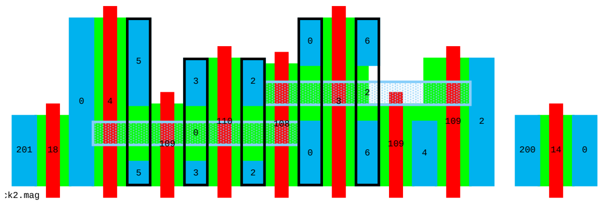

M6
M5
4
2
3
M11
M12
5
0
6
S
M7b
M8b
M44b
M45b
200
201
(break) 201 0 5 3 2 0 6 4 2 (break) 200 0 (break)
(break) 201 0 5 3 2 (break) 200 0 6 4 2 0 (break)

Figure 46: The chain modification move

ing not only the ordering of the transistors within the sub-chains, but also the relative sizes of the sub-chains in the placement.

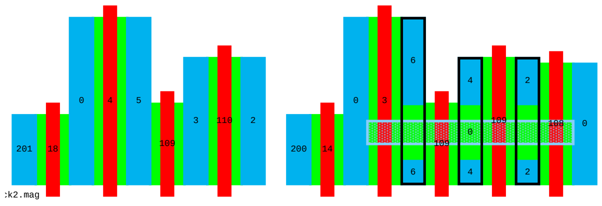
It can be shown [5] that the chain modification move is complete, and can thus be used to construct every possible Euler walk which can be embedded in the selected graph. However, a second move was added to increase the efficacy of the algorithm at low temperatures. Late in the optimization process, large chain modification moves can be very disruptive and are thus rarely accepted. The second move, the **sub-chain modification move**[1], simply selects one sub-chain and locates a new ordering for its constituent transistors. The sizes of the sub-chains in the layout will be unaffected, but the move may reduce the top-level routing cost.

1. The reader may note that we do not make use of Basaran's "trail-rotation" move [5 pp. 42-43.] This move simply has the effect or changing the ordering of the sub-chains within the chain. Since we are placing the sub-chains individually in the layout, this move is redundant in our case.



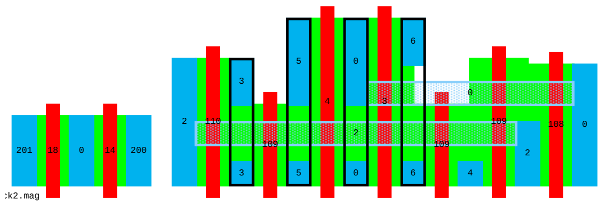
(break) 201 0 5 3 2 0 6 4 2 (break) 200 0 (break)

reconfiguring chain index1=4, index2=11



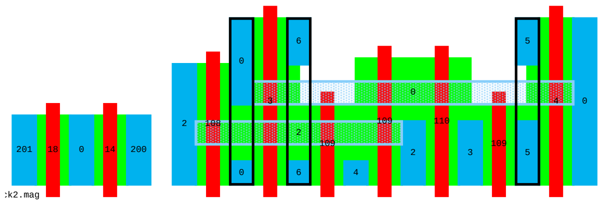
(break) 201 0 5 3 2 (break) 200 0 6 4 2 0 (break)

reconfiguring chain index1=2, index2=9



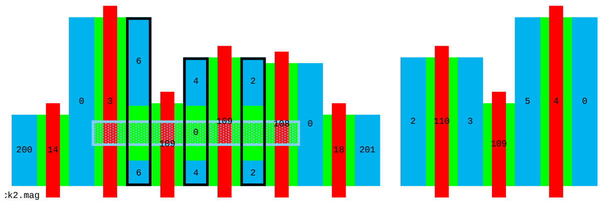
(break) 201 0 200 (break) 2 3 5 0 6 4 2 0 (break)

reconfiguring chain index1=5, index2=11



(break) 201 0 200 (break) 2 0 6 4 2 3 5 0 (break)

reconfiguring chain index1=8, index2=0



(break) 200 0 6 4 2 0 201 (break) 2 3 5 0 (break)

Figure 47: A sequence of 4 chain reconfigurations for chain #2 of the dec-csa-mux benchmark [53] (see Figure 44)

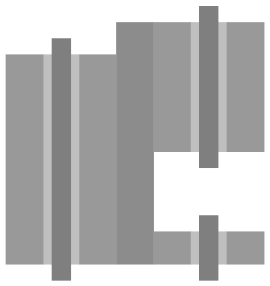


Figure 48: An example of a second-order shared structure

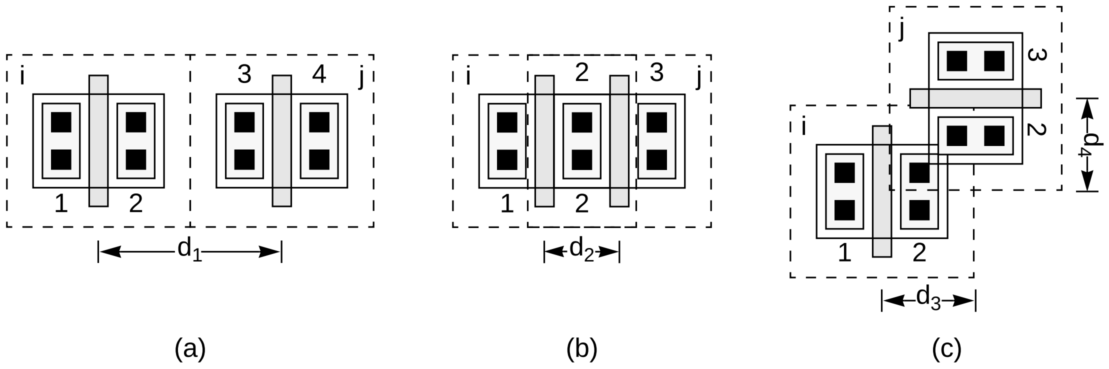


Figure 49: Three examples demonstrating a pair of transistors in different configurations with respect to geometry sharing.

4.4.3 Support for Arbitrary Geometry Sharing

The most noticeable difference between block placement and transistor-level placement is the fact that the objects being placed are no longer forbidden from overlapping. If two neighboring transistors of the same diffusion type share a common electrical node, it will be desirable to allow the objects to merge.

Our methodology supports geometry sharing at two different levels. The formation of transistor chains takes advantage of a great deal of the potential geometry sharing which is present in the circuit. We also allow arbitrary pieces of geometry to merge during the placement step, permitting larger merged structures to form and taking advantage of less obvious patterns of connection. We call the resulting merged objects **second-order shared structures**. An example in Figure 48 shows two small transistors merging with a single larger transistor to form a complex two-dimensional structure.

In order to support geometry merging at the transistor level we use a fairly simple mechanism, as shown in Figure 49. If two objects are in the proper configuration such that they have electrically compatible ports facing each other, as in Figure 49(b), the design rule constraint δ_1 can be relaxed to a smaller value, δ_2, to allow those ports to overlap. This may result in design rule violations in the final placement if the ports do not line up precisely, but this can be easily repaired in a post-processing step. This latter form of geometry sharing is formulated to encourage the formation of second-order source/drain shared transistor chains, and complex chains such as Figure 48. It does not currently support more complex shared structures such as Figure 49(c). This structure would require a *y-axis* constraint as well as an *x-axis* constraint to ensure design rule correctness.

4.4.4 Cell Template

In our cell synthesis methodology outlined in Section 4.1, Figure 33, along with the design rules and input schematic, a specification for the requirements of the cell template is required. The cell template, which was discussed briefly in Section 1.2 of Chapter 1, provides the specification for the physical requirements of the cell and the cell's interface to the external block placement and routing environment. In an industrial setting, the constraints which may be implied by the cell template can become quite complex and numerous [63] as we discussed previously. Our prototype implementation currently supports a fairly simple datapath style cell template which we will outline here.

We assume that the cells are intended for use in a datapath environment in which they will be arrayed in regular multi-bit patterns. We allow simple unconstrained cell area or perimeter optimization for the case in which the cells will be arrayed in a 2-dimensional grid and no inter-cell pitch matching is required. For situations in which several cells must be pitch-matched, we also support width or height minimization in which, respectively, the height or width is given a fixed upper bound.

At a minimum, the cell template must specify the method by which the cell primary inputs and outputs, as well as power and ground, are supplied. We assume that the multi-bit datapath is arranged so that bit-slices in the array are oriented horizontally in rows, and each column contains a single multi-bit component composed of identical cells. Bits are numbered starting with zero in the top row. Common control signals are routed vertically over the cells in second-level metal, and individual bits of the data input vector are routed horizontally over the cells in third-level metal. Power and ground are also supplied as horizontal third-level metal straps. Thus the primary inputs

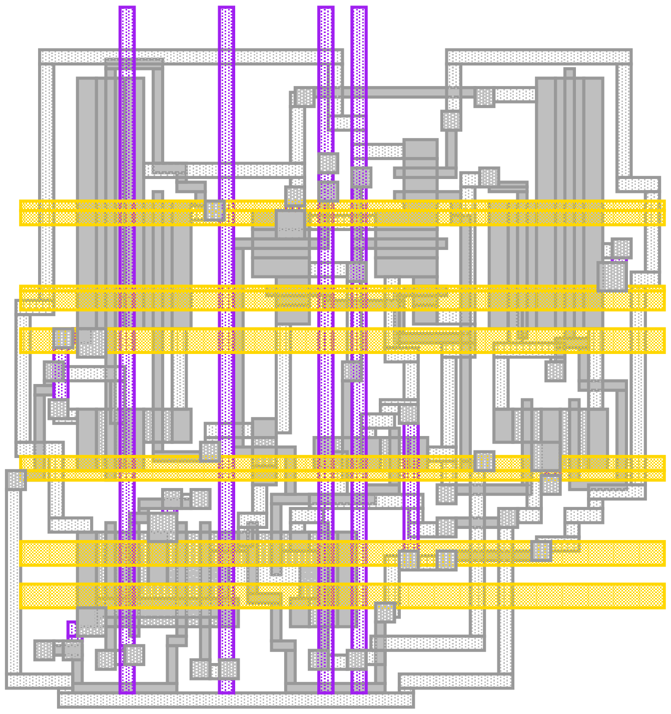


Figure 50: An example demonstrating the fixed overcell straps used to enforce a datapath-style port structure

and outputs, as well as power and ground, will come in to the cell as either second or third-level metal ports. Horizontal inputs are gridded in the vertical direction, while vertical routes are gridded in the horizontal direction. The grid location, or track, at which a particular port is located can either be specified by the user, or it can be left floating for assignment by the router in order to minimize internal cell routing.

In order to enforce the proper port gridding during the routing and compaction steps, at the conclusion of the placement step and prior to the routing step we assign each floating input/output port to an unoccupied overcell track in the proper layer. Tracks are assigned in a random order, and the assignment is made by attempting to place the track as close as possible to the center of the bounding box of its associated internal net. An example of a cell demonstrating the overcell input/output straps is shown in Figure 50. During the routing step, internal polysilicon or first-level metal internal routes are connected to these fixed ovecell straps with a second or third-level metal contact. After the compaction step the overcell straps are removed to increase the cell's porosity and allow the detailed router to connect to the ports contacts from any direction it chooses.

4.4.5 The Placement Cost Function

In Section 2.4.6 we briefly discussed the nature of the block placement cost function, and in Section 3.3.3.2 we discussed a general cost function for use with our simulated annealing implementation. Here we will develop this topic more fully in the context of our specific application.

In Section 3.3.3.2 we presented a cost function which was a weighted sum of a placement and a routing term. Here we make use of a similar cost function, but we have split the routing cost term into two separate terms as shown below.

$$\text{cost} = w_1 \cdot \text{placement} + w_2 \cdot \text{global routing} + w_3 \cdot \text{local routing} \tag{41}$$

The placement term supports all of the options discussed in Section 3.3.3.2: area minimization, perimeter minimization, width (height) minimization with bounded height (width), and area or perimeter minimization with a fixed aspect ratio.

For routing cost minimization we make use of either the RISA heuristic [14] for the Rectilinear Steiner Minimum Tree (RSMT) multi-terminal net model, or the Minimum Spanning Tree (MST) model as discussed in Section 2.4.6. However, in Equation 41 we show the routing cost split into two separate terms. We wish to allow different weights to be assigned to the estimates for top-level and local routing. This split is made because our model for routing, discussed separately in Section 4.4.6, uses different techniques to estimate the top-level inter-cell routing and the local incomplete intra-chain routing which has been delegated to the detail router for completion.

At this time we do not include terms in the cost function for the optimization of such things as electrical performance, yield or reliability, etc. However, the general simulated annealing formulation which has been developed here does not preclude the consideration of such topics. Any such goals can be treated as long as they can be calculated from the independent problem variables in a reasonable amount of time. Such secondary criteria are a major topic of concern in the field of analog layout synthesis and have been treated extensively. For example, see [4,12,13,58,82,83]. Applying similar techniques to high-performance digital circuits is an interesting topic for future work. For example, constraints ensuring the integrity of the dynamic node in domino-CMOS circuits, or constraints ensuring symmetric layout for the two halves of a CVSL pulldown network, could be envisioned.

4.4.6 Routing Model

Proper modeling of the routing contribution to the cost function of Section 4.4.5 is essential if high quality layout solutions are to be obtained. Because the routing step is performed serially after placement, it is critical that the placement step be able to accurately evaluate candidate placements with respect to the difficulty of their final routing.

Ideally, we would prefer to model the routing using the same metric as that of the placement: the cell area. If electrical performance is neglected, the cost of routing a candidate placement can be modeled solely by its contribution to the area. We simply need to compute how much extra area must added to the placement in order to permit a feasible routing. Or, alternately, given a placement with a given area, we need to determine if the routing problem is feasible or not. High quality placements will not only consist of a small arrangement of transistors, but they will also be efficiently routeable with a minimum of extra area.

The routing metrics discussed in Section 2.4.6 all model the routing cost in a more indirect manner through the use of total routing length. It is assumed that placements with minimum routing *length* with translate into placements which require a minimum of extra routing *area*. Ordinarily these two measures remain correlated, but it is natural that pathological cases exist for which this will not be the case. The main problem with the routing length estimates is that they neglect to consider routing *congestion*. If many short routes all end up being routed through the same area of the layout, the routing problem will become much more difficult and may prove infeasible.

The proper modeling of routing congestion has become an important topic in standard cell placement. A popular approach is to break the placement region up using a large-scale grid, and calculate the congestion as the ratio of routing demand versus its supply in each grid location. The routing **demand** is the number of routing tracks which are required to traverse the area, while the **supply** is the number of tracks which can be accommodated. The supply may be reduced by the presence of **blockages** within the grid location. A recent example of this philosophy is given by the RISA system due to Cheng [14].

The modeling of routing congestion may be equally important at the transistor level. However, routing at the transistor level bears little resemblance to standard cell routing. In standard cell blocks, most current designs are done in processes with four or more metal layers and routing is performed mostly over-cell. Blockages are small, usually representing cell port locations or prerouted nets. At the transistor level most local routing is required to be in polysilicon or first-level metal. Some metal routing may pass over the placed transistors and transistor chains, but transistor porosity is poor and most routing is performed in the channels between objects. If partially or fully strapped source/drain contacts are required the transistor objects will be completely non-porous. In the case of transistor routing, congestion must be defined as the demand versus supply present only within the routing channels.

Estimating channel routing congestion is a more difficult problem than estimating routing length. Many different minimum length paths for a specific net segment may exist, and these different paths are equivalent from the perspective of routing length. But to properly estimate channel congestion, a complete global routing must be computed. We have to know which specific channels are traversed by each net so that channel demand can be computed. Once channel demand is known, it is possible to estimate the width of each channel. If the width of a channel exceeds the width as determined by the design rules separating the blocks on either side, the channel will have to be **bloated** and the placement area will likely increase.

The detailed router which we have adopted (see Section 4.5) does not have the ability to modify the placement. It simply attempts to find a routing given a set of port locations, treating all of the placed circuit elements as fixed blockages. Therefore the placement stage is required to supply the router with sufficient empty space to guarantee a feasible routing. During placement some type of estimate must be made for the amount of extra area which will be required by the routing geometry. If this estimate is accurate enough, it may also be used in the placement cost function as a metric for the routing complexity.

Our facility for routing area insertion is described in Section 4.4.6.2. We have found that, for the purpose of guaranteeing a feasible routing, only crude estimates of the routing area, and the distribution of the routing area within the placement, are required. However, these estimates are of insufficient accuracy to provide the simulated annealing placement stage with guidance toward finding easily routed solutions. The *sensitivity* of the routing area estimates to the detailed placement is small and therefore provide insufficient feedback to the annealer. For this reason we still include a traditional routing length metric within the placement engine as well. We will describe our model for routing length estimation first in Section 4.4.6.1 and then cover routing area insertion in Section 4.4.6.2.

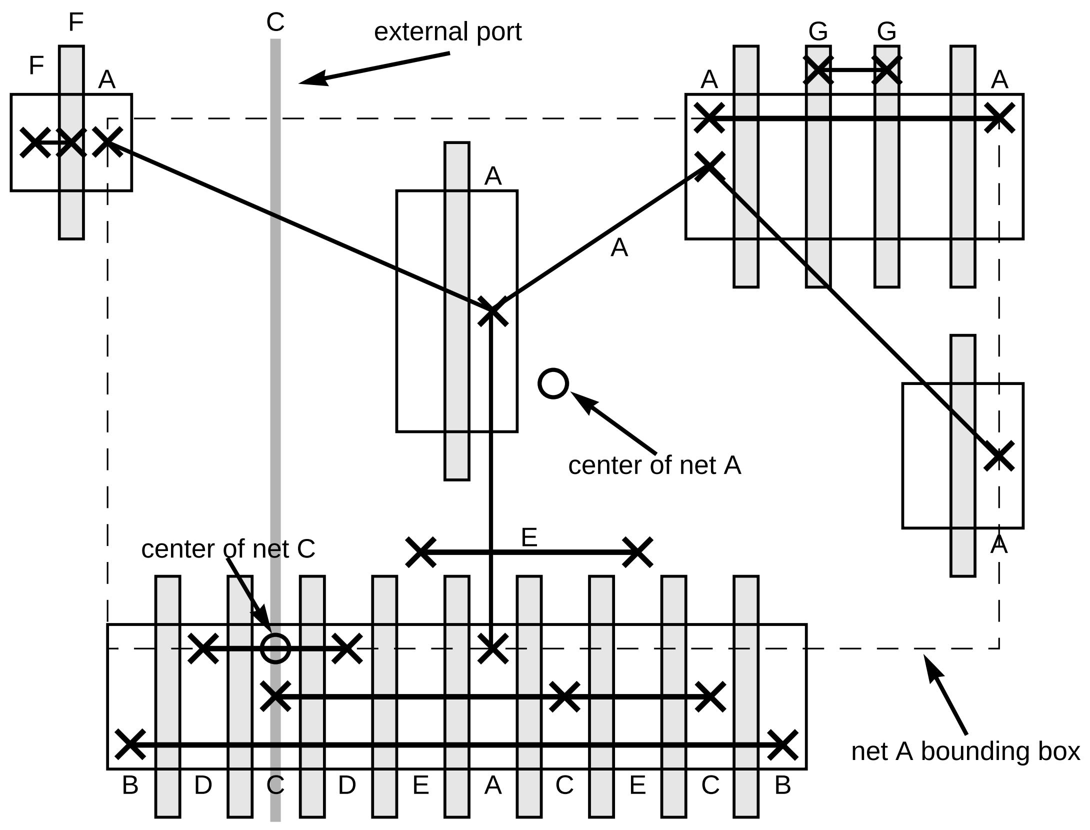


Figure 51: An example demonstrating the components of the routing cost function

4.4.6.1 Routing Length Estimation

Our model for transistor level routing length estimation is demonstrated in Figure 51. Here we show examples of each of the components of the routing length cost. We currently assume that all local routing is done in polysilicon and first-level metal. Upper level metal layers are reserved for block and chip-level routing, and any use of these layers within the cell would adversely effect the cell’s porosity. For connections to the cell’s primary input and output ports we assume the cell template described in Section 4.4.4. Control inputs arrive vertically in second-level metal and data inputs, outputs, power, and ground arrive horizontally in third-level metal.

If full diffusion strapping is required, all routing will have to be performed in the channels separating the placement objects. Otherwise, a large number of local source/drain routes within the transistor chains should be completed over top of the chains by the intra-chain router in the chain generator (see Section 4.2.7.) Examples of intra-chain source/drain routes are provided by nets B,C, and D, and a portion of net A, in the figure. These nets are routed with zero cost to the placement and do not enter into the placement cost function.

However, if the intra-chain router cannot complete an internal source/drain route, the route will be pushed out into the routing channels and will be delegated to the inter-chain detail router. An example is provided by net E in the figure. Intra-chain routes may also involve gate routing: either source/drain-to-gate (net F), or gate-to-gate (net G). Since polysilicon cannot be routed over the chain, it also must be handled by the detail router. In cases such as these the chain generator supplies the total length of all incomplete intra-chain routes, and these are summed over all chains to yield the *local routing* term in Equation 41.

The primary contribution to inter-chain routing is made by multi-terminal nets which span between different transistors and transistor chains in the placement. Net A is an example of such a net. Inter-chain nets are treated as Minimum Spanning Trees and their length is estimated using the methods described in Section 2.4.6. An interesting problem is introduced by the presence of nets for which multiple ports are available on the same transistor chain. This can be seen on net A in the chain at upper right in the figure. In such a case it is not always clear which port should be used when making the estimate. We use a simple heuristic in this case. We first find the minimal enclosing bounding box for the net, as shown in the figure. We then select the port on each chain which lies closest to the center of this box.

A final concern is the proper modeling of the cell template and the routing of primary inputs and outputs. Net C in the figure is an example of a cell-level primary output. Our datapath-style cell template, described in Section 4.4.4, specifies gridded horizontal and vertical ports that, during routing, will be full width/height straps in second and third-level metal. The track assignment for each port can either be fixed by the user, or left floating for assignment by the router in order to optimize internal routing.

The router discussed in Section 4.5 attempts to place each floating overcell primary input/output port at the position nearest to the center of the bounding box of the internal net to which is must connect, as shown in the figure. During routing length estimation we assume that the router will be successful at this. Even if there are conflicts for specific primary input/output overcell tracks, we assume that the router will be able to place the port somewhere within the bounding box of the minimum spanning tree for the internal portion of the net. Therefore the routing of floating primary input/output connections will not incur any inter-chain routing cost in the routing length estimation. If a fixed input/output port is not overlapped by the bounding box of its associated internal net, the shortest distance from the edge of the net bounding box to the port is added to the routing cost for that net.

4.4.6.2 Routing Area Insertion

As outlined earlier, the insertion of extra area—area above and beyond that already present between objects due to the design rules—must be performed prior to circuit routing in order to ensure that the routing problem will be feasible. This area expansion step may be performed once at the conclusion of the placement stage, or it may be performed on every candidate placement in the inner loop of the placement step. If the latter option is exercised, the extra routing space will contribute to the cell area component of the placement cost function and may be used as an estimate of routing cost. We have explored three techniques for routing area insertion which are diagrammed in Figure 52.

Figure 52(a) shows an intermediate placement without the addition of routing space. It is clear that some nets will be un-routeable. In Figure 52(b) we use a technique from the *Koan* analog placement tool [18]. Here, each object’s bounding box is increased to include an extra routing “*halo*”. This halo is proportional in size to the number of I/O ports and inversely proportional to the perimeter of the object:

$$\text{halo}_i = \alpha \cdot \frac{\#\ \text{I/O ports}_i}{\text{perimeter}_i} \tag{42}$$

where α is an experimentally determined scaling factor for tuning purposes.

We have found that this halo-based method is difficult to tune and tends to leave more space than necessary between most blocks, requiring post-routing compaction. In addition, the halos tend to make the objects more uniform in size, due to its inverse relationship to the perimeter, resulting in placements which are more regular in appearance than manual designs. An additional problem is that this non-uniform object growth tends to break apart second-order shared structures that are formed through design rule relaxation as described in Section 4.4.3. An example of this is marked with an arrow in the figures.

In Figure 52(c) we use a technique for routing space insertion due to Murata [77], which is an elaboration of a technique described by Onodera [84]. Here each object is translated from its original coordinates (x_i, y_i) to a new set of coordinates (x_i', y_i') based on an estimate of the routing resources which will be required below it and to the left. The new position is calculated as fol-

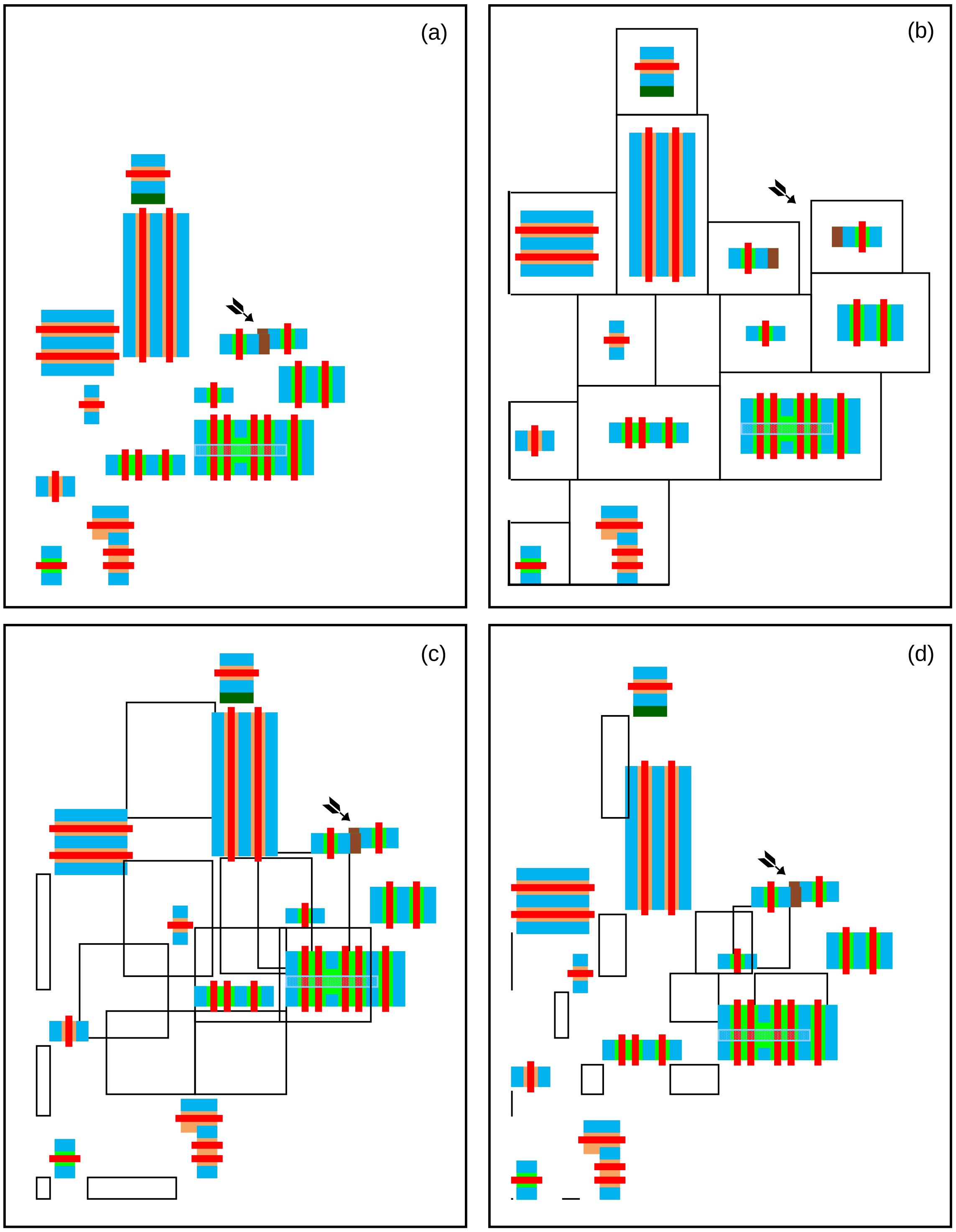


Figure 52: A demonstration of different routing space estimation techniques. Example (a) has no extra space reserved for routing, example (b) uses the method from [77], and example (c) uses the method from [18]. Example (d) uses our modified version of the method in [84].

lows:

$$x_i' = x_i + \beta_x T\left(\frac{\sum_{i \in N_{x_i}} H_i}{H}\right) \qquad y_i' = y_i + \beta_y T\left(\frac{\sum_{i \in N_{y_i}} W_i}{W}\right) \tag{43}$$

where T is the routing pitch, N_{x_i} and N_{y_i} are the set of nets whose bounding boxes begin before object i in x and y, H_i and W_i are the height and width of net i's bounding box, and H and W are the height and width of the original placement. Again, β is an experimentally determined scaling factor which can assume separate values in x and y. In the figure (x_i, y_i) correspond to the coordinate of the lower-left corner of each object. The additional boxes demonstrate the movement of each object—their lower left corners mark (x_i, y_i) and their upper-right corners mark (x_i', y_i'), for the objects i appearing at their upper-right corners.

Murata's method, designed for block placement, is making use of a greedy heuristic which assumes that every horizontal (vertical) net makes exactly one vertical (horizontal) jog at the far left edge of its bounding box. We have found that this method introduces a nonuniform bias in the routing, concentrating the routing space toward the lower left corner, leaving objects closer to the upper and right edges crowded together. This bias should be evident in the figure.

The final technique which we have explored is shown in Figure 52(d). This method, attempting to correct some of the problems which we found with Murata's method, makes different use of the method of Onodera [84]. We simply solve Equation 43 for $x_{\max}'$ and $y_{\max}'$, the upper right corner of the design, and assigns new coordinates to the blocks based on the following:

$$x_i' = x_i + \delta_x(x_{\max}' - x_{\max})\frac{x_i}{x_{max}} \qquad y_i' + \delta_y\left(y_i + (y_{\max}' - y_{\max})\frac{y_i}{y_{max}}\right) \tag{44}$$

where δ is the experimentally determined scaling factor. This method approximate the total number of horizontal and vertical routing tracks which will be required, assuming that each net occupies one horizontal and one vertical track, and distributes these uniformly throughout the design. As can be seen in the example, the allocation of routing space appears more uniform than in example (c), without the strong bias toward the lower-left corner.

As a final observation, note that the latter two expansion-based techniques based on Onodera's method do not break apart second-order shared structures (marked with an arrow) which were present in the original un-expanded placement. This is explicitly supported in the expansion algorithm by identifying instances of geometry sharing and assigning all shared structures the same expansion factor as the most lower-left shared object.

We have chosen to adopt the final technique described above, which we term *uniform expansion*, for the reasons already discussed. This is an extremely crude technique, performing no global routing to determine the exact demand cost in each channel, but instead making the assumption that the final routing will assume a statistically uniform distribution. If there are areas of congestion in the routing, the uniform expansion needs to be tuned using the scaling factor δ in order to accommodate the worst case spacing required in the congested region. The pessimism of this technique requires the use of a post-routing compaction step in order to clean up the un-utilized empty space which remains after routing.

The simplicity of this approach was motivated by its inclusion in the inner loop of the placement step so that routing space could be included in the estimate for cell area used by the placement cost function. The use of global routing techniques to more accurately perform the pre-routing expansion, or the development of detailed routers capable of channel expansion (as in the case of standard cell channel routers) will be an interesting topic of future research.

4.5 Routing

It is the task of the routing step to complete all of the electrical connections in the circuit which we not made either through geometry merging or by the intra-chain router within the transistor chain generator. The output of the placement step consists of the mask geometry for the placed transistors and transistor chains. Information is supplied to the router about the locations and mask layer of each electrical terminal that must be routed. In addition, as mentioned in Section 4.4.4, the placement step has made overcell track assignments for the cell's input, output, and power ports. Vertical ports were assigned unique horizontal track numbers and full-height second-level metal strap were placed in the placement to enforce the gridding of the ports. Similarly, horizontal ports were assigned a vertical track number and their grid positions were enforced through a full-width horizontal third-level metal track.

To complete the cell routing we make use of a third party router, *Anagram-II* [31], which is a detail router originally designed for use in an analog placement and routing environment [18]. *Anagram-II* uses a non-gridded line-expansion routing algorithm along with an aggressive rip-up

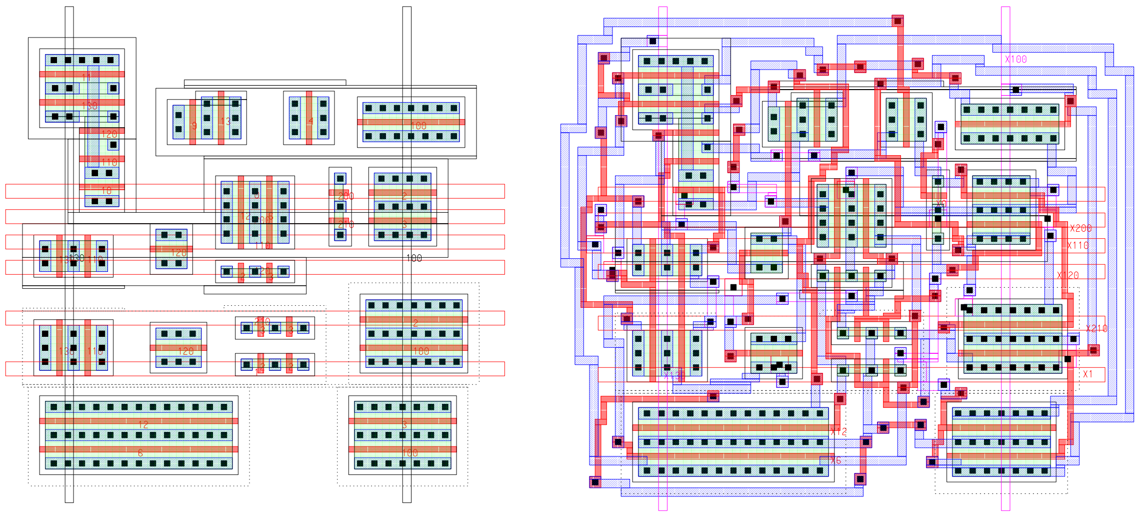


Figure 53: An example of a cell placement before and after routing with *Anagram-II*

and re-route capability in order to achieve extremely dense transistor-level detailed routes. *Anagram-II* also supports a number of analog-specific capabilities of which we do not currently take advantage, namely symmetric routing and crosstalk avoidance. In Figure 53 we show an example of a completed cell placement before and after routing with *Anagram-II*.

Anagram's line expansion routing algorithm is a form of maze routing [106] which at every time step preferentially expands an uncompleted route from the point on the route which currently lies on the optimum path to the sink terminal. For any pair of unrouted points, this strategy is guaranteed to find the optimum route if one exists. However, global optimality is dependent on the order in which net segment port pairs are selected for routing. In order to reduce this order dependency, a rip-up and re-route scheme, along with net aging, is implemented to allow the port routing order to be determined dynamically during routing.

4.6 Compaction

Ideally there should be no need for a compaction step in a polygon based non-symbolic placement and routing environment. If design rules are modeled correctly the mask data generated by the tool should be correct by construction and require no legalization. However, as we outlined in Section 4.4.6.2, our routing area insertion technique is based on a rather crude heuristic which often results in a layout which contains un-utilized empty space. The final step in our methodology from Figure 33, compaction, is an optional step which can be used to remove this unwanted empty

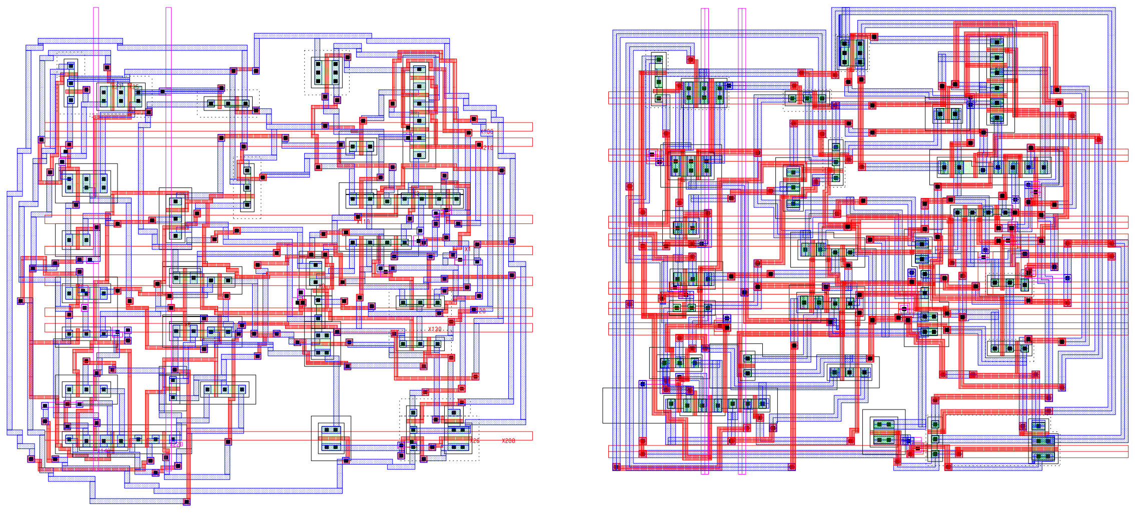


Figure 54: An example of a cell placement before and after compaction with *Masterport*

space and reduce the final cell size.

The amount of empty space which is ultimately removed will depend on how aggressively the routing area insertion scaling parameter, δ, was set by the user, and by how non-uniform the final routed layout turned out to be. As we mentioned previously, if the final detailed routing is heavily congested in some regions, and therefore very non-uniform, the scaling parameter will need to be set somewhat conservatively, and some regions of the layout will contain unwanted extra space.

In order to implement our compaction step we make use of a third-party commercial cell compactor, Masterport [22] from Duet Technologies (formerly Cascade Design Automation.) An example of a congested cell design before and after compaction is shown in Figure 54. As discussed in Section 4.4.4, after the compaction step is completed the overcell second and third-level metal input/output straps are removed. The remaining metal contacts remain as gridded input/output ports available to the higher-level cell placement and routing environment.

4.7 Summary

The subject of Chapter 4 was the detailed modeling of transistor-level placement and routing within the generic sequence pair placement model developed in Chapters 2 and 3. It was observed that the primary difference between arbitrary block placement and transistor placement is the need to capture geometry sharing between objects. The formation of transistor source-drain connection through object overlap serves both to reduce the placement area and to reduce the need for expensive wiring. In this context we developed two linked approaches to capture geometry sharing. The explicit formation of linear **chains** of merged transistors is used to form what we refer to as **first-order shared structures**. We also allow more random geometry merging to take place during placement through the use of an adjacency analysis step. Electrically compatible source-drain ports on adjacent transistors or transistor chains are allowed to overlap through the relaxation of their pairwise separation constraint. We refer to the resulting merges objects as **second-order shared structures**.

Our complete methodology was first outlined in Section 1.3 of Chapter 1, and this chapter served to elaborate in detail each phase. To review, we begin with a static clustering step which is used to group transistors into diffusion connected **chains**. This is followed by a dynamic placement phase which assigns an orientation and physical location to each object in the placement. The atomic placeable objects are individual transistors and transistor **sub-chains**. The latter are defined as transistor chain segments split at the location of the diffusion breaks. The placement phase is given the capability of dynamically adjusting the ordering of the transistors within the chains, which has the effect of altering the sizes of the atomic sub-chains and allows the placement algorithm more freedom to optimize the placement and the top-level routing cost.

In Section 4.2 we reviewed the basic problem of transistor chaining as applied to non-dual transistor chains. A graph construction technique and associated Euler-walk algorithm developed by Basaran [5] was described. We make use of this model to represent the set of all minimum width chainings for each cluster. The topic of transistor chain routing, or “height” minimization was addressed, though we do not seek to optimize this figure directly. Rather, we allow the cost of chain routing to be reflected in the cost of the top-level detail routing problem, and rely on the placement optimization algorithm to optimize chain routing directly. The specifications for our transistor chain geometry generator were given and our dynamic sub-chain optimization approach was justified.

Section 4.3 introduced our technique for static circuit clustering which is based on a general purpose FM graph bipartitioning algorithm. This basic approach was augmented with a cluster size constraint used to maintain a fixed maximum size differential between transistors in the same chain. Section 4.4 developed the details of our dynamic simulated annealing based transistor-level placement approach, a problem which we call **micro-placement**. Our method for dynamic transistor chain optimization, based on an Euler graph sub-chain modification move due to Basaran [5], was discussed. This was followed by the details of the adjacency analysis step used to allow the formation of second order shared structures.

In discussing our routing model we explained two critical topics, routing length estimation and routing area insertion. Routing length estimation was addressed with a standard Minimum Spanning Tree model. Three different techniques for routing channel area insertion were examined, the most promising of which is based on a uniform expansion by an amount proportional to the routing length cost. These two problems turn out to be critical to the success of the approach, and it will be shown in the experiments of Chapter 5 that our relatively simple solutions are not always adequate. We concluded Chapter 4 with a discussion of the third-party applications selected for the final top-level detailed routing and compaction phases of our methodology.

CHAPTER 5
Experimental Evaluation

5.1 Implementation

In this section we discuss a prototype tool called *TEMPO* (Transistor Enabled Micro Placement Optimization) which has been developed as a framework with which to explore the ideas discussed in this dissertation. *TEMPO* is implemented in approximately 48,000 lines of C++ code and has been tested under the Sun Solaris and Linux operating systems. A screen shot of the TEMPO main window is shown in Figure 55

To summarize the results of Chapters 3 and 4, we have implemented a generic simulated annealing placement engine based on the symbolic sequence pair model. This engine has been adapted to transistor level placement through the representation of placeable transistor and transistor chain geometry primitives. A static transistor clustering step is followed by a placement step with integrated dynamic transistor chain optimization. The formation of second-order shared structures is encouraged through an adjacency analysis which collapses design rule constraints to allow source-drain geometry overlaps. Routing length estimates are made using a minimum spanning tree model, and three alternative methods for performing routing space estimates have been implemented. A graphical user interface written in Tcl/Tk allows visual inspection of the optimization process and facilitates parameter tuning.

The input to *TEMPO* is a Spice netlist file and a technology file specifying the design rules. Generators have been implemented to construct the transistor chain geometry, making use of dynamic chain ordering assignments from the annealing engine, or optional static clustering and chaining order specifications supplied by the user through annotations in the Spice file. We make use of the *ANAGRAM-II* router from Carnegie Mellon University for post-placement routing, and *MASTERPORT* from Duet Technologies inc. for post-routing compaction.

In the current implementation we make use of a simple datapath-style cell template as

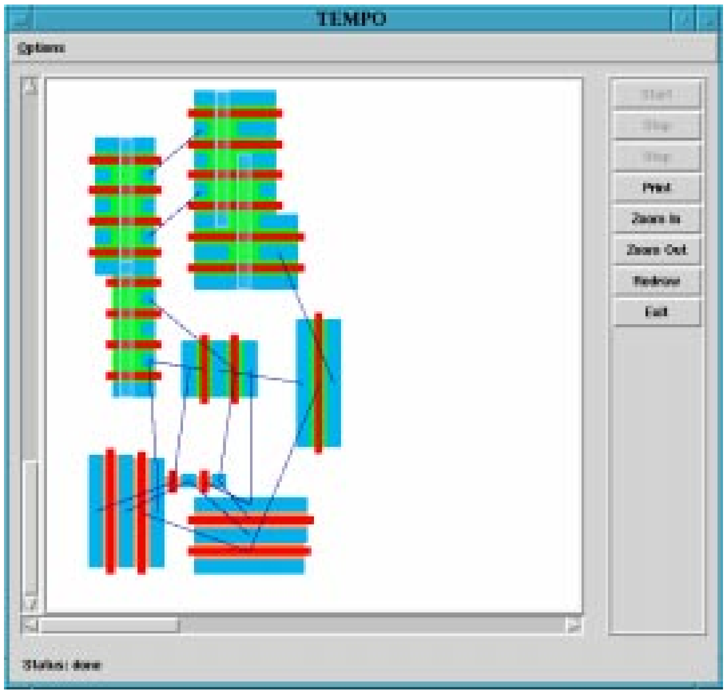


Figure 55: A screen shot of the *TEMPO* main window

described in Section 4.4.4. Internal routing is performed in polysilicon and first-level metal. Control inputs are assumed to arrive vertically in second-level metal and data inputs arrive horizontally in third-level metal. We place these inputs as over-cell routing tracks in a position which is as close as possible to the center of the associated net's bounding box. If desired, the positions of external feedthrough nets can be specified manually through annotations in the spice file.

5.2 Benchmark Circuits

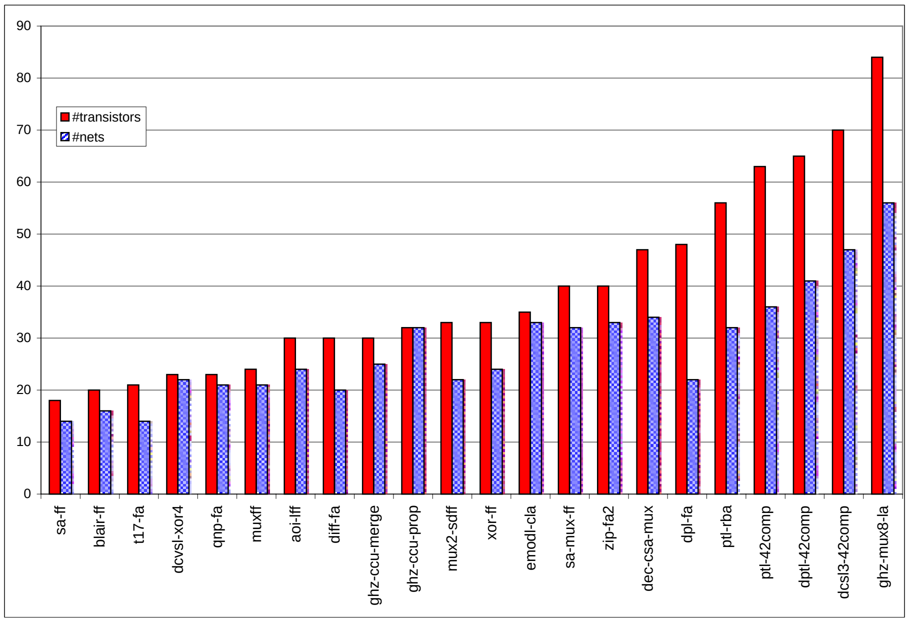
One-dimensional cell synthesis tools can be rigorously tested using standard complementary CMOS logic gates, and it is relatively easy to reproduce these circuits when making comparisons between different techniques. Maziasz and Hayes [73], in fact, show that there are exactly 3505 dual series-parallel circuits of "practical size" (i.e. with a maximum series chain length of 4.) However, no such set of easily generated benchmarks exist to demonstrate the full capabilities of the 2-D cell synthesis style. Such circuits may consist of non-dual, non-series-parallel topologies with non-uniform transistor sizing, and should represent real circuit designs of interest to the designers of modern high-performance VLSI chips. The lack of a standard set of benchmark circuits has made quantitative comparison among 2D cell synthesis tools difficult.

In order to address this situation we have assembled a suite of 22 benchmark circuits, taken mostly from the solid state circuits literature, which we present as a new tool for use in cell synthesis research. These should be useful not only for problems related directly to leaf-cell circuit layout synthesis, but also for problems in transistor sizing, circuit testing and test vector generation, and timing characterization and timing analysis.

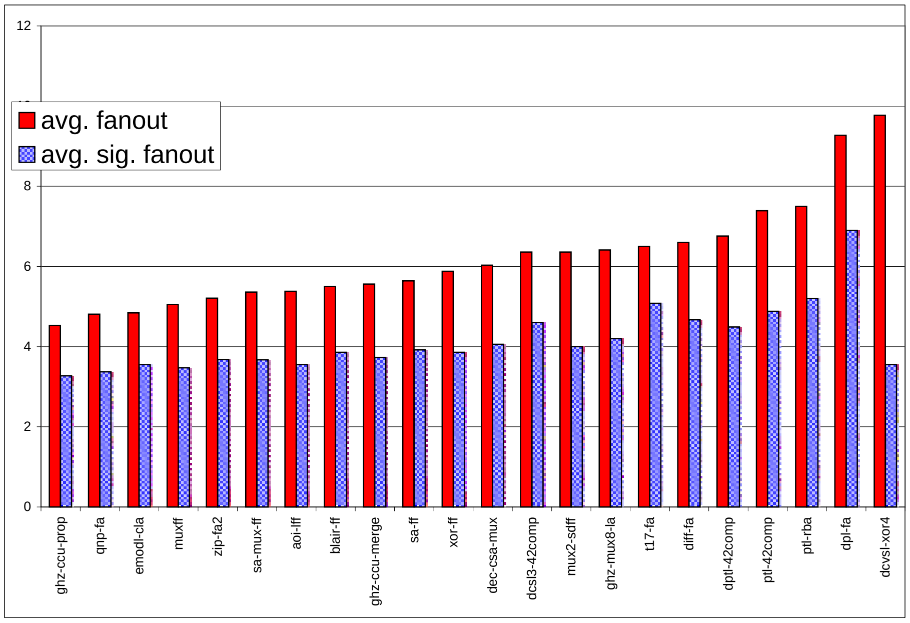
Our benchmark circuits represent a wide range of design styles which are of common use in high-performance and low-power applications: True Single-Phase Clocked (TSPC) logic flip-flops, sense-amp flip-flops, various static and dynamic Cascode Voltage Switch Logic (CVSL) families, single-ended and dual Pass Transistor logic (PTL), true and quasi domino logic, multiple-output domino logic, and zipper logic. Most of the circuits consists of arithmetic functions, or latching elements with embedded logic, that would be found in typical datapath applications. Less of an emphasis is given to circuits with control applications, as these are generally made up of simpler static or dynamic CMOS complex gates which are well handled by traditional 1-D techniques.

A description of each benchmark circuit is given in Table 6. Figure 56 displays some data which can be used to judge the relative complexity of the different benchmark circuits. In Figure 56(a) we show the circuit sizes, as measured both by the number of transistors and the number of nets. Figure 56(b) gives information about the average net fanout in each circuit which may give an indication of the relative routing complexity. Because the power and ground nets typically fan out to a large fraction of the transistors, we show net fanout for all nets and for signal nets only. The data on which these two figures are based can be found in Table 7 of Appendix B.

In most cases, the specifications for the benchmark circuits supplied by the original authors did not include transistor sizing information. Even had it been included, the sizes would have been optimized for a wide variety of different fabrication processes. In order to facilitate meaningful comparisons made with the use of these benchmarks it is important that a single fabrication process be adopted and a consistent transistor sizing methodology be used across all benchmarks. Transistor size tuning turned out to be a non-trivial problem as we were unable to gain access to suitable tools. Our own transistor tuning solution, named *Topt*, is based on a simple heuristic nonlinear gradient descent optimization algorithm and makes use of *Hspice* for time-domain transient simulation. Appendix A gives a brief survey of the field of transistor sizing and a detailed description of the *Topt* algorithm.



(a) benchmark size data



(b) benchmark fanout data

Figure 56: Benchmark size and complexity statistics

Table 6: Benchmark Circuit Descriptions

Name [source]	Description
aoi-lff [89]	True Single Phase Clocked Logic (TSPCL) and-or-invert logic flip-flop
blair-ff [7]	high-speed differential double edge triggered CMOS flip-flop
dcsl3-42comp [110]	Differential Current Switch Logic 4-2 compressor (DCSL-3 style load)
dcvsl-xor4 [42]	Dynamic Cascode Voltage Switch Logic 4-way XOR gate
dec-csa-mux [53]	high-speed carry-save adder with muxed latching inputs
diff-fa [107]	differential full adder
dpl-fa [116]	Double Pass transistor Logic full adder
dptl-42comp [40]	Dynamic Pass Transistor Logic 4-2 compressor sum logic
emodl-cla [124]	Enhanced Multiple Output Domino Logic carry-lookahead adder stage
ghz-ccu-merge [10]	4-way dynamic merge circuit for 1GHz PowerPC condition code unit
ghz-ccu-prop [10]	4-way dynamic propagate circuit for 1GHz PowerPC condition code unit
ghz-mux8-la [109]	8-input latching mux with hold and scan for 1GHz PowerPC
mux2-sdff [51]	Semi-Dynamic dual-rail logic flip-flop with embedded 2-input mux
muxff [113]	dynamic Complementary GaAs mux flip-flop
ptl-42comp [128]	Pass Transistor Logic 4-2 compressor
ptl-rba [67]	Pass Transistor Logic redundant binary adder
qnp-fa [66]	Quasi NP-domino pipelined full adder
sa-ff [76]	differential edge-triggered sense-amplifier flip-flop
sa-mux-ff [112]	differential edge-triggered sense-amp flip-flop with embedded 4:1 mux
t17-fa [105]	seventeen transistor low-power full-adder
xor-ff [70]	sense-amplifier flip-flop with dynamic differential embedded xor gate
zip-fa2 [27]	dual-bit adder in CMOS zipper logic

The benchmarks used to conduct the experiments in this dissertation were tuned for the MOSIS scalable submicron design rules (rev. 7.2) with a lambda value of 0.3µm. We are assuming the use of the 0.5µm HP-CMOS14TB process with a 3.3 volt supply. Spice level-13 process parameters were taken from post-fabrication test structure measurements of the n73d MOSIS run dated April 29, 1997. All gates were sized assuming a 40fF capacitive load on the primary outputs, which is roughly equivalent to three inverter or pass-gate inputs and 300 microns of metal routing. All primary inputs are driven by piecewise linear input waveforms with 500pS rise/fall times driving two stages of non-tunable inverters. These input buffers were given a fixed 30fF load in addition to the load provided by the gate, and were sized to provide balanced 500pS rise and fall times with a 40fF load.

5.3 Experimental Results

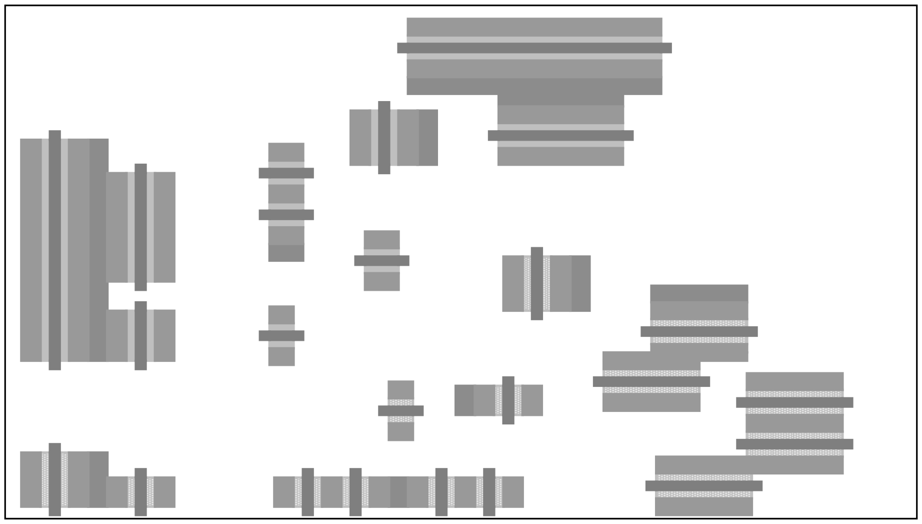
In this section we present a number of experiments which are intended to demonstrate the capabilities and limitations of the two-dimensional cell synthesis methodology implemented in *TEMPO*. We begin with a final detailed look at the mux-flipflop benchmark which we have been using as a running example throughout this dissertation. We then present the results of a study conducted on the complete set of twenty two benchmark circuits introduced above in Section 5.2.

The heart of *TEMPO* is its ability to manage transistor geometry sharing through integrated transistor chaining (which we refer to as first-order geometry sharing) and arbitrary geometry merging (which we refer to as second-order geometry sharing.) Figure 57(a) shows the output of *TEMPO* when clustering and dynamic transistor chaining have been disabled. Figure 57(b) shows the mux-f flipflop placement as realized by the *Koan* analog placement system [18]. Here all geometry sharing in *TEMPO* results from the merging of adjacent geometry. It closely resembles the output of *Koan*. Without integrated transistor chaining, *Koan* and *TEMPO* are not able to discover the long transistor chains which are clearly visible in the manual design.

Figure 58(c) shows the final routed and compacted layout of the mux-flipflop benchmark produced using the full capabilities of *TEMPO*. For reference Figure 58(a) shows a manually designed version of this cell and Figure 58(b) shows the layout produced by the 1-1/2 dimensional LAS cell synthesis package from Cadence Design Systems inc. All three cells are shown approximately to scale, however the manual cell was designed in a complementary GaAs process so a direct area comparison should not be made. Figure 59 shows the static transistor clusterings which were used in the manual design and by *TEMPO*.

The reader should observe that the mux-flipflop layout produced by *TEMPO* clearly demonstrates a much more hand-crafted appearance than that produced by *LAS*. The row-based layout style used in *LAS* is intended for use on cells designed in a static CMOS or ordinary dynamic CMOS logic style and is less suited to complex custom circuit topologies. In order to demonstrate

(a)



(b)

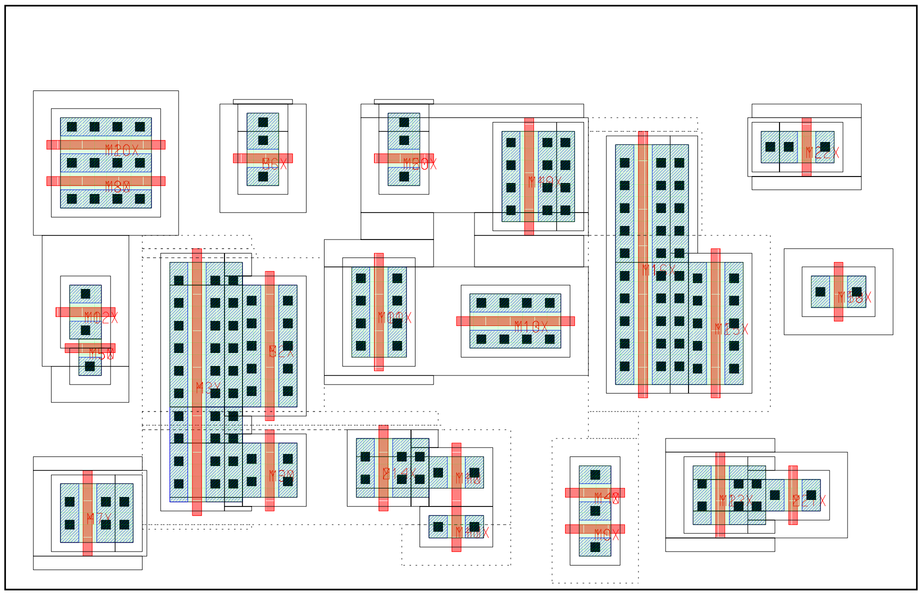
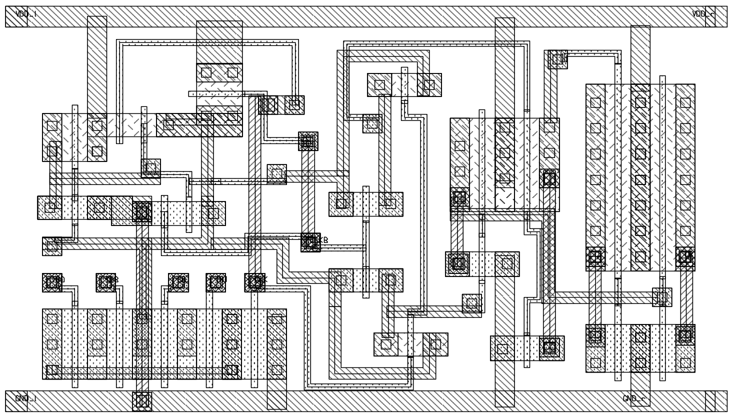
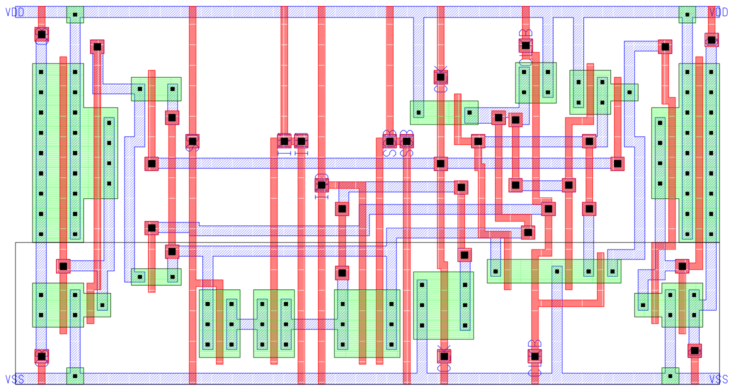


Figure 57: The mux-flipflop placement produced by: (a) TEMPO with transistor chaining disabled, (b) the *Koan* analog placement system [18].

this observation more convincingly we now present the results of a set of experiments conducted on the complete set of benchmark circuits discussed in Section 5.2.

Our experiments were performed on the set of 22 circuits summarized in Table 6 and Figure 56. As discussed in Section 5.2, the circuits were tuned for a 0.5μm CMOS process with the use of SPICE models extracted from fabricated wafer test structures. The MOSIS scalable sub-micron CMOS design rules were used to produce all layouts. The layouts produced by *TEMPO* are compared with those produced by LAS, a state-of-the-art commercial cell synthesis environment


(a) manual layout


(b) Cadence *LAS* layout

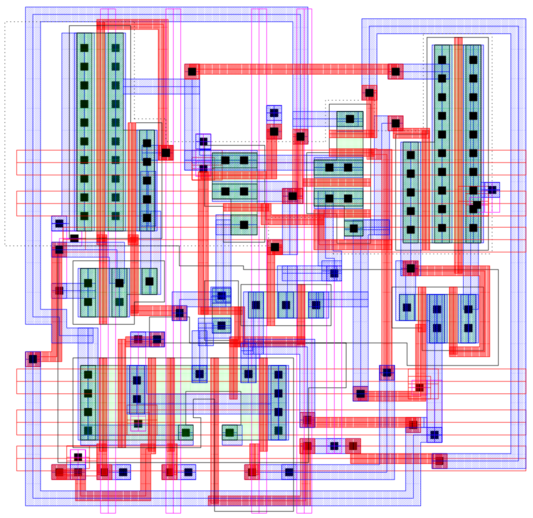
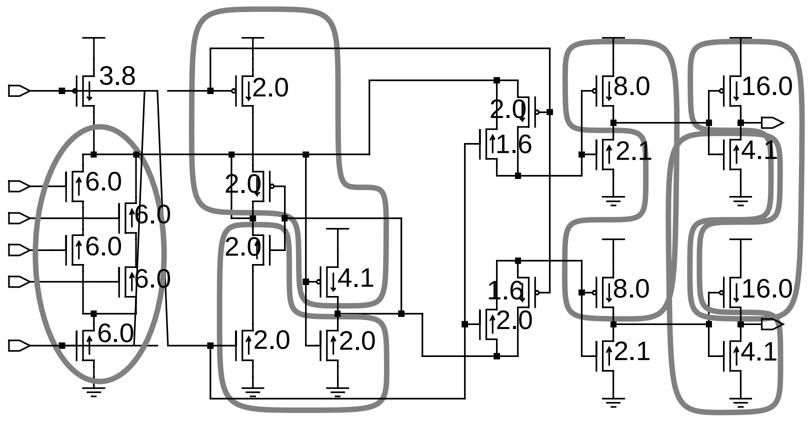
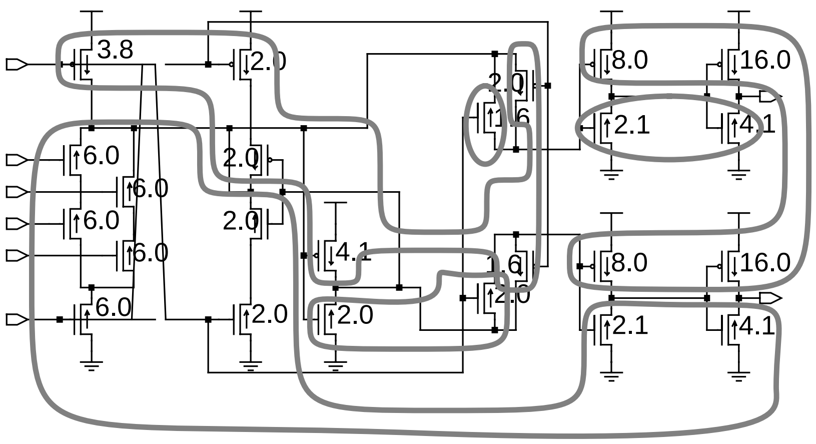

(c) *TEMPO* layout

Figure 58: The MUXFF benchmark (all layouts shown approximately to scale)



(a) manual clustering



(b) TEMPO automatic clustering

Figure 59: The muxff circuit static clusterings used in the manual design and by *TEMPO*

from Cadence Design Systems inc.[1]

As discussed in Section 3.4.2 of Chapter 3, the stochastic simulated annealing optimization algorithm in *TEMPO* will produce a different solution each time that it is run, and the quality of this solution will vary with some statistical distribution. In order to assess the amount of variability in the solutions which are produced we conducted a large number of trials on each benchmark. An initial series of experiments were conducted to determine appropriate values for all of the tuning parameters available in TEMPO (primarily the cluster size upper bound, routing space bloat factor, and the two annealing control parameters). We then ran 100 trials of each benchmark and selected the smallest placed and routed layout for compaction. We will first report results for

1. See Section 1.2 for a more detailed review of the capabilities of LAS and other competing cell synthesis systems.

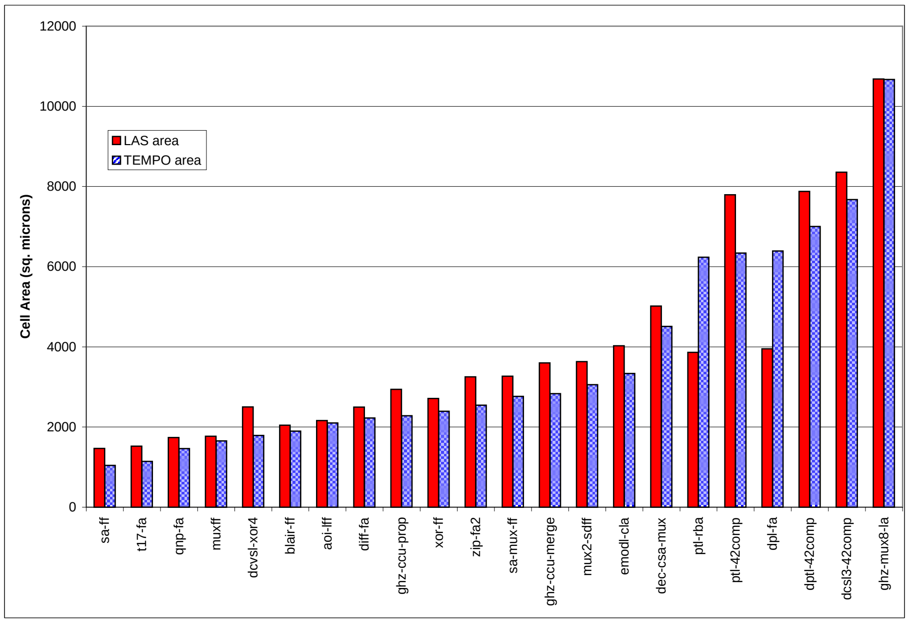
this set of selected layouts, and then discuss the statistical variation that was seen among the complete set of trials. The TEMPO runs were performed on a cluster of thirty 300MHz Pentium-II workstations each with 128 megabytes of memory.

The *LAS* tool has a large number of options which are available for tuning. We ran the tool in full optimization mode which iterates over a large set of these options to find an optimal solution. Among these options are the transistor folding parameter, which controls how large transistors are folded into a set of smaller parallel connected transistors, and the number of rows which are used. The optimization process was given full control over the folding parameter. Two separate runs were conducted, one which specified a target aspect ratio of 1:1 and a second which specified a single row be used. For each benchmark the best of these two runs were selected for comparison with *TEMPO*. Unfortunately *LAS* and *TEMPO* adopt different cell template conventions—*LAS* uses a standard cell template with first-level metal power rails, while *TEMPO* uses a datapath-style template with third-level metal overcell power rails. In order to minimize the bias introduced by the *LAS* power rails we specified that these be sized to minimum width in the LAS layouts. All *LAS* runs were performed on a 170MHz Sun Ultra-1 workstation with 128 megabytes of memory.

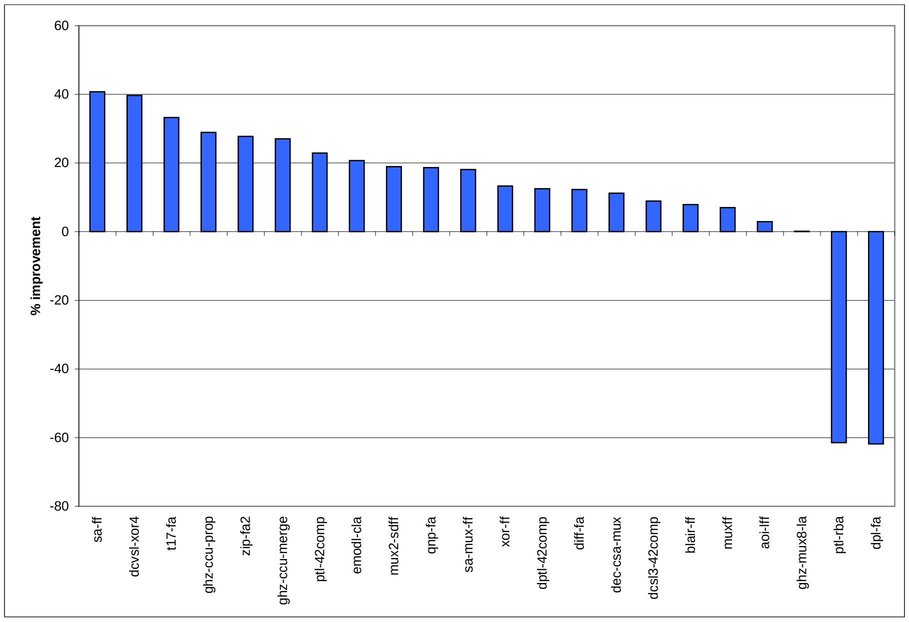
Figure 60 summarizes the results of our experiments. Figure 60(a) shows a comparison of the absolute cell area obtained by *TEMPO* and *LAS* while Figure 60(b) shows the relative percentage decrease in size obtained by *TEMPO*. The numerical data from which these figures were derived is given in Table 8 of Appendix B.

With two significant exceptions, *LAS* produced layouts which were consistently larger than those produced by *TEMPO*, often by as much as 20% to 30%. The reader is referred to Appendices C–E for full-sized layout plots of all benchmark circuits. However, it is instructive to examine several examples in detail in order to obtain some insights into the results.

Figure 61 shows three sample layouts as produced by *TEMPO* and *LAS*. Each pair of layouts are drawn approximately to scale for purposes of visual comparison. The first example in Figure 61(a) shows the dcvsl-xor circuit which, with 23 transistors, is one of the smaller circuits. The *LAS* layout is 39.69% larger in area than the *TEMPO* layout (the *TEMPO* layout could probably be reduced somewhat more if it were hand edited to clean up the poorly routed contact knot on the right-hand edge.) It is clear that in this case the two-dimensional transistor arrangement is much more tightly packed, and yields a layout with much better routing, than the two-row channel routing solution produced by *LAS*.

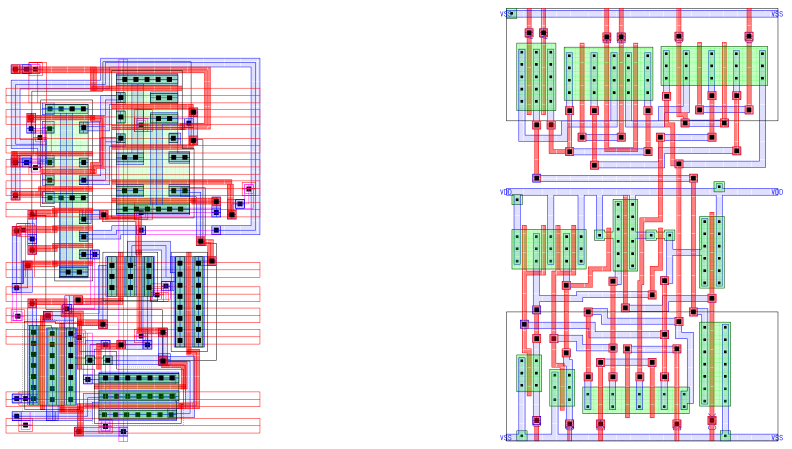


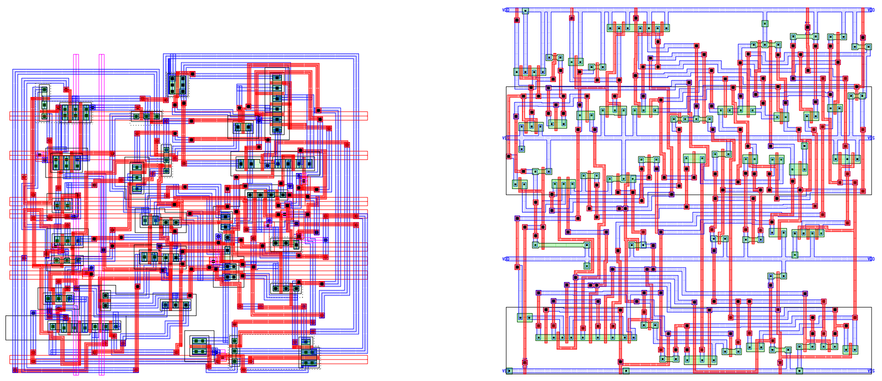
(a) cell area comparison between *TEMPO* and *LAS*



(b) cell area improvement obtained with *TEMPO*

Figure 60: Benchmark cell area data comparing *TEMPO* with Cadence *LAS*


(a) dcvsl-xor4 benchmark


(b) ptl-42comp benchmark

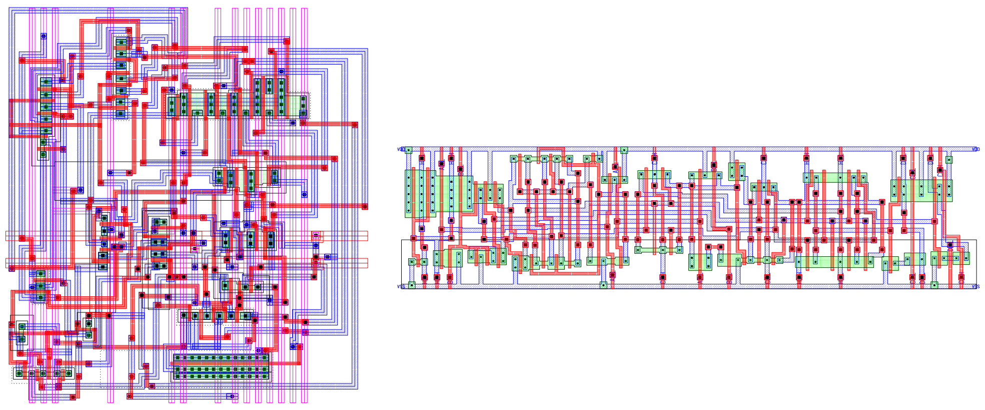

(a) ptl-rba benchmark

Figure 61: Three representative examples comparing layouts produced by TEMPO (left) with those produced by LAS (right). Pairs are shown approximately to scale.

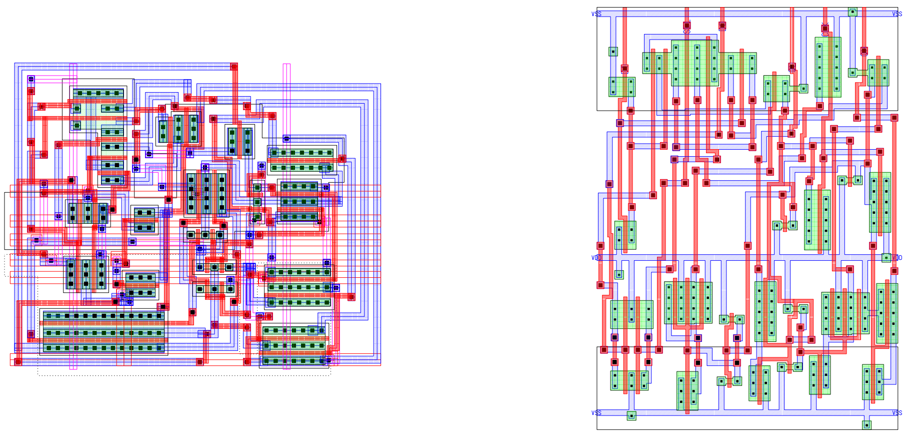
Our second example, shown in Figure 61(b), is the ptl-42comp benchmark circuit. This is one of the larger circuits, consisting of 65 transistors. The high average net fanout of this circuit (7.39, as compared to 4.77 in the dcvsl-xor4 circuit) results in a layout with significantly more routing than the previous example, and the transistor arrangement is visually less compact. Nevertheless *LAS* produced a layout with 22.92% larger area than *TEMPO*.

Our final example, shown in Figure 61(c), is the ptl-rba circuit. This circuit represents one of our two negative results, with *TEMPO* producing a layout 61.45% *larger* than *LAS*. In this case the structure of the circuit allowed LAS to discover an extremely small single-row implementation with very little channel routing and therefore very small cell height. Our layout does not appear markedly worse than any of our other solutions, though there is a large amount of routing. But this example provides a demonstration that high-quality row based solutions can be found for some custom circuits, and that *TEMPO* is currently not able to converge on such a constrained solution style.

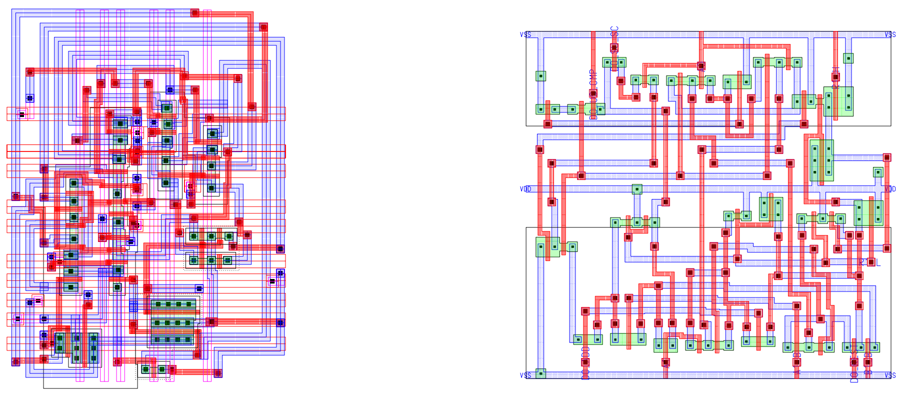
In the absence of routing, *TEMPO* would be capable of finding a very tight packing for the transistors and transistor chains. However, as indicated in the discussion on routing in Section 4.4.6 of Chapter 4, the presence of routing adds a significant complication to the problem. Because the placement and routing steps have been decoupled in our methodology, routing cost estimation and routing space insertion have become two of the most critical problems in obtaining high-quality layout. The effect of routing can be more clearly appreciated through some specific examples.

In Figure 62 we show three example circuits which illustrate some of the issues associated with transistor routing. Figure 62(a) shows the mux2-sdff circuit. This first *TEMPO* layout example demonstrates extremely clean routing with an even routing distribution and relatively narrow routing channels between objects. This clean routing results in a very visually appealing cell layout—in this case the LAS layout is 18.94% larger. This circuit actually has a relatively high average net fanout value of 6.36, so this is not necessarily a good indication of final routing quality.

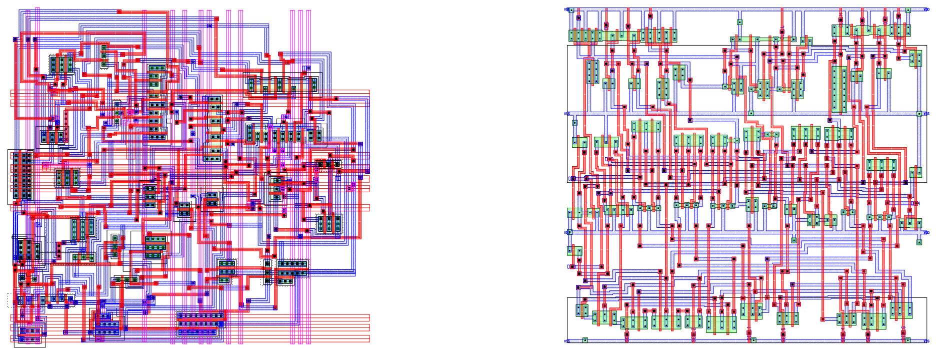
Our second example in Figure 62(b) is the ghz-ccu-prop circuit. While the *TEMPO* layout compares well with *LAS*, (the *LAS* layout is 22.42% larger,) the routing solution is visually less appealing. The first problem to notice is that there is a large amount of routing around the periphery which indicates problems in the routing space estimation process. We note that the ghz-ccu-prop layout is characterized by several high aspect ratio transistor chains made up of near-minimum width devices, while the mux2-sdff layout consists mostly of wider devices and chains with a



(a)mux2-sdff benchmark



(b) ghz-ccu-prop



(c) ghz-mux8-la

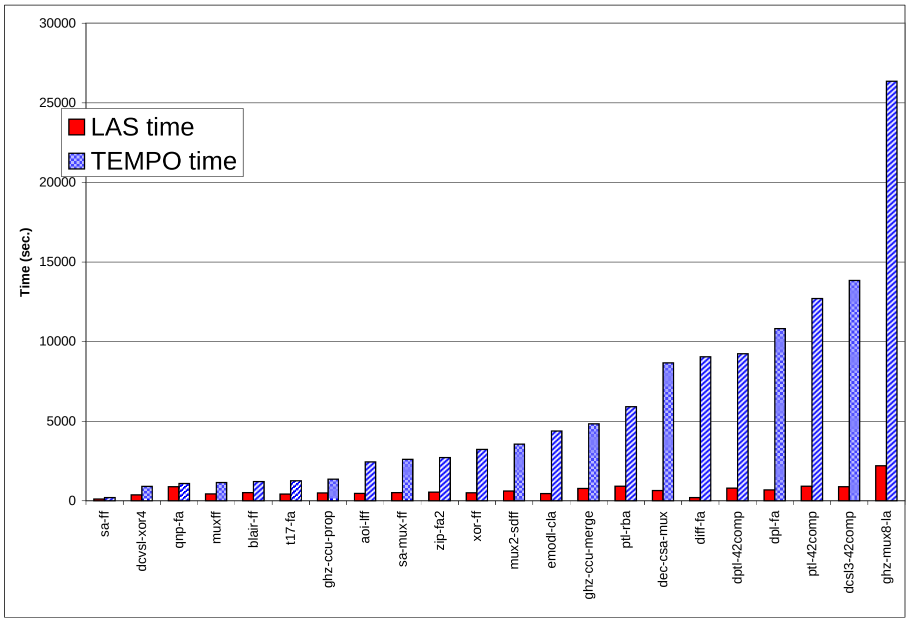
Figure 62: Three benchmarks used to illustrate the effect of routing on layout quality. *TEMPO* layouts are shown on the left with *LAS* layouts on the right.

more even aspect ratio.

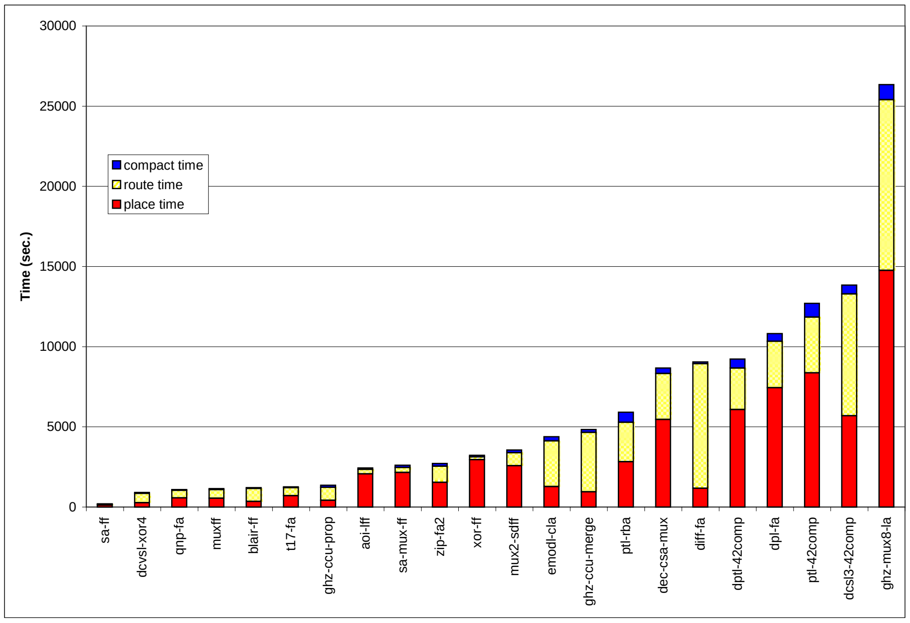
As is also evident when comparing the layouts of Figure 61, layouts with long thin chains of minimum width transistors tend to be less visually appealing than layouts composed of wide aggressively sized transistors. The former case tends to result in layouts with a higher ratio of routing area to active area which probably accounts for this subjective measure of quality. This can be explained by noting that these chains present complex blockages with high incoming routing density which together cause increased local routing congestion in the neighboring routing channels. Our simple routeability estimation algorithm take a more global view and cannot account well for these local effects. It is evident that the routing quality of the ghz-ccu-prop circuit would increase if several of the routing channels were widened and less routing was pushed out to the cell periphery.

Our final layout example is the ghz-mux8-la circuit, which is shown in Figure 62(c). The layout produced by *TEMPO* for this example is virtually identical in size to the *LAS* layout (*LAS* is 0.09% larger.) This is the largest circuit in our benchmark suite, with 84 transistors. Both layouts show a great deal of routing (the average net fanout is relatively high at 6.41), though *TEMPO* hides this somewhat by distributing the routing more evenly between the objects. The size of this circuit and the complexity of the routing make this a challenging problem for any cell synthesis tool. We note that there is a large amount of symmetry in this circuit, as it consists of a flip-flop with eight identical dynamic multiplexor inputs. However, neither *TEMPO* nor *LAS* were able to capitalize on this regularity. As we discuss in Chapter 6, human designers typically take heavy advantage of regularity in the circuit schematic in order to obtain tight visually appealing layouts. Emulating algorithmically this aspect of manual design will be a significant avenue for future work.

We conclude this section with a further analysis of some of the experimental results. Numerical data corresponding to all of the graphs shown in the section can be found in Appendix B. In Figure 63 we show some statistics concerning the runtime performance of *TEMPO* as compared with *LAS*. As shown in Figure 63(a), the *TEMPO* methodology is clearly more compute intensive than *LAS*. Figure 63(b) shows the separate contributions of placement, routing, and compaction to the total *TEMPO* runtime. *TEMPO*'s stochastic exploration of the highly non-linear two-dimensional placement search space, along with *ANAGRAM-II*'s aggressive rip-up and re-routing, requires considerably more effort than the heuristic transistor pairing and chaining algo-

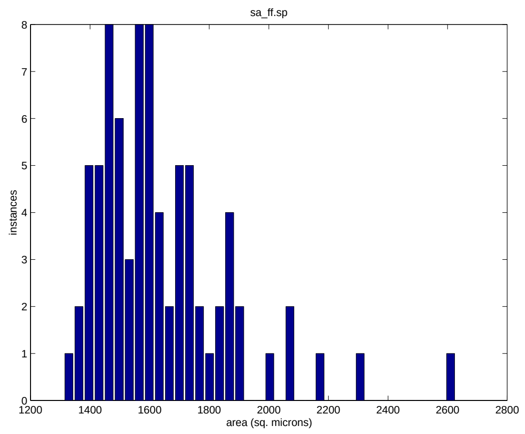


(a) total *TEMPO* runtime compared with LAS runtime

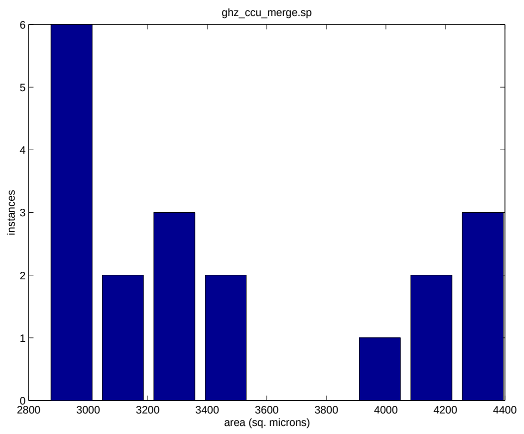


(b) *TEMPO* runtime per phase

Figure 63: Benchmark runtime data comparing the phases of *TEMPO* with Cadence *LAS*



(a) a benchmark displaying a gaussian-like distribution



(b) a benchmark displaying a bimodal distribution

Figure 64: Two representative examples of cell area distribution histograms

rithms and symbolic router used in *LAS*. We note that a single pass through the *LAS* algorithm never takes more than a few seconds, however the outer *LAS* parameter optimization loop requires several dozen passes through the entire process. (This optimization capability generally improves the LAS results by approximately 10%–30%.)

Recall that TEMPO was run 100 times on each benchmark circuit in order to gather statistics concerning the variance in final layout size. In Figure 64 we show histograms for two circuits which demonstrate typical solution area distributions. Most circuits display distributions which are essentially gaussian in appearance, as on the left, though occasionally we observe bi-modal distributions as on the right. This latter case may indicate the presence of two strong local minima, one with larger average area than the other.

In Figure 65 we show a very significant result of our statistical study. For each benchmark we find the mean area δ over all 100 runs and compute the standard deviation σ. We then calculate the number of standard deviations below the mean Δ at which the minimum area solution was found as follows:

$$\Delta = \frac{\delta - \min}{\sigma} \tag{45}$$

This number gives some indication of the number of runs which must be conducted before we can trust that we have located a near minimum solution. If we assume a Gaussian distribution, the number of solutions expected to fall within $\delta \pm \sigma$ is approximately 68%. The number expected to fall within $\delta \pm 2\sigma$ is 95%. In Figure 65 we show the value of Δ calculated for each benchmark.

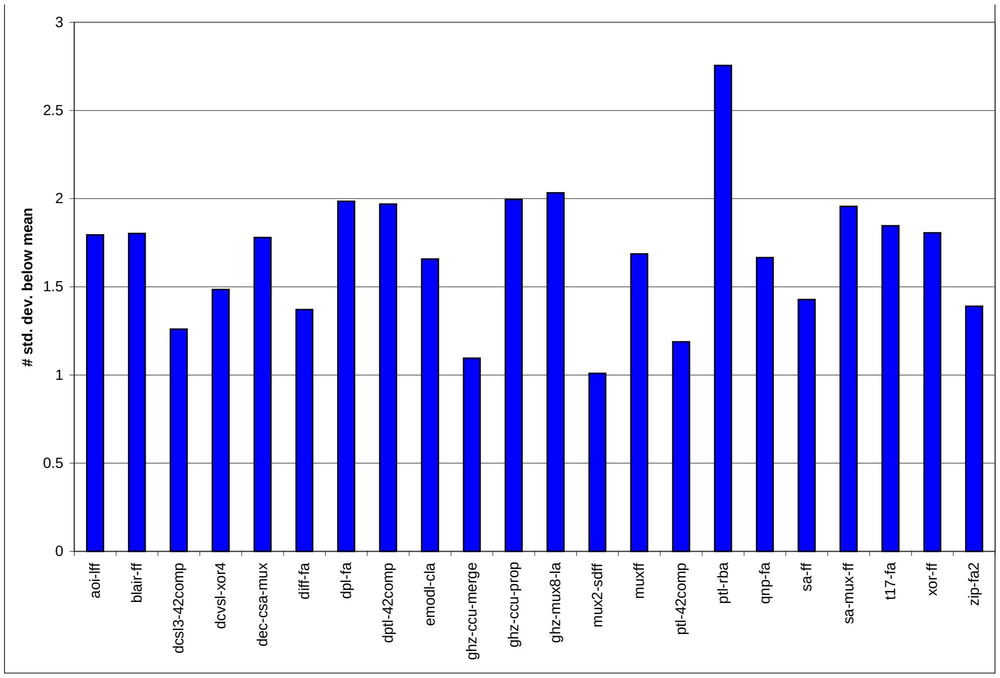


Figure 65: The number of standard deviations below the mean at which the smallest benchmark lies

This number is typically near two, which means that a single trial is likely to find our best solution with a probability of only 2.5%. This would seem to indicate that at least 40 trials should be conducted for each benchmark.

A difficulty which we encountered in conducting these experiments which also motivated the need for multiple trials is the fact that *TEMPO* does not guarantee routeable solutions. Only rough routeability estimates based on the minimum spanning tree routing length are available, and the only control which the user has over routeability is through the routing bloat factor parameter. Our experience indicates that the densest layouts are produced with an aggressive bloat factor setting, but as the value of this parameter is reduced the number of placements aborted by the router will increase. Therefore the user must be willing to pay for high cell quality at the expense of an increased number of trials. In Figure 66 we show the routing completion statistics for each of the benchmark circuits. These numbers range from a high of 79% to a low of only 12%. These results clearly demonstrate the need for more accurate routeability estimation techniques, which will be a significant topic for future work.

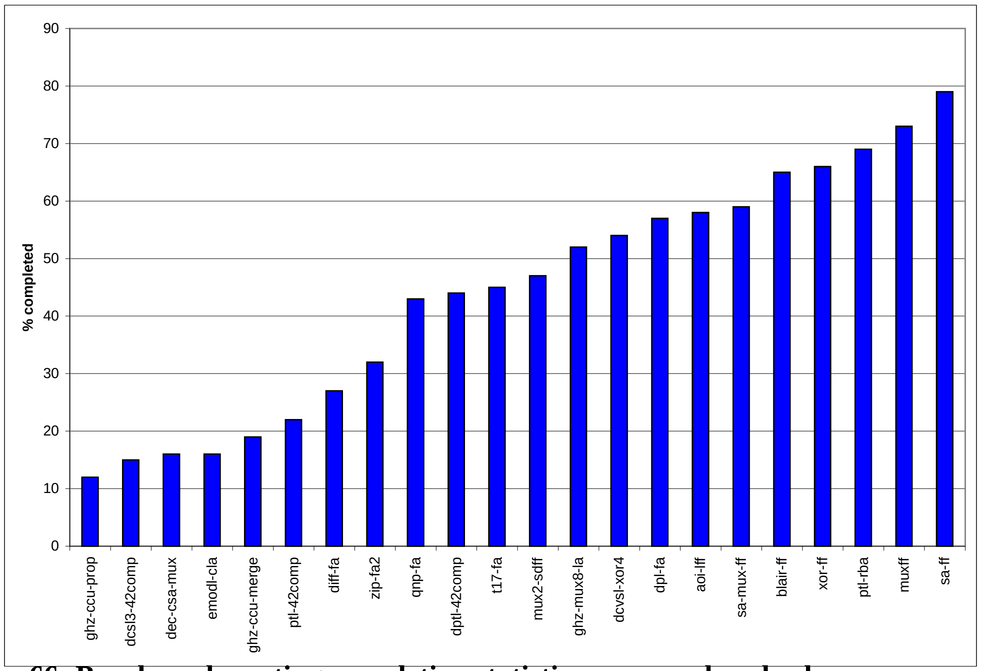


Figure 66: Benchmark routing completion statistics over one hundred runs

5.4 Summary

In this section we introduced a prototype implementation of our methodology for two-dimensional digital cell synthesis. This prototype, *TEMPO*, was used to conduct a series of experiments to gauge the effectiveness of our approach. A set of 22 benchmark circuits were developed as representative examples of a new class of designs for which traditional cell synthesis environments have not been applied. Experiments with these benchmarks resulted in encouraging results. For 20 of the 22 circuits *TEMPO* produced layouts with smaller area than *LAS*, a commercial cell synthesis tool from Cadence Design Systems inc. Furthermore, for 15 of the 22 benchmarks the area improvement exceeded 10%, and on 8of the 22 the area improvement exceeded 20%.

Visual inspection of the layouts produced by *TEMPO* indicate several areas for future work. Cell routing and routeability estimation present difficult problems and there is significant room for improvement, especially for circuits which contain high aspect ratio structures such as long chains of minimum width transistors. These objects represent complex blockages with high input routing density, and this situation can lead to high routing congestion in the neighboring channels.

The most significant disadvantage of the proposed approach is the high computational cost. Several hours of CPU time were required to run all 22 benchmarks through *LAS*, while *TEMPO* required approximate three weeks of computer time on a cluster of 30 workstations to complete all 100 trials for all of the 22 benchmark circuits. This large number of trial was required for two reasons. First, a statistical study indicates that at least 40 trials is required per benchmark in order to account for the variability in solution quality introduced by the stochastic simulated annealing optimization algorithm. Second, a significant numbers of placements produced by *TEMPO* prove to have infeasible routing and are aborted by the *ANAGRAM-II* router, despite its aggressive rip-up and re-route capability. We will have to reduce the computational cost by at least an order of magnitude before this methodology becomes practical.

CHAPTER 6

Conclusions and Future Work

6.1 Summary

In this dissertation we have outlined a new methodology for digital VLSI cell synthesis. Existing approaches to this problem have concentrated primarily on the synthesis of complementary static CMOS circuits using a rigorous highly structured two-row design style based on dual N and P diffusion chain optimization. We refer to these as one-dimensional problem formulations. Extensions to this one-dimensional technique have been developed to handle non-dual transistor pairing and chain formation, allowing limited application to non-complementary logic styles. Two-dimensional techniques have also been developed, though these are generally formulated as extensions to the basic one-dimensional problems which allow optimization across multiple one-dimensional rows.

We began by defining a new cell synthesis problem domain related to the automated layout generation of complex non-dual digital circuits. A growing need has developed for low power and high speed circuits for use in new high-performance integrated circuit designs, and circuits of this type bear little resemblance to the stylized static CMOS circuits for which existing approaches were designed. We provided one example of such as cell, a dynamic CGaAs mux-flipflop used in a high-speed microprocessor design, which illustrated some attributes of circuits of this type: lack of regularity, non-dual ratioed transistors, complex transistor geometry sharing, and non-trivial routing topologies. Cells of this type demand a completely unconstrained true two-dimensional full-custom design style for which existing synthesis approaches are not suited.

Chapter 1 outlined our methodology which is based on a general placement and routing framework. This defines a problem which we term **micro-placement and routing** to distinguish it from the field of chip-level macro-block placement and routing. Existing models for placement and detailed routing were adopted and extended to support the complex needs of transistor-level layout. Central to this methodology is a means for modeling and optimizing transistor geometry sharing. However, the stylized graph-theoretic methods applied to one-dimensional layout styles cannot be applied to the problems which we are targeting.

Our synthesis flow begins with a static clustering stage to discover strongly connected sub-circuits which are good candidates for geometry sharing. We refer to these as **chains**. However, we do not statically optimize the ordering of the transistors within these chains. Instead we defer their optimization to the placement stage. This “late binding” permits chaining decisions to be made at a stage when top-level as well as local routing effects can be measured. We refer to geometry sharing obtained through transistor chaining as **first-order geometry sharing**. We also support more general geometry sharing through a unique adjacency analysis step that allows adjacent objects with electrically compatible ports to be merged simply through the relaxation of their pairwise separation constraints. We refer to sharing of the latter type as **second-order geometry sharing**. Our methodology concludes with the detailed routing of the complete optimized placement solution, and an optional compaction phase to remove excess area un-utilized by the router.

During placement our atomic placeable objects consist of individual unchained transistors, as well as transistor sub-chains. **Sub-chains** are formed by splitting the chains at each diffusion break. Different chaining solutions for a given chain will cause transistors to move from one sub-chain to another, changing the sizes of the sub-chains relative to one another. During placement, in addition to the optimization of object location and orientation, we also explore different chain configurations. Thus the optimization process is capable of locating chaining solutions which simultaneously optimize the relative sizes of the sub-chain objects along with their contribution to the detailed cell-level routing.

In Chapter 2 we examined the low-level modeling issues associated with the generic placement problem. We discussed the distinction between direct and indirect placement models, and for reasons of efficiency selected a symbolic indirect model based on two-dimensional compaction constraints. Our choice was motivated by the observation that direct models, while they provide a smoother continuous solution space, contain a large number of infeasible placements. Through compaction, large numbers of feasible and infeasible placements will map to the same symbolic placement. We reason that a well designed symbolic model, one with the capability to model the universe of feasible and unique solutions, will be exploring a much smaller solution space. The chapter concluded with a discussion of one limitation of our two-dimensional compaction formulation, which we call the **pairwise relative placement** model. It is possible to generate infeasible placements because of a transitivity relationship between the constraints, and this required the use of more expensive constraint satisfaction algorithms capable of detecting these infeasibilities.

Chapter 3 extended the placement modeling discussion by introducing the sequence pair, a recently developed method for representing two-dimensional compaction constraints developed by Murata et al [77]. The sequence pair representation neatly captures the transitivity relationship between the constraints and eliminates the consideration of infeasible placements. We discussed three different formulations for the sequence pair placement optimization problem. Two exact optimization formulations were explored, one based on integer linear programming, and one based on a branch-and-bound traversal of the solution space. A set of pilot studies indicated, however, that these exact approaches were incapable of scaling to problems with more than seven or eight objects. We therefore concentrated our efforts on a heuristic simulated annealing optimization framework. Pilot studies indicated that simulated annealing was capable of obtaining near-optimal placements for problems of one hundred objects and more.

Chapter 4 represents the heart of our work. Here we outlined our techniques for extending the basic simulated annealing placement framework to the problem of transistor-level micro-placement and routing. We began with a look at the theory behind single-row transistor chain optimization. A graph-theoretic model for chain optimization developed by Basaran [5] was described and adapted for our use. We then introduced our algorithm for static circuit clustering which is based on a general FM bipartitioning algorithm augmented with constraints on intra-chain transistor height mismatches. The problems connected with transistor-level micro-placement were examined next. Our method for dynamic transistor sub-chain optimization was discussed, followed by the details of the adjacency analysis step which permits the formation of second-order shared structures.

A discussion of the micro-routing problem revealed one of the more difficult problem with which this methodology is faced. Because we adopt the traditional approach of serialized placement and routing, placement optimization must proceed with the use of loose approximations for the routing cost. Furthermore, in order to guarantee a feasible routing, the placement phase must estimate the routeability of a candidate placement and insert sufficient empty space between the objects. We examined several techniques to address these problems. Routing cost estimation is carried out with a minimum spanning tree approximation, while routing area insertion is carried out with the use of a uniform placement expansion which is proportional to the estimated routing cost. Chapter 4 concluded with a review of our detailed routing and compaction solutions, both of which make use of third party applications: *ANAGRAM-II* from Carnegie Mellon University for detailed routing and *MASTERPORT* from Duet Technologies for compaction.

In Chapter 5 we presented the results of a large-scale set of experiments designed to demonstrate the strengths and weaknesses of our proposed approach. A set of 22 benchmark circuits were assembled to provide test cases representative of our target application area. These benchmarks consist primarily of arithmetic and latching circuits with datapath applications, and included a wide variety of logic families such as static and dynamic CVSL, Pass Transistor Logic (PTL), domino CMOS, and zipper CMOS. Our experiments compared the layout area obtained with our prototype tool, named *TEMPO*, with a commercial cell synthesis tool, *LAS* from Cadence Design Systems inc.

Our experimental results were encouraging. Of the 22 benchmarks, TEMPO obtained smaller layouts in every case but two. More significantly, 15 of the 20 showed area improvements exceeding 10% and 8 of the 22 showed improvements exceeding 20%. A subjective visual assessment of the layouts produced by TEMPO was encouraging as well. Many circuit exhibit tight placements with visually appealing routing solutions which have a manually designed full custom appearance. However, a problem was noted in cells which contain long chains of near minimum width transistors. These high aspect-ratio objects introduce blockages and local routing congestion which are not well modeled by the routing estimation techniques within the placement phase. Addressing these concerns represents a significant avenue for future work.

The runtime performance of a single *TEMPO* synthesis pass is competitive with *LAS*. However, statistical studies indicate that at least 40 trials are required per circuit to account for statistical variance between runs of the stochastic simulated annealing algorithm. Furthermore, placements are occasionally aborted by the router as infeasible, necessitating a larger number of trials for each circuit. Addressing the excessive computation resources currently required by our methodology represents a second significant area for future work.

6.2 Contributions

In this section we enumerate the major contributions provided by this work. A distinction is made between the theoretical modeling contributions and the experimental contributions provided through the implementation of a functioning prototype.

- **Problem formulation**. Our single most significant contribution is the formulation of a new category of the digital cell synthesis problem. Previous approaches targeted static CMOS and dynamic domino CMOS circuit which yield well behaved problem formulations and which can be treated with elegant graph theoretic methods. However, they break down when faced with complex logic families such as static and dynamic CVSL or Pass Transistor Logic (PTL.) We target such circuit families, which are finding increased use in modern high speed and low power designs. Some characteristics exhibited by these difficult circuits are a lack of regularity, non-dual ratioed transistors, complex transistor geometry sharing, and non-trivial routing topologies. We formulate this problem as an unconstrained true two-dimensional full-custom placement and routing problem, a problem which we term *micro-placement and routing*.
- **Problem methodology**. To address this new problem formulation we have assembled a complete end-to-end methodology. At the heart of this methodology lies the detailed modeling and optimization of transistor geometry sharing, which is critical in obtaining compact cell layouts. We begin with a static clustering step which is used to form diffusion connected transistor chains. The chains are split at the locations of their diffusion breaks, and the resulting sub-chains become the atomic placement objects. During the dynamic placement process we optimize the placement and orientation of the objects, as well as ordering, or **chaining** of the transistors in the chains. An adjacency analysis step during placement permits the formation of complex second-order shared structures to further increase geometry sharing. Our methodology concludes with a detailed area routing phase and an optional compaction phase.
- **Dynamic transistor chaining.** Perhaps the single most novel aspect of our methodology is its capability for dynamic transistor chain optimization. The static clustering step performs some initial chaining when the transistor chains are formed. However, by deferring the ordering of the transistors in the chain until the dynamic placement step, we are able to optimize the chain orderings at a time when detailed placement and routing information is available. By splitting the chains at the locations

of the diffusion breaks, the resulting sub-chains are given more freedom to assume an optimal placement.

- **Second-order geometry sharing optimization**. A large percentage of potential transistor geometry sharing is captured through the transistor chaining process, which was inspired by techniques used in traditional digital circuit layout synthesis. However, we also developed an adjacency analysis step utilized during placement which allows for geometry sharing of a more general nature. Complex shared structures, which we call s*econd-order shared structures*, can be formed through this process. This latter idea more closely resembles the geometry merging techniques used by *Koan* [17] for the synthesis of analog cell layout.
- **Placement modeling and optimization**. While we have adopted an existing model for the representation of the placement problem, two dimensional compaction constraints represented with the use of the sequence pair notation, we believe that we are the first to apply this model to problems at the transistor level. In the process of researching the placement problem we developed two new optimal techniques for the optimization of sequence pair placements: one based on integer linear programming, and one based on a branch and bound traversal of the search space. While we ultimately adopted a simulated annealing framework, these optimal formulations provide useful insights into the structure of the problem.
- **Incremental SSLP optimization algorithm**. In connection with our research into the two-dimensional compaction formulation of the placement problem we observed that a single move within the search space perturbs the underlying constraint graph very little. We developed an incremental Single Source Longest Path algorithm for constraint graph satisfaction. This algorithm was shown to provide a small but significant improvement in the asymptotic runtime behavior of the algorithm of approximately 10%.
- **Routing area insertion**. One of the more difficult problems associated with transistor-level micro-placement turned out to be the proper modeling of routing effects. In order to guarantee that a feasible routing exists, extra space must be included for routing. Experiments conducted with several existing techniques proved inadequate. We developed a unique method, based on an earlier technique due to Onodera [84], which

we refer to as the *uniform expansion* technique. While primitive, this technique has proved to be adequate when applied to many problems.

- **Prototype implementation**. In addition to the theoretical and modeling contributions provided in this dissertation, we also developed a prototype implementation, called *TEMPO*, to test our methodology. *TEMPO* implements all of the major ideas connected with our transistor micro-placement methodology. A complete end-to-end system is provided by linking *TEMPO* with an analog detailed router, *ANAGRAM-II* from Carnegie Mellon University, and with an optional cell compactor, *MASTERPORT* from Duet Technologies inc. A series of experiments conducted with this prototype system yielded encouraging results.
- **Circuit benchmarks**. The need to demonstrate our methodology pointed out the lack of a comprehensive set of benchmark circuits of the type which we have targeted. We have assembled a set of 22 benchmark circuits, taken mostly from the solid state circuits literature, which represent a wide cross section of circuit logic families. Static and dynamic CVSL, single and dual ended Pass Transistor Logic (PTL), dynamic and quasi-dynamic domino CMOS, and zipper CMOS are all well represented. These benchmark circuits should prove useful not only in the study of the cell synthesis problem, but also in the fields of logic testing and test vector generation, transistor size optimization, and timing characterization and optimization. They may also be of use to those studying high-performance logic synthesis and logic mapping.

6.3 Future Work

As in any research project, every question we asked yielded multiple possible solutions. And every solution yielded more questions. If every lead were followed this dissertation would never have been completed. However, there are a large number of intriguing questions which remain unanswered, and a number of aspects to our methodology which demand further exploration. The work which culminated in the implementation of *TEMPO* represents only a first attempt at the new problem of high performance digital cell synthesis. In this section we discuss several problems which we intend to address in further work on this subject. We begin with several “grand challenge” problems which will require considerable work, and conclude with a short laundry list of features which we intended to implement but which for one reason or another ended up here.

- **Partial chain symmetry**. In abandoning the dual transistor chain optimization techniques common in early cell synthesis work we made the choice to abandon the regular structure of such circuits. While it is true that many of the circuits which we target present very irregular schematics, it is almost never true that they are completely free of some regularity or symmetry. One component of the “black art” of cell design as practiced by experienced human designers is the clever use of such patterns of symmetry. In the same way that the use of geometry sharing results in reduced transistor source/drain metal routing, the presence of symmetric chain layouts will reduce the complexity of the polysilicon gate routing.

  It is likely that significant gains in layout quality will be realized through the development of automated methods for discovering and exploiting patterns of regularity and symmetry in the designs. This will demand effort both in the static clustering stage of the methodology, which must be responsible for producing symmetric transistor clusters, and in the dynamic chain optimization stage, which must enforce symmetry constraints between chains.

  We point out that it is not necessary for chains to be completely symmetric. If say 4 out of 10 of the gate inputs are shared between two transistor chains, it may be advantageous to enforce a symmetry constraint which maintains the same ordering for these 4 transistors but allows the remaining 6 to assume any ordering in the chain. It is an open question how partial symmetry constraints which involve more than two chains should be handled.
- **Routing space insertion**. The problem of accounting for routing effects within placement is often cited as one of the more difficult aspects of physical design. In order to guarantee a feasible routing it is important that extra space be included in the channels between objects. However, if too much space is inserted the placement quality will be effected as it becomes more distorted, and extra unused space will remain after routing. Post routing compaction can be used to remove some of this space, but if the amount of space in non-uniform the layout may become very distorted during compaction.

Our benchmark experiments demonstrated the need for more powerful routing space insertion techniques which account for the local congestion around high-aspect ratio transistor chains. It is likely that what is needed is some form of global routing analysis which determines the actual path that each net will take in the layout. Care must be taken to keep the cost of this analysis down if it is applied in the inner loop of the placement process, but more expensive techniques can be used if the routing space insertion step is performed only once just prior to routing.

- **Routing cost estimation**. Long transistor chains in the layout present a number of challenges to routing cost estimation as well as routing space insertion. Their high aspect ratio and large fanin create regions of intense routing congestion, and the objects themselves present large blockages in the layout. The simple Minimum Spanning Tree model advocated in this dissertation does not account for the presence of blockages in the design, and the cost function does not seek to even out routing congestion. It is likely that significant gains in routing predictability would be seen if a routing estimate based on a more accurate supply and demand model [122] were developed. However, as these estimates are made in the inner loop of the placement process great care must be taken with their computational overhead.
- **Layout optimization for performance**. Currently, our optimization cost function emphasizes only the minimization of cell size. However, the minimization of performance degradation due to parasitic resistances and capacitances is an important secondary concern. Two of the most significant sources of such parasitics are the polysilicon routing and parasitic source/drain capacitances. The minimization of polysilicon routing must be accounted for in both the placement and routing phases. Parasitic source/drain capacitance is effected by the transistor clustering and chaining decisions, and both of these phases can be augmented to encourage the merging of timing critical electrical nodes.
- **Well contacts and well generation.** The correct generation of diffusion well geometry is a sometimes non-trivial problem. This task is currently performed by the Anagram-II router in our system, which uses a fairly simple heuristic. However, several problems currently exist with this system. The most serious problem concerns the fact that most standard-cell and datapath cells are normally designed to abut one another, implying

that attention has been paid to the well-to-well design-rule spacing at the edges of the cell. Allowances must be made, possibly in the optimization cost function, to encourage legal cell abutment. The positioning of well contacts in the layout should also receive more attention. We currently take a conservative approach and place one well contact on each degenerate chain (i.e. single transistor) object in the layout. Well contact placement in chains is not so simple, and the implementation of this feature in the chain generator remains to be done. Ideally, a simple well contact coverage analysis should be performed during placement and the appropriate chains constructed with the necessary contacts. However, this would probably be too expensive. It is also possible that the router could handle the task of well contact placement.

- **Links to high-level tools**. As a long term goal we would like to explore methods for linking the cell synthesis process to higher level tools in the VLSI circuit synthesis process. In the introduction of Chapter 1 we envisioned a system linking on-demand cell synthesis to front-end logic synthesis and back-end buffer sizing, power reduction, and wire optimization techniques. Such a system would do away with the concept of cell libraries altogether, and create many more opportunities for detailed circuit optimization. However a large number of problems related to cell characterization and timing analysis would need to be addressed.
- **Expansion of cell template support**. If the *TEMPO* tool is to be adopted for widespread use its ability to support more complex cell templates must be increased. Standard cell style power routing should be supported, along with more complex I/O port structures and more tightly controllable cell height/width/aspect-ratio constraints. The required flexibility may require major work on the underlying geometry database and placement methodology.
- **Reduction in execution time**. A topic which requires major attention is the reduction in the computational complexity of our methodology. The execution time of a single pass is actually quite acceptable. However, multiple trials are currently needed to ensure a high quality solution. Future work needs to address the large variance seen in the solutions produced by the simulated annealing optimization engine. The work mentioned above on increasing routing predictability should address the other concern, the large number of placements rejected as infeasible by the router. However, it may

be worth exploring the use of constructive or hierarchical placement approaches in order to reduce the placement time.

- **Transistor folding/fingering.** A technique which is central to many cell synthesis environments is the folding, sometimes also called "fingering", of large transistors [36]. The exploration of different transistor foldings provides an extra degree of freedom which the placement process can use to locate a more optimal transistor arrangement. We intended to address this capability in some form but it remains un-implemented. Instead we allow an optional constraint to be placed on the clustering process which enforces a maximum size variance between transistors in the same chain. The *KOAN* [17] analog placement algorithm contains an annealing move which explores different foldings for large transistors. A similar capability in *TEMPO* would be straightforward to implement as it can simply be treated as an additional object configuration, much like object rotation. It seems likely that transistors within chains should simply be folded to match the average height of the other transistors within the chains.
- **Placement compaction with routing minimization**. As discussed in Section 2.4.4 of Chapter 2, the solution of the placement constraint graph is currently performed in such a way that all objects are compacted toward the lower left corner of the placement. However, objects not on the critical path have some slack in their motion, and it may be advantageous to assign their positions based on secondary criteria such as the minimization of routing length. These techniques are well known in the compaction literature.
- **Annealing control algorithms**. In Section 3.3.3.3 of Chapter 3 we discussed our implementation of the simulated annealing control algorithms. However, more complex schemes are possible which have been shown to reduce runtimes significantly while maintaining solution quality. One cooling scheme due to Lam [56,57], or a simplified version described by Ochotta [83], would be worth examining. In Section 3.3.3.3 we noted a profound difficulty with the stability of our dynamic move selection algorithm, and this feature often must be disabled. This aspect of the annealing control system could also use improvement.
- **Transistor chain generators**. Several aspects related to transistor chain generation are

currently implemented in a non-ideal way. A simple left-edge algorithm is used to perform the intra-chain routing which does not account for the presence of irregular transistor widths within the chain. The best solution here may be to defer the chain routing to the cell-level detail routing, however the *ANAGRAM-II* router cannot make connections to diffusion, so we were forced to perform the routing statically. This has a number of undesirable implications, especially concerning situations when a transistor source or drain port within a chain is only given one escape direction by the router. Another interesting problem in chain generation is the optimization of the vertical positions of narrow transistors within the chain. It may even be advantageous to make the chains flexible so that the vertical transistor positions within the chains could be adjusted to optimize the packing of the chains in the layout. Recent work by Kang & Dai [48] on sequence pair placement of arbitrary rectilinear blocks may prove useful for this problem. One may also wish to examine issues related to the electrical optimization of the transistors within the chains.

6.4 Conclusions

In this dissertation we have defined a new problem in the field of cell-level geometry synthesis: transistor-level micro-placement and routing. This problem is characterized by the unconstrained placement and routing of transistors and is intended for the realization of the mask geometry for complex non-complementary digital circuits. We have presented a complete end-to-end methodology to address this problem which is based on the sequential application of four steps: transistor clustering, placement, routing, and compaction. Central to this methodology is the modeling and optimization of transistor geometry sharing, and a highlight of the work is the introduction of a novel method for performing transistor chaining optimization dynamically during the placement step. Our methodology has been implemented in a prototype tool named *TEMPO* which has been used to conduct a number of experiments. A new set of benchmark circuits was assembled to provide representative samples of several complex CMOS logic families. In most cases, layouts for these circuits produced with *TEMPO* show promising area reductions when compared with a commercial tool. In conclusion, this work can be taken to demonstrate the potential effectiveness of the unconstrained two-dimensional layout topology when applied to complex non-complementary circuit families. However, there is a great deal of potential for further work on many aspects of this problem.

APPENDICES

APPENDIX A

Transistor Sizing Methodology

A.1 Introduction

A.2 Previous Work

The optimization of transistor widths and lengths in a complex non-complementary digital circuit represents a classic non-linear multi-objective constrained optimization problem. A great deal of research has been conducted in the area of transistor sizing for digital circuits [19, 26, 97, 41, 81, 108], an excellent survey of which can be found in Visweswariah [121]. The usual formulation of the problem is the minimization of one of the following functions: delay, area, power dissipation, output rise/fall time, signal integrity, or yield. Some subset of the remaining quantities may also be given upper or lower limits through simple constraints. For example, we may wish to minimize the circuit area with upper bounds placed on the circuit delay and power dissipation.

In general these objective functions are nonlinear in the transistor and wire dimensions. The optimization of delay, in particular, represents a special challenge as it is often specified as a *minimax* objective. A minimax objective is defined as the minimum of the maximum of one or more, possibly dependent, functions, and in this case reflects a goal of minimizing the slowest path delay in the circuit.

The optimization of signal integrity and yield often conflict directly with the minimization of delay, and is not always easy to measure their values or express them directly in the problem. For example, CMOS domino logic is often designed with a small p-channel feedback transistor (the “trickle” gate) designed to maintain a logic high on the dynamic node during static operation. This transistor must be sized to offset the subthreshold leakage current of the pull-down network, but its effect on the falling edge slew rate must be minimized. When optimizing strictly for delay this gate will simply be driven to its minimum size, which may be inadequate. For this signal integrity goal to be addressed by the optimization algorithm, the rate of decay on the dynamic node must be measured and supplied as a constant to the problem.

Using the classification scheme of [121], traditional transistor sizing approaches can be divided into two broad categories: *static tuning* and *dynamic tuning*. Static tuning uses static timing analysis to determine the delay along each path in the circuit. It has the advantage of being fast and input pattern independent, but is usually restricted to simple models for delay and is thus somewhat inaccurate. Dynamic tuning makes use of dynamic, i.e. time domain, simulations of the circuit, and uses sophisticated gradient-based nonlinear optimization techniques to solve the resulting optimization problem. Dynamic tuning methods can make use of highly accurate simulators, as long as they can supply the necessary gradient information (first-order partial derivatives, or "sensitivities", and possibly also second order derivatives, or "Hessians") to the optimization package, and are therefore generally more accurate than static techniques. However this accuracy usually comes at the price of increased run times and limits on the maximum circuit size. They also have the disadvantage of requiring the designer to explicitly specify the input patterns and path delays of interest. It should also be pointed out that nonlinear optimization techniques converge to a locally feasible and stationary point, but no guarantees of global optimality can usually be made. We review several representative examples of each class of transistor sizing techniques below.

An early example of the static tuning technique is *TILOS* due to Fishburn and Dunlop [26]. *TILOS* adopts a gate delay model based on the Elmore delay [23,90]. This is an RC-tree model with linear resistors and grounded capacitors whose values are related directly to the channel widths. Complex gates are collapsed to their equivalent series pullup/pulldown chains. This gate delay model has the advantage of being a *posynomial* function of the gate width, and which therefore, with a simple change of variables, can be converted into a *convex* function. Since convexity is preserved over the arithmetic sum and maximum operators, complete path delays, and the minimax objective function over these path delays, also remain convex. Convex functions have the property that any local minimum is also the global minimum, so convex programming approaches can be used to find optimal solutions to the objective function. In the interest of efficiently however, *TILOS* makes use of the following heuristic algorithm. All transistors are set to their minimum size and a static timing analyzer is used to calculate the delay along each path through the circuit. For each path which doesn't meet its timing goal *TILOS* walks backwards along the path and calculates the sensitivity of the path delay to each transistor size. It then increments the size of the transistor with the largest sensitivity value by some small value. This process is then repeated until the timing goals are met.

A large source of error in the delay model used by *TILOS* is that the timing model assumes step function inputs on all transistor gates, neglecting the true non-ideal shape of the inputs. This has the effect of making the delay of a gate independent of the drive strengths of its fanin gates. Sapatnekar et al [97], using a model developed by Hedenstierna and Jeppson [41], demonstrate a model with a ramp function input which remains posynomial, thus preserving the convexity of the solutions space. Sepatnekar's *iCONTRAST* transistor sizing tool uses a more accurate means of finding the worst-case Elmore delay for complex gates, and makes use of a convex optimization technique based on an interior point method to find a globally optimal sizing solution. It has been shown that this optimal solution can be as little as 1/3 of the area returned by the *TILOS* heuristic when the delay constraints are tight.

A final system of interest is that of Shyu et al [108]. They make use of the *TILOS* solution as a starting point for a generalized nonlinear gradient-based optimization package. While the minimax objective function is convex, it is not differentiable, and this represents a problem for gradient computation. The authors address this with a new technique which they call *generalized gradients*. Note that, unlike the dynamic tuning approaches discussed next, this system is calculating analytic gradients using the Elmore delay model, not making use of more accurate transient circuit simulations.

In contrast to static tuning techniques, which generally adopt some form of the Elmore delay model, dynamic tuning techniques make use of highly accurate transient time-domain simulators such as *SPICE* [78]. The resulting cost functions and constraints are therefore not guaranteed to be convex, and more general non-linear optimization packages must be used. An early example of this technique is *DELIGHT.SPICE* [81] which combines the *SPICE* circuit simulation package with the *DELIGHT* optimization environment. Modifications to *SPICE* were required for the calculation of sensitivity information. The *DELIGHT* optimization engine makes use of the Method of Feasible Directions (MFD) to locate a solution which satisfies the *Kuhn-Tucker* [86] conditions for optimality (feasibility and stationarity). A major strength of this work is its emphasis on the user interface: it supports the notions of both *hard* and *soft* constraints, the latter being a range of bounds on acceptable solutions, with increasing "goodness," which can be adjusted by the user to obtain acceptable solutions.

A more modern example of a dynamic tuning tool is *JiffyTune*, an internal tool at IBM due to Conn et al [19]. *JiffyTune* makes use of a transient simulator called *SPECS* which adopts simplified device models and an event-driven charge transfer algorithm for efficient time domain MOS-

FET simulation. It is reported that *SPECS* runs transient simulations about 70 times faster than a *SPICE*-like simulator while providing acceptable accuracy. In addition, parameter sensitivity information can still generally be supplied in less time that a single *SPICE* transient run. A general gradient-based non-linear optimization package called *LANCELOT* is used to perform the optimization. The non-differentiability of the minimax objective function is handled with a simple change to the objective function:

$$\underset{x \in R^n}{\text{minimize}} \quad \underset{i \in M \equiv \{1, 2, \ldots, m\}}{\text{maximum}} \quad f(x) \tag{46}$$

can be reformulated as

$$\underset{z \in R, x \in R^n}{\text{minimize}} \; (z) \tag{47}$$

subject to the inequality constraints

$$z - f_i(x) \geq 0 \qquad 1 \leq i \leq m\,. \tag{48}$$

It has been reported [121] that JiffyTune was able to tune a circuit with 6,900 transistors (4,128 of then tunable) in about 2 hours of CPU time.

We conclude this review with some relevant observations about the future of the field of transistor sizing which are taken from Visweswariah [121]. While most existing tuning tools are capable of optimizing for a single objective function, it is often the case that users would prefer to specify multiple objectives and see a complete set of *pareto-optimal*[1] solutions from which they may choose. This would require the use of more sophisticated *multi-criteria* optimization methods [111]. In addition, optimization frameworks which include models for circuit reliability and yield are becoming increasingly important. Currently most designers rely on simple practices of *corner analysis* and *design centering,* which can be incorporated in existing systems, but practical implementations of more sophisticated statistical optimization techniques are needed.

A.3 The *Topt* Transistor Optimization Tool

In approaching our transistor sizing problem we immediately rejected the static tuning

1. An assignment to the free problem parameters α^0 is *pareto optimal* if, over all of the objective functions $f_1(\alpha), \ldots, f_k(\alpha)$, there does not exist an assignment α such that $f_i(\alpha) \leq f_i(\alpha^0)$ for all $i \in \{1, \ldots, k\}$, and $f_j(\alpha) < f_j(\alpha^0)$ for at least one $j \in \{1, \ldots, k\}$. See [111, page 4].

approaches. Dynamic tuning has been shown to be considerably more accurate for ordinary series-parallel circuits. In addition, the RC-tree Elmore delay model cannot be applied to some of the more “analog” circuit families—pass transistor logic and the positive feedback portions of CVSL gates in particular. Therefore we have adopted a dynamic tuning approach. However, most of the state-of-the-art dynamic tuning tools are proprietary and we were unable to gain access to them. In order to size our benchmark circuits, and in the interest of keeping our methodology open to the widest possible group of users, we have implemented our own heuristic transistor sizing algorithm.

Our tool, called *Topt*, is implemented as a *PERL* script which iteratively calls *SPICE* in a manner that mimics the manual design process. More formally, *Topt* makes use of a direct search technique for iterated univariate nonlinear optimization. We begin with a description of the input files which are used to specify the optimization problem, which is followed by a detailed discussion of the nonlinear optimization algorithm used in *Topt*.

A.3.1 The *Topt* Input Specification

As an example of the operation of *Topt* we will be using portions of the *SPICE* and *Topt* input files from the “sa_ff” benchmark, which is a dynamic flip-flop used in the Digital Equipment Corporation’s StrongArm™ microprocessor [76].

The user supplies *Topt* with a *SPICE* input file in which the tunable transistors are given parameterized widths and possibly lengths (see Figure 67). A sequence of input vectors must be supplied (see Figure 68) along with one or more named “measure” statements which measure DC or transient behaviors of interest: usually rise/fall delay times and peak or RMS power (see Figure 69). In addition, in order to supply boundary conditions on the optimization, some form of non-tunable load must be placed on each primary output, and a non-tunable driver with finite drive strength (i.e. not a square wave or piecewise linear supply) must be placed on each primary input.

It is important that the input vectors and “measure” statements exercise a rising and a falling transition through each tunable transistor. Otherwise the algorithm will greedily reduce each uncovered transistor width to its minimum in order to reduce its parasitic effects on paths which are measured. The user is currently required to derive this information manually—an interesting avenue for future work is the automation of this test vector generation process.

The *SPICE* input file is supplied to *Topt* along with a control file written in a simple format. This file, shown in Figure 70, begins with a series of “SWEEP” statements that specify the

```
* FF NMOS devices
M1   VSS  CLK      2    VSS  NMOS     W=wn1
M2   3    IN_H     2    VSS  NMOS     W=wn2
M3   4    IN_L     2    VSS  NMOS     W=wn2
M4   3    VDD      4    VSS  NMOS     W=wn3
M5   5    6        3    VSS  NMOS     W=wn4
M6   6    5        4    VSS  NMOS     W=wn4

* FF PMOS devices
M7   5    CLK     VDD  VDD  PMOS     W=wp1
M8   5    6       VDD  VDD  PMOS     W=wp2
M9   6    5       VDD  VDD  PMOS     W=wp2
M10  6    CLK     VDD  VDD  PMOS     W=wp1

* latch NAND #1
M11  OUT_L    6        VDD  VDD  PMOS     W=wp3
M12  OUT_L    OUT_H    VDD  VDD  PMOS     W=wp3
M13  OUT_L    6        7    VSS  NMOS     W=wn5
M14  7        OUT_H    VSS  VSS  NMOS     W=wn5

* latch NAND #2
M15  OUT_H    5        VDD  VDD  PMOS     W=wp3
M16  OUT_H    OUT_L    VDD  VDD  PMOS     W=wp3
M17  OUT_H    5        8    VSS  PMOS     W=wn5
M18  8        OUT_L    VSS  VSS  PMOS     W=wn5
```

Figure 67: *SPICE* file excerpt showing parameterized transistor widths for *Topt*.

```
* measure latch evaluate phase
.MEASURE  TRAN  dly_1   TRIG  V(CLK)  VAL=0.33  RISE=1  TARG  V(OUT_L)
VAL=0.33 FALL=1
.MEASURE  TRAN  dly_2   TRIG  V(CLK)  VAL=0.33  RISE=1  TARG  V(OUT_H)
VAL=2.97 RISE=1
.MEASURE  TRAN  dly_3   TRIG  V(CLK)  VAL=0.33  RISE=2  TARG  V(OUT_L)
VAL=2.97 RISE=1
.MEASURE  TRAN  dly_4   TRIG  V(CLK)  VAL=0.33  RISE=2  TARG  V(OUT_H)
VAL=0.33 FALL=1

*measure latch precharge phase
.MEASURE TRAN dly_5  TRIG V(CLK) VAL=2.97 FALL=1 TARG V(3) VAL=2.00
RISE=1
.MEASURE TRAN dly_6  TRIG V(CLK) VAL=2.97 FALL=1 TARG V(4) VAL=2.00
RISE=1
```

Figure 69: *SPICE* file excerpt showing delay measurement statements for *Topt*.

names of each tunable parameter along with its initial value, upper and lower bounds, and step size[1]. Multiple transistors can be assigned the same parametric width (c.f. `M2` and `M3` in Figure 67) which allows symmetry to be retained in regular circuits and which also reduces the size of the search space.

```
V1 CLK_f VSS PWL
+   0ns      0v,
+   10ns     0v,
+   10.5ns   3.3v,
+   20ns     3.3v,
+   20.5ns   0v,
+   30ns     0v,
+   30.5ns   3.3v,
+   40ns     3.3v

V2 IN_H_f VSS PWL
+   0ns      0v,
+   5ns      0v,
+   5.5ns    3.3v,
+   25ns     3.3v,
+   25.5ns   0.0v,
+   40ns     0.0v

V3 IN_L_f VSS PWL
+   0ns      0v,
+   25ns     0v,
+   25.5ns   3.3v,
+   40ns     3.3v
```

Figure 68: *SPICE* file excerpt showing input waveforms for *Topt*

```
CIRCUIT sa_ff

SWEEP wn1 3.0   0.9  20.0  0.3
SWEEP wn2 1.5   0.9  20.0  0.3
SWEEP wn3 0.9   0.9  3.0  0.3
SWEEP wn4 2.4   0.9  20.0  0.3
SWEEP wn5 4.2   0.9  20.0  0.3
SWEEP wp1 3.3   0.9  20.0  0.3
SWEEP wp2 1.2   0.9  20.0  0.3
SWEEP wp3 3.6   0.9  20.0  0.3

DELAYS dly_1 dly_2 dly_3 dly_4
DELAYS dly_5 dly_6

GROUP wpn1 wp3 wn5
ORDER  wpn1  wp2  wp1  wn4  wn3  wn2
```

Figure 70: *Topt* control file for sa_ff benchmark circuit

Each “DELAY” statement in the control file specifies a group of one or more “measure” statement names from the *SPICE* file. The measurements specified in each delay group are

1. We make use of Mead and Conway [74] style Lambda rules, so our step size is set to the value of Lambda.

summed, and the optimization objective function is taken to be the minimization of the maximum (minimax) over each of these delay sums. The summation of measurement groups facilitates the specification of circuits, such as latches and dynamic logic, in which the delay is specified as the sum of the setup and hold times, or the precharge and evaluate times, respectively. Note that since "measure" statements can measure more than just delay values, one can also specify a multi-objective goal that includes such things as power dissipation and output-high voltage (though all objectives should be scaled to be of approximately the same magnitude).

As a final step, the user assigns a fixed ordering to the transistor size parameters with an "ORDER" statement. Transistor "GROUP" statements can be used to group two transistor size parameters into a single item in this ordering. The purpose served by the "ORDER" and "GROUP" statements are described next as we discuss the nonlinear optimization algorithm which is used in *Topt*.

A.3.2 The *Topt* Nonlinear Optimization Algorithm

Most sophisticated nonlinear optimization algorithms intended for general problems make use of a gradient descent method for performing *multivariate minimization*. Most of the following review material is take from Scales [99]. In general, the goal is to minimize an objective function $F(\bar{x})$ which is a nonlinear function of a vector of *n* continuous independent variables

$$\bar{x} = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}^T . \tag{49}$$

Most multivariate minimization algorithms are *iterative linear search* procedures which perform the following operation until some stopping criterion is met:

$$\bar{x}_{k+1} = \bar{x}_k + \alpha_k \bar{p}_k \tag{50}$$

where $\bar{x}_k$ corresponds to the variable values at the current estimate of the objective function minimum at iteration *k*, $\bar{p}_k$ is a unit vector in the *n*-dimensional space, and α_k is a scalar. At iteration *k* the central problem is to find an appropriate $\bar{p}_k$ vector beginning at $\bar{x}_k$ which will move the estimate closer to the global minimum, and an appropriate value of α_k which will correspond to a minimum of, or a sufficient reduction of, $F(\bar{x})$ in this direction. The algorithm concludes when two conditions are met:

1. the vector of first-order derivatives (sensitivities) of $F(\bar{x})$ is zero, a so-called *stationary point*

2. the matrix of second-order derivatives (the Hessian matrix) of $F(\bar{x})$ is positive semi-definite.

Such a point represents a local minimum where the function is non-increasing in all directions.

Gradient descent methods choose $\bar{p}_k$ based on an analysis of the first-order, and possibly second-order derivatives, and seek to move the solution in the direction of steepest descent. It should be noted that this algorithm is not guaranteed to locate a global minimum, only a local minima. In particular, if the algorithm encounters a stationary point along $\bar{x}_k + \alpha \bar{p}_k$ which is a local maximum (or "saddle-point") it must make a heuristic decision concerning which direction away from the maximum to move. If the wrong decision is made, the choice may trap the algorithm in a local minimum.

Multivariate optimization methods can be quite computationally intensive, as every iteration requires information about the derivatives of the objective function with respect to every independent variable. If the objective function is specified in analytic form it may be possible to calculate the derivatives analytically. If this is not possible, and the objective function value at every point must be solved numerically, the algorithm must calculate the first-order derivatives using the *central difference approximation*:

$$\delta F / \delta x_j \cong \frac{F(\bar{x} + \varepsilon_j e_j) - F(\bar{x} - \varepsilon_j e_j)}{2\varepsilon_j} \tag{51}$$

where ε_j is a sufficiently small scalar and e_j is the unit vector in the *j*-th coordinate direction. If this information is not available directly from the numerical solver it will require $O(n)$ calls to the solver in order to calculate it. The calculation of second-order derivatives is even more expensive.

In our transistor sizing problem the objective function seeks to minimize the transient delay of the circuit by optimizing the transistor width and length parameters. The objective function can only be solved with the use of a time domain circuit simulator such as *SPICE*. Topt makes use of the *HSPICE* simulator from Avant! inc. which does not make derivative information available to the user. In the interest of minimizing the number of circuit simulations which must be run, we seek to avoid the $O(n)$ cost per iteration which is associated with the gradient calculation. Our method reduces this cost to $O(1)$ by adopting a simpler alternative to the multivariate optimization approach: *univariate optimization*.

Univariate optimization[1] is defined as the optimization of a function of only one indepen-

1. See [98] chapter 2.

dent variable. When applied to a multivariate optimization problem, however, we interpret this as an optimization step in which the vector $\bar{p}_k$ is non-zero for only a single variable. In other words, at every iteration the algorithm follows a gradient along which only a single variable is changing—all other variables are held constant. The selection of this single free variable could be made by selecting the variable with the largest sensitivity, but this would still require the calculation of all n first derivatives. In order to avoid the expensive derivative calculation step we adopt an even more constrained method of free variable selection. In our method, which we call *iterated univariate optimization*, we simply iterate through the variables in a fixed user-supplied order. The objective function is separately optimized with respect to each variable in the given order, and the process is repeated until it converges on a local minimum. Pseudo-code for this process is shown below.

In the *Topt* pseudo-code, the function `INITIALIZE()` orders the parameters as specified by the user with the "ORDER" directive in the Topt control file, and sets them to their initial values. The function `SIMULATE()` calls the *SPICE* simulator and returns the maximum of the measurement value sums specified by the user with the "DELAY" directives in the control file. Similarly, `SIMULATE-UP()` and `SIMULATE-DN()` call the simulator with the selected parameter either incremented or decremented by its step size. The three simulation calls at lines 4–6 of `ITERATED-GRADIENT-DESCENT()` effectively allow the gradient of the cost function, with respect to the single free variable, to be calculated by the central difference approximation. The while loop beginning at line 15 then follows this gradient until a local minimum is found. After this process is repeated for every variable, `ITERATED-GRADIENT-DESCENT()` returns. The `TOPT()` main loop repeats this entire process until no change is observed in the parameters. Note that in this pseudo-code we do not show the bookkeeping required to keep track of the current or the optimal value of the vector of parameters.

One way to view the *Topt* algorithm is by noting that, in each iteration of the loop starting at line 2 in `ITERATED-GRADIENT-DESCENT()`, the algorithm is following the 1-dimensional cost surface which lies between the initial solution estimate and the final locally optimal solution. As an added feature, *Topt* also allows two free parameters to be explored simultaneously. By extension this would be referred to as *bivariate optimization*, and it can be viewed as the exploration of a two-dimensional cost surface. This option is exercised through the use of the "GROUP" directive in the Topt control file (see Figure 70) and is useful for speeding up convergence when optimizing inverters, complementary pass gates, and length-two series chains.

```
TOPT (Circuit C, Parameter_List P)
1     P ← INITIALIZE(P)
2     done ← FALSE
3     while done = FALSE
4         temp ← P
5         ITERATED-GRADIENT-DESCENT(C,P)
6         if temp = P
7             done ← TRUE
8     return
```

```
ITERATED-GRADIENT-DESCENT(Circuit C, Parameter_List P)
1     for i ← 1 to length[P]
2         done ← FALSE
3         val ← SIMULATE(P)
4         val_up ← SIMULATE-UP(P,i)
5         val_dn ← SIMULATE-DN(P,i)
6         if val_up < val
7             dir ← UP
8             val ← val_up
9         else if val_dn < val
10            dir ← DOWN
11            val ← val_dn
12        else
13            done ← TRUE
14        while done = FALSE
15            if dir = UP
16                new_val ← SIMULATE-UP(P)
17            else
18                new_val ← SIMULATE-DN(P)
19            if new_val < val
20                val ← new_val
21            else
22                done ← TRUE
23    return
```

In order to implement the bivariate optimization feature, in the pseudo-code at lines 4–6 of `ITERATED-GRADIENT-DESCENT()` where three simulations are required to determine the gradient, two grouped parameters will require nine simulations. During the following **`while`** loop at lines 16–19 which then follows this gradient, instead of performing a single simulation along this same gradient we perform five more simulations to calculate a new gradient direction (three of the simulations would take us back to the same point as the previous iteration, so they do not need to be repeated.) Note that we are no longer performing linear search, instead we are finding a new

```
optimizing parameter: wp1
    curr=3.3 delay=1.19582e-09 power=5.2377E-04
    trying wp1=3 delay=1.20821e-09 power=5.2062E-04
    trying wp1=3.6 delay=1.18695e-09 power=5.2479E-04
    new=3.6 delay=1.18695e-09 power=5.2479E-04
optimizing parameter: wp1
    curr=3.6 delay=1.18695e-09 power=5.2479E-04
    trying wp1=3.9 delay=1.18169e-09 power=5.2687E-04
    new=3.9 delay=1.18169e-09 power=5.2687E-04
optimizing group: wpn1 {wp3,wn5}
    curr=(3.6,4.2) delay=1.20646e-09 power=5.0859E-04
    trying {wp3,wn5} = {3.3,3.9} delay=1.21708e-09 power=5.0307E-04
    trying {wp3,wn5} = {3.3,4.5} delay=1.20867e-09 power=5.0812E-04
    trying {wp3,wn5} = {3.3,4.2} delay=1.21161e-09 power=5.0563E-04
    trying {wp3,wn5} = {3.6,3.9} delay=1.21316e-09 power=5.0591E-04
    trying {wp3,wn5} = {3.6,4.5} delay=1.20188e-09 power=5.1122E-04
    trying {wp3,wn5} = {3.9,4.2} delay=1.20333e-09 power=5.1144E-04
    trying {wp3,wn5} = {3.9,3.9} delay=1.21106e-09 power=5.0865E-04
    trying {wp3,wn5} = {3.9,4.5} delay=1.19917e-09 power=5.1420E-04
    new=(3.9,4.5) delay=1.19917e-09 power=5.1420E-04
optimizing group: wpn1 {wp3,wn5}
    curr=(3.9,4.5) delay=1.19917e-09 power=5.1420E-04
    trying {wp3,wn5} = {3.6,4.8} delay=1.19995e-09 power=5.1381E-04
    trying {wp3,wn5} = {3.9,4.8} delay=1.19719e-09 power=5.1692E-04
    trying {wp3,wn5} = {4.2,4.5} delay=1.199e-09 power=5.1708E-04
    trying {wp3,wn5} = {4.2,4.2} delay=1.20323e-09 power=5.1422E-04
    trying {wp3,wn5} = {4.2,4.8} delay=1.19657e-09 power=5.1993E-04
    new=(4.2,4.8) delay=1.19657e-09 power=5.1993E-04
```

Figure 71: An excerpt from a Topt optimization run of the sa_ff circuit

gradient at each step in an effort to find the exact local minimum of the two dimensional cost surface in the neighborhood of the starting point.

The operation of *Topt* is demonstrated in Figure 71. First we show the initial gradient calculation for a single parameter wp1, followed by one step along the calculated gradient. Next we show the calculation of the initial gradient for the parameter wpn1, which groups parameters wp3 and wn5, followed by a second gradient calculation of the same group.

The *Topt* algorithm is somewhat primitive and possibly expensive in terms of the number of calls to *SPICE*, but it is extremely flexible and has been found to work well in practice. Its convergence on a global solution is not guaranteed, but in our experience with the tool we have not noticed convergence to be a problem. We justify this statement with the following analysis.

The global objective function is highly non-linear. However the one-dimensional cost surface of each delay function with respect to a single free parameter, while not guaranteed to be convex, will in general have a single minimum and remain non-decreasing in the neighborhood of that minimum. In addition, the minimax of the sum over several of these functions will retain this prop-

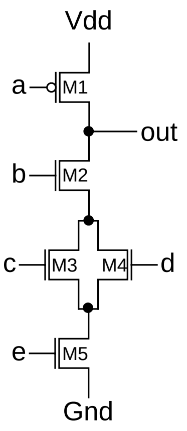


Figure 72: A subcircuit of the aoi_lff benchmark used to produce Figures 73 and 74.

erty and also have a single easily found minimum. In fact, we find that the minimax objective function usually has its minimum at a discontinuity where a falling delay and a rising delay at some critical output meet.

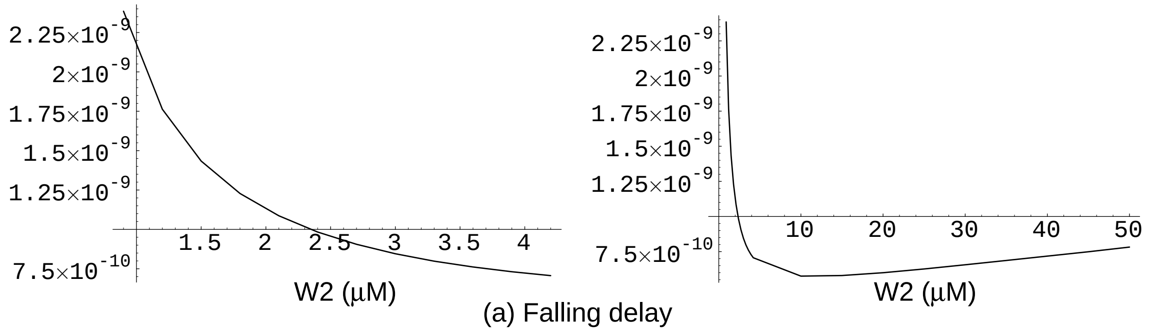
Our claim can be justified intuitively from the observation that when an individual transistor is given a size away from this local minimum:

- as its size is increased the loading on its driving gate increases monotonically, resulting in a slower input slew rate and increased delay.
- as its size is decreased its ability to drive its output load will decrease monotonically resulting in a slower output slew rate and increased delay

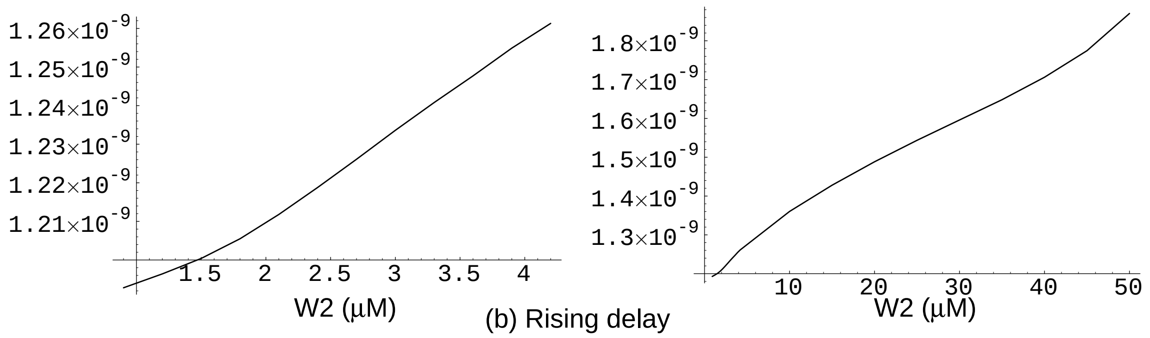
As the path delays in the circuit depend in a nonlinear fashion on both the input and output slew rate of a particular gate, the *monotonicity* of the delay measurement cannot be guaranteed, but it is in general guaranteed to be *non-decreasing* as it moves away from the local minimum.

We provide several examples to support the above claim. In Figure 72 we show part of the output stage of the aoi_lff benchmark circuit. In Figure 73 we show the falling and rising delays of the circuit with respect to the width of transistor M2. On the right we show a large scale plot varying the width up to 50 µm. On the left is a finer resolution curve over more reasonable sizes near the optimum. One can see that the curves, while not convex, are non decreasing on either side of the minimum. In Figure 73(c) we superimpose the two curves and it is easy to see that their minimax has a single minimum at 1.8µm.

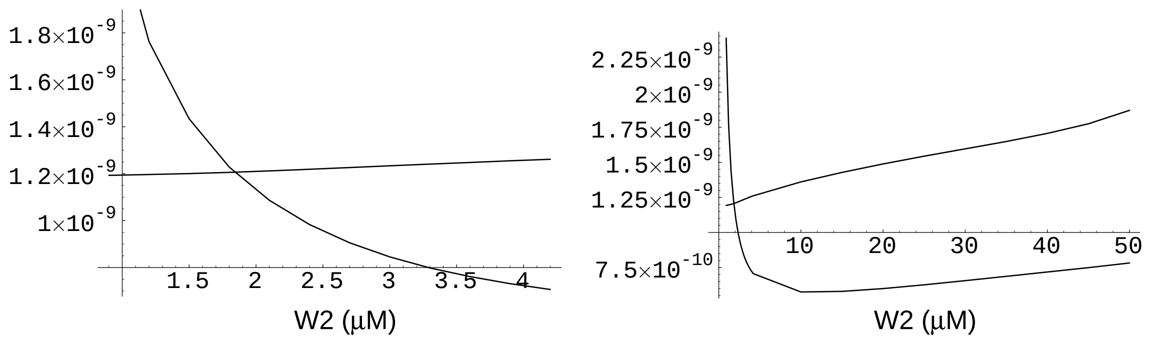
We also argue that the quality of the optimization is retained for paired transistors over their 2-dimensional cost surface. Using the same example circuit, in Figure 74 we show the delays



(a) Falling delay



(b) Rising delay



(c) Rising and falling delays superimposed

Figure 73: HSpice simulations showing the circuit rising and falling delays as a function of the width of transistor M2 in Figure 72.

as a function of the widths transistors M2, M3, and M4 (M2 and M3 are given the same width parameter.) One can see the minimax of the falling and rising delay curves has a single global minimum.

In Figure 75 and Figure 76 we show an example of the dynamic behavior of *Topt* during the course of an optimization. In the first plot we show the values of three transistor width parameters in relation to the objective function. An example of a single parameter as well as two paired parameters are shown. One can see the points during the iteration when the sensitivity of each

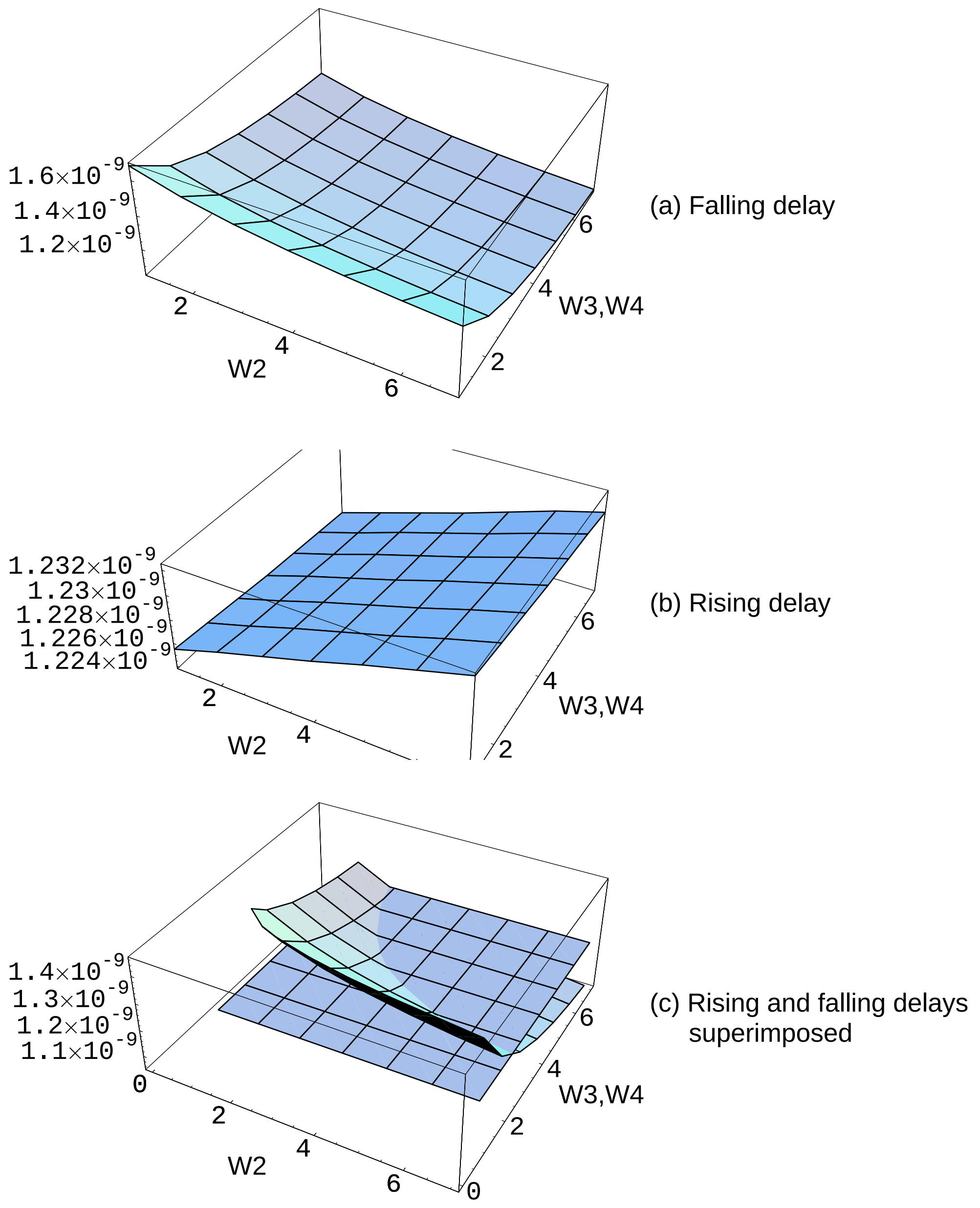


Figure 74: HSpice simulations showing the circuit rising and falling delays as a function of the widths of transistors M2, M3 and M4 in Figure 72. Transistors M2 and M3 are assigned the same width, and all width values are normalized to lie between 0 and 8, with the optimum at 4.

parameter is being evaluated. The width of transistor M15 in particular shows a dramatic jump at one point during the process. The peaks in the delay curve show the simulation at points which represent attempts to move in the opposite direction of the gradient. The second plot shows the

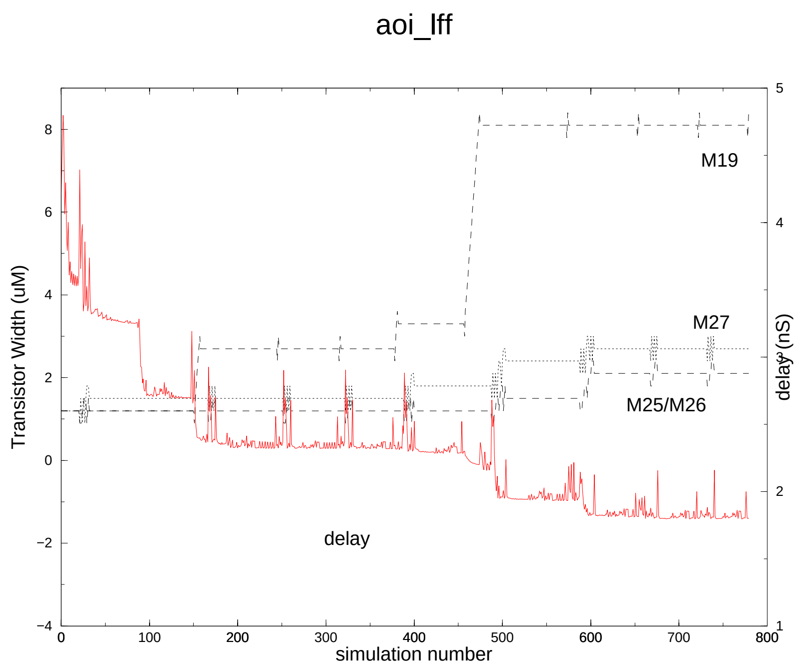


Figure 75: An example showing the dynamic behavior of the Topt transistor optimization tool. Dynamic size information is shown for M19, a single transistor, and the group M27, M25/M26 in which M25/M26 are parallel connected with the same width and are in series with M27.

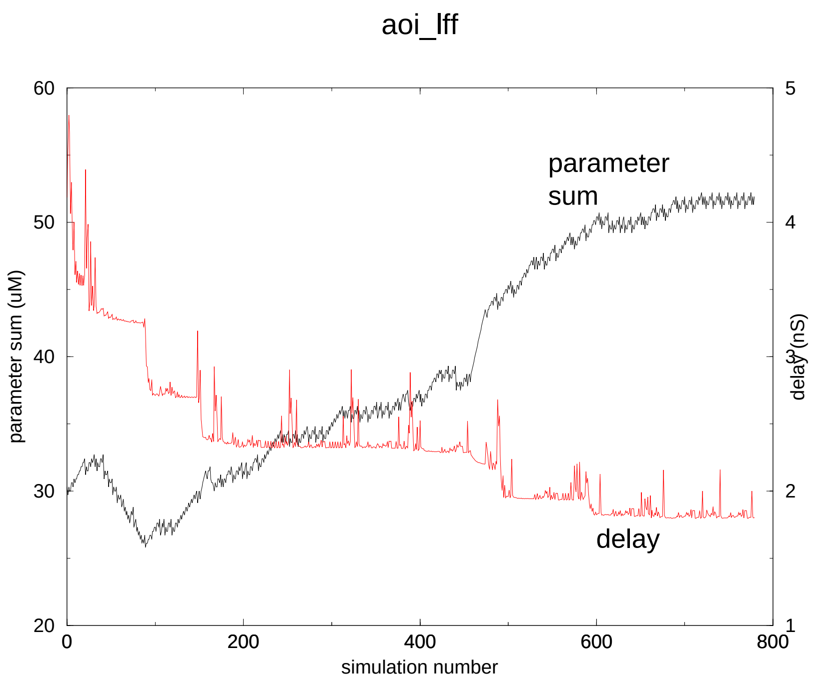


Figure 76: An example showing the relationship between the sum of the tunable transistor widths and the minimax of the path delays in the circuit.

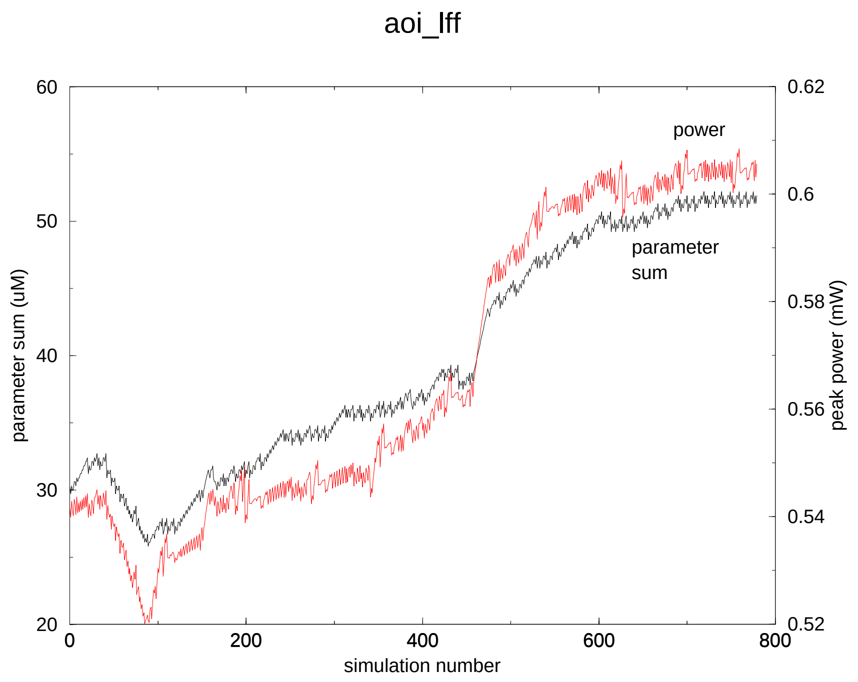


Figure 77: An example showing the relationship between peak power consumption and the sum of the tunable parameters in the circuit. This sum is a reasonably good estimate of the power

objective function and the sum over all of the parameter values. This plot provides a good summary of the overall progress of the simulation. Notice that this sum is not always an increasing function, it can easily explore situations in which transistors should be reduced in size to speed up the circuit. Figure 77 demonstrates that the sum over the parameter values is generally a good approximation of the peak power required by the circuit.

We conclude with some notes on the use of the algorithm. We have found that the best results are obtained when the user iterates through the parameters roughly in order from the primary outputs toward the inputs. The reason for this is that the delay of a gate depends more strongly on its output load than the strength of its driver (recall that *TILOS* neglects the effect of the driver altogether.) It therefore makes more sense to attempt to determine a transistor's load before that transistor is evaluated. The choice of good starting values for each width parameter can greatly aid convergence.

Currently the algorithm attempts to optimize delay as the only objective. However, a log of parameter values at each iteration is kept, and the user can make use of this data, e.g. Figure 76, to stop the algorithm at any point they judge to be a good trade-off between area and delay. In this example we may wish to stop the process at approximately iteration number 500 as the delay has very nearly reached its optimal value. In our experiments we are interested in exploring the optimization of circuits with a wide range of transistor sizes, and in order to avoid arbitrary stopping criteria selection, we have chosen to take our benchmarks transistor sizes from the solution at the final stationary point.

APPENDIX B

Experimental Data

This appendix contains the detailed data associated with the experiments of Chapter 5. Table 7 gives some relevant statistics concerning the size and complexity of each benchmark. The number of transistors and number of nets are given with the average net fanout. Net fanout is reported both for all nets, and for all signal nets (excluding power and ground nets.) This data corresponds to the charts in Figure 56.

Table 8 gives experimental circuit area data comparing *TEMPO* and Cadence *LAS*. Over the 100 experiment runs, the table gives the minimum and average placement area, from among the routeable solutions, returned by *TEMPO* before compaction. We also show the standard deviation among the routeable placements, and the number of standard deviations below the mean at which the minimum placement was located. Finally we show the post-compaction final area of the best solution, compared with the area obtained with Cadence LAS. This data corresponds to the charts in Figure 68.

Table 9 gives some execution statistics for the circuit benchmark experiments. We show the percentage of the 100 experiment runs that yielded routeable placements, along with the average runtime for each phase of the cell synthesis process (placement with *TEMPO*, routing with *ANAGRAM-II*, and compaction with Cascade *MASTERPORT*.) This runtime data is taken from the minimum area solution. For reference we show our total runtime with the Cadence *LAS* runtime. This data corresponds to the charts in Figure 63.

Table 7: Benchmark Circuit Characteristics

Name	#trans	#nets	Avg. net fanout	Avg net fanout (–vdd/gnd)
aoi-lff	30	24	5.38	3.55
blair-ff	20	16	5.50	3.86
dcsl3-42comp	70	47	6.36	4.60
dcvsl-xor4	23	22	9.77	3.55
dec-csa-mux	47	34	6.03	4.06
diff-fa	30	20	6.60	4.67
dpl-fa	48	22	9.27	6.90
dptl-42comp	65	41	6.76	4.49
emodl-cla	35	33	4.84	3.55
ghz-ccu-merge	30	25	5.56	3.73
ghz-ccu-prop	32	32	4.53	3.27
ghz-mux8-la	84	56	6.41	4.20
mux2-sdff	33	22	6.36	4.00
muxff	24	21	5.05	3.47
ptl-42comp	63	36	7.39	4.88
ptl-rba	56	32	7.50	5.20
qnp-fa	23	21	4.81	3.37
sa-ff	18	14	5.64	3.92
sa-mux-ff	40	32	5.36	3.67
t17-fa	21	14	6.50	5.08
xor-ff	33	24	5.88	3.86
zip-fa2	40	33	5.21	3.68

Table 8: Benchmark circuit area data

Name	min. area	avg. area	std. dev.	# std. dev.	final area	LAS area	% area decrease
aoi-lff	2275	2931	365.6	1.79	2100	2161	2.90%
blair-ff	1858	2381	290.1	1.80	1897	2047	7.91%
dcsl3-42comp	9878	11252	1090.1	1.26	7072	8356	8.92%
dcvsl-xor4	1895	2286	263.3	1.49	1789	2499	39.69%
dec-csa-mux	4782	5454	377.6	1.78	4511	5017	11.22%
diff-fa	2449	3011	409.9	1.37	2224	2497	12.28%
dpl-fa	7305	9087	897.4	1.99	6392	3950	–61.82%
dptl-42comp	9009	10738	878.1	1.97	7003	7879	12.51%
emodl-cla	3638	4275	384.3	1.66	3336	4027	20.71%
ghz-ccu-merge	2858	3500	586.0	1.10	2832	3598	27.05%
ghz-ccu-prop	2746	3472	363.9	1.99	2280	2939	28.90%
ghz-mux8-la	14709	17126	1189.0	2.03	10670	10680	0.09%
mux2-sdff	3152	3637	480.2	1.01	3054	3632	18.93%
muxff	1851	2279	253.8	1.69	1654	1770	7.01%
ptl-42comp	7934	8828	752.0	1.19	6339	7792	22.92%
ptl-rba	6610	8690	755.0	2.75	6237	3863	–61.45%
qnp-fa	1468	1840	223.3	1.67	1463	1736	18.66%
sa-ff	1312	1637	227.5	1.43	1041	1465	40.73%
sa-mux-ff	3780	4701	470.8	1.96	2765	3266	18.12%
t17-fa	1199	1587	210.1	1.85	1143	1523	33.25%
xor-ff	3072	3819	413.4	1.81	2394	2713	13.32%
zip-fa2	2888	3485	429.4	1.39	2546	3253	27.77%

Table 9: Benchmark circuit execution data

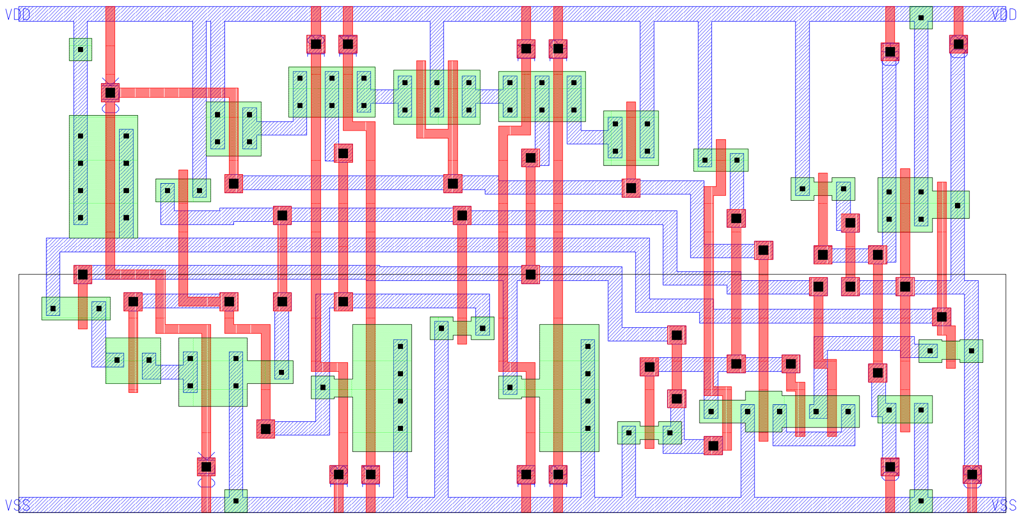
Name	complete	TEMPO time	ANAGRAM time	MASTER PORT time	total time	LAS time
aoi-lff	58%	2066	281	94	2441	462
blair-ff	65%	347	809	53	1209	510
dcsl3-42comp	15%	5697	7596	545	13838	888
dcvsl-xor4	54%	272	572	61	905	368
dec-csa-mux	16%	5463	2863	342	8668	639
diff-fa	27%	1171	7763	116	9050	198
dpl-fa	57%	7443	2896	475	10814	685
dptl-42comp	44%	6085	2584	562	9231	797
emodl-cla	16%	1277	2847	258	4382	454
ghz-ccu-merge	19%	951	3704	183	4838	776
ghz-ccu-prop	12%	416	817	120	1353	490
ghz-mux8-la	52%	14764	10639	947	26350	2203
mux2-sdff	47%	2575	811	171	3557	618
muxff	73%	554	530	60	1144	428
ptl-42comp	22%	8372	3474	854	12700	915
ptl-rba	69%	2831	2455	623	5909	918
qnp-fa	43%	567	474	47	1088	881
sa-ff	79%	123	39	40	202	113
sa-mux-ff	59%	2153	311	141	2605	508
t17-fa	45%	706	499	51	1256	420
xor-ff	66%	2960	169	99	3228	503
zip-fa2	32%	1539	1003	170	2712	546

APPENDIX C

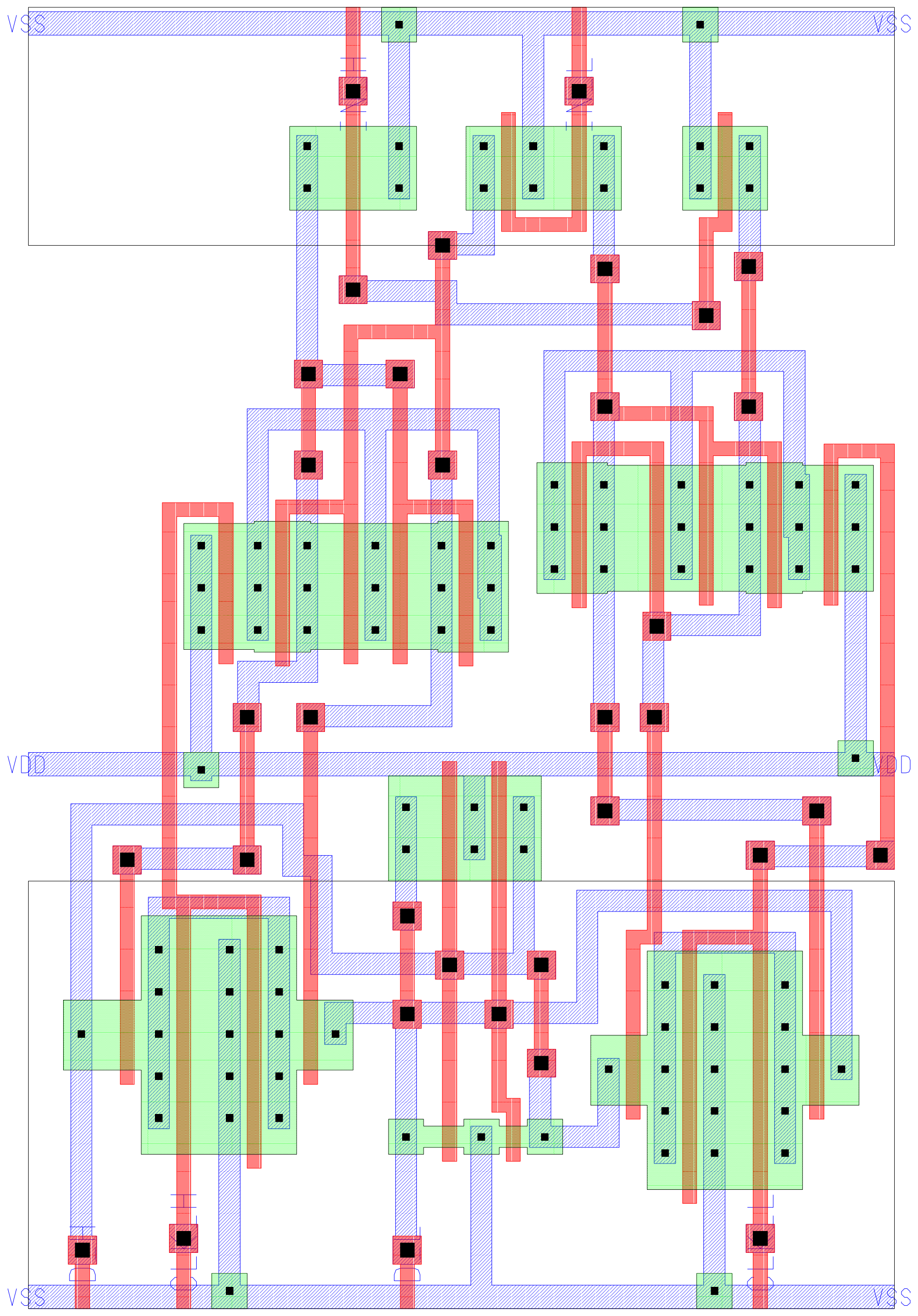
LAS Layouts

This appendix contains the benchmark layouts produced by the LAS (LAout Synthesizer) tool from Cadence Design Systems inc.

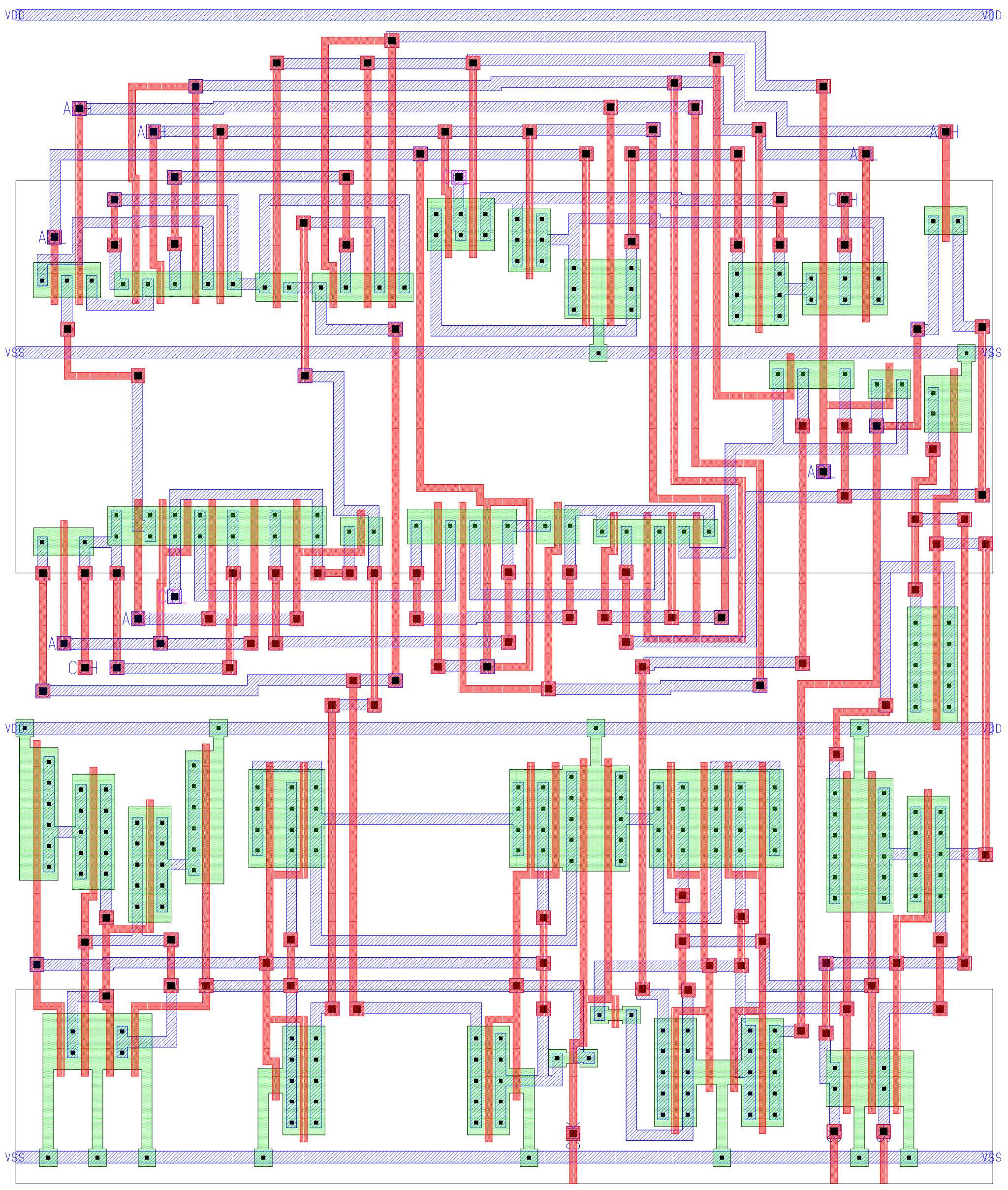
LAS layout: AOI-LFF (area = 2161μm^2)



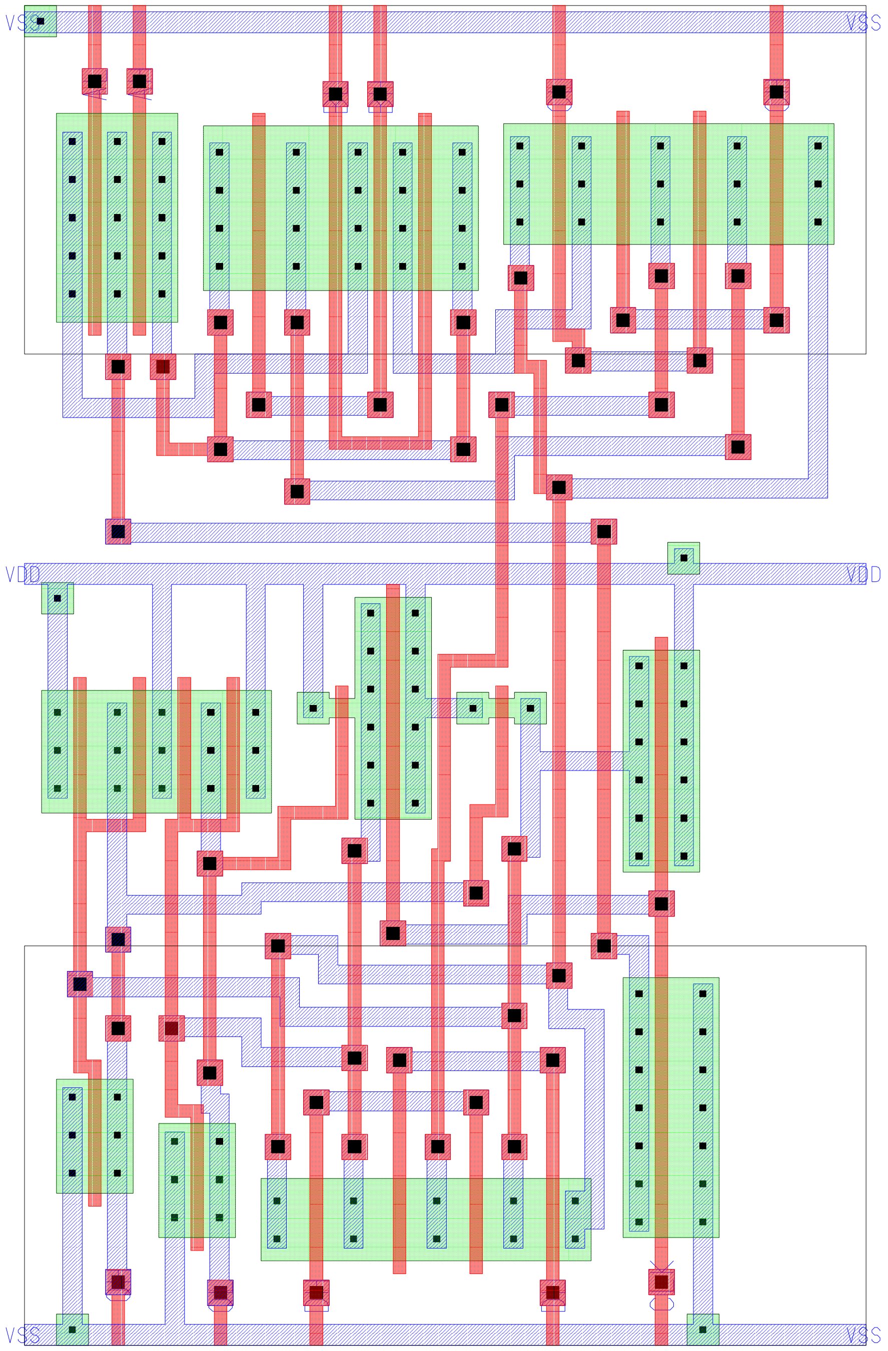
LAS layout: BLAIR-FF (area = 2047μm^2)



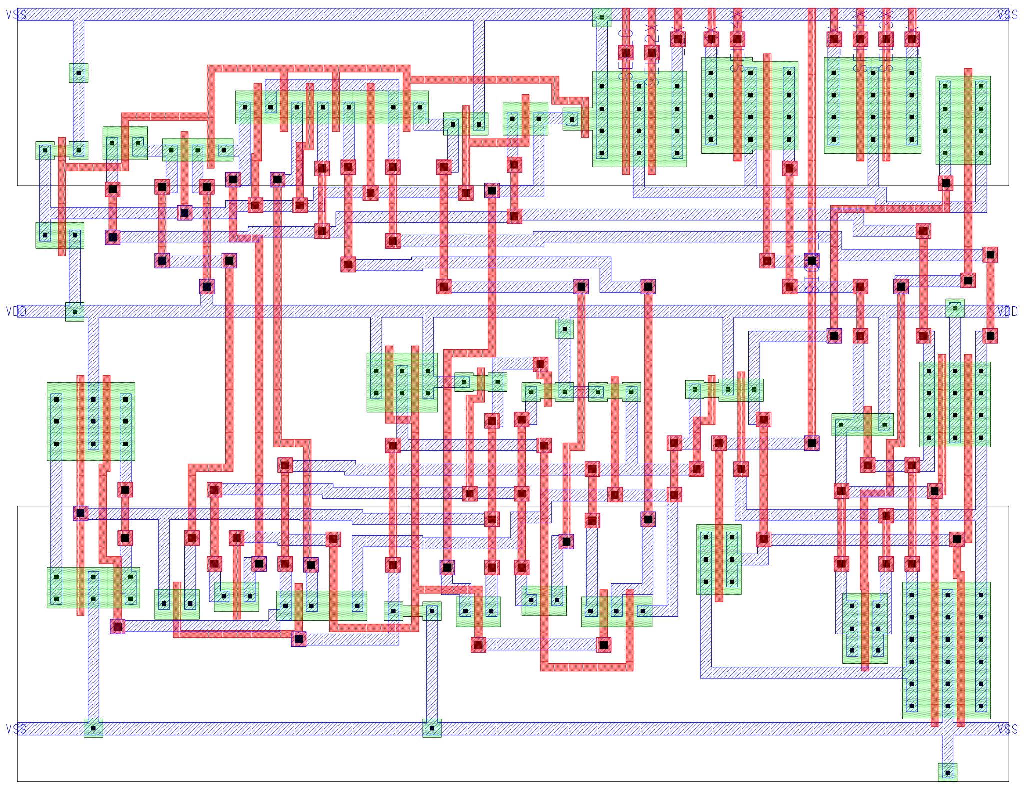
LAS layout: DCSL3-42COMP (area = 8356μm^2)



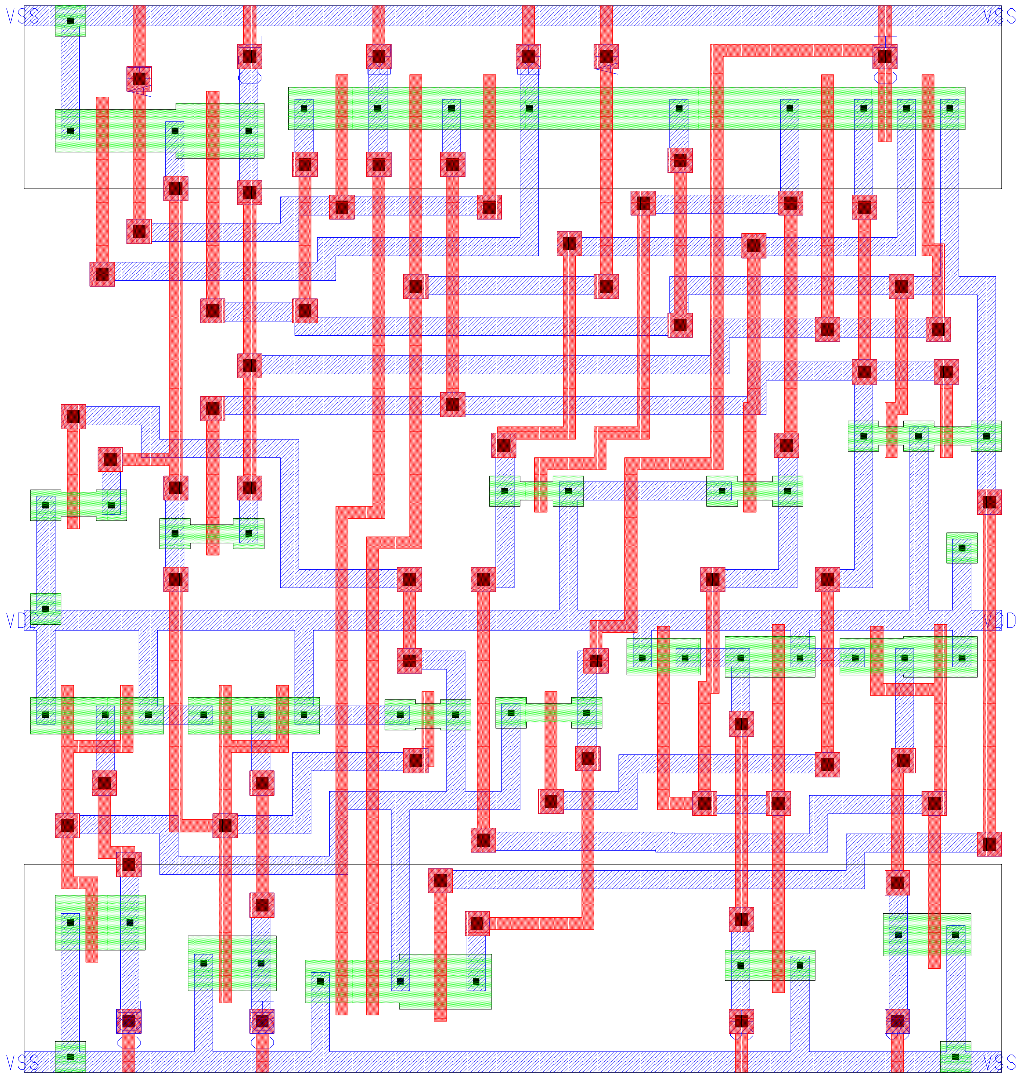
LAS layout: DCVSL-XOR4 (area = 2499μm^2)



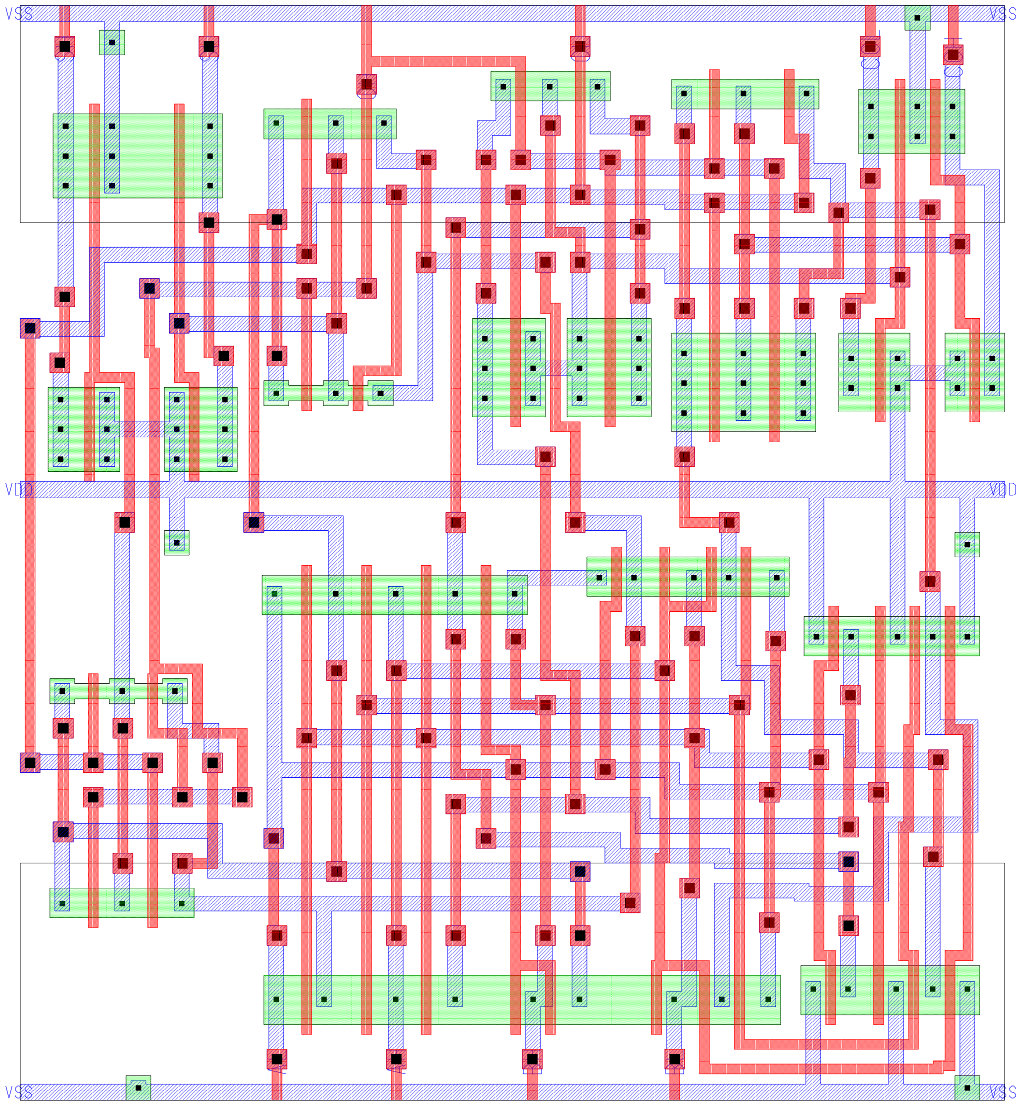
LAS layout: DEC-CSA-MUX (area = 5017μm^2)



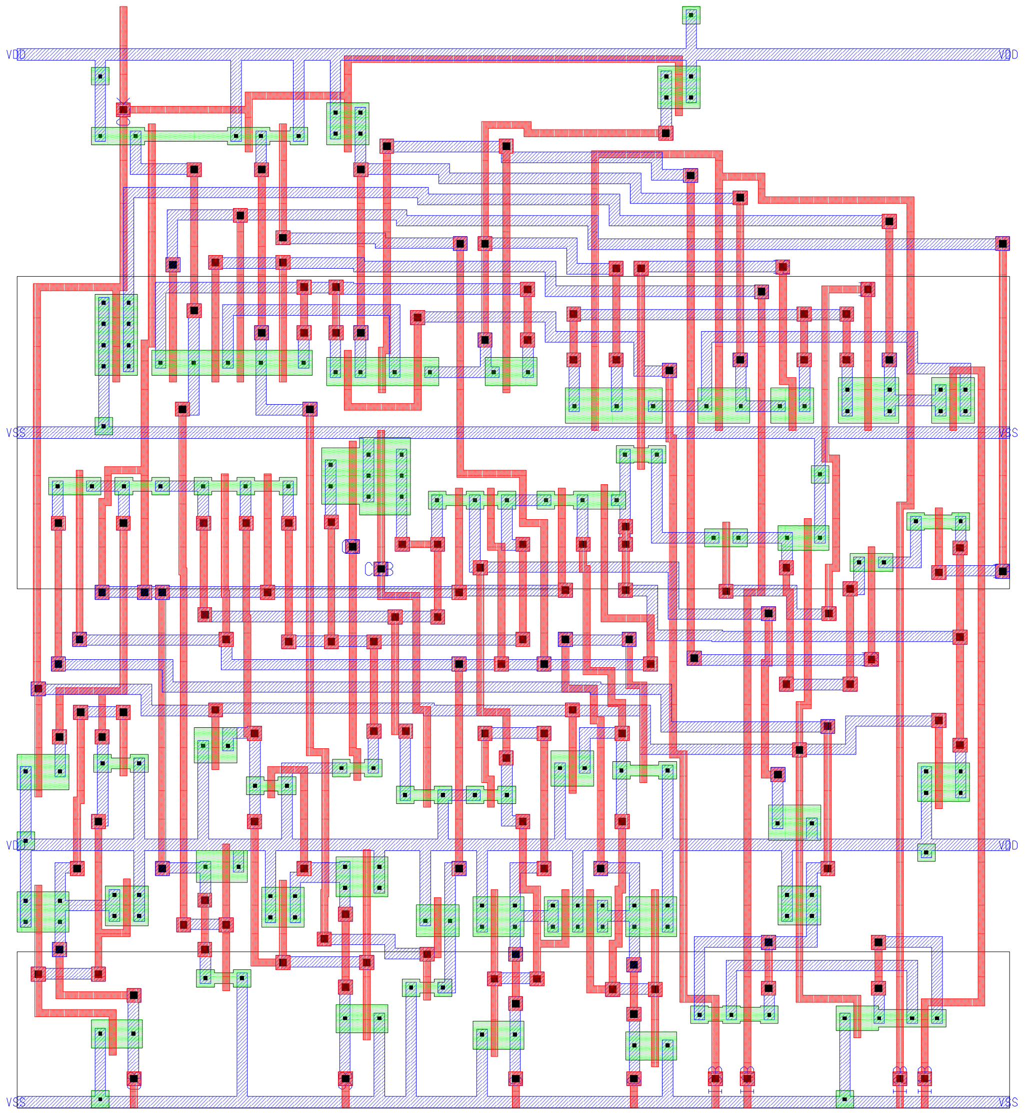
LAS layout: DIFF-FA (area = 2497μm^2)



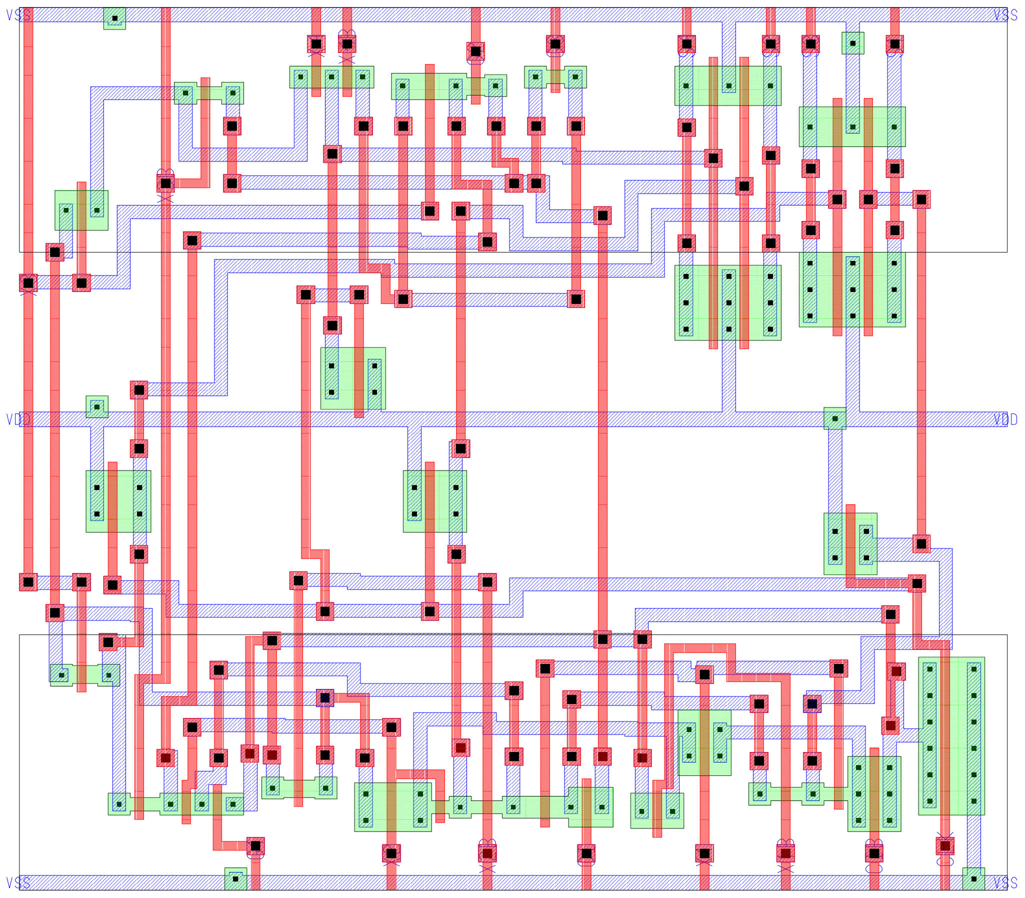
LAS layout: DPL-FA (area = $3950\mu m^2$)



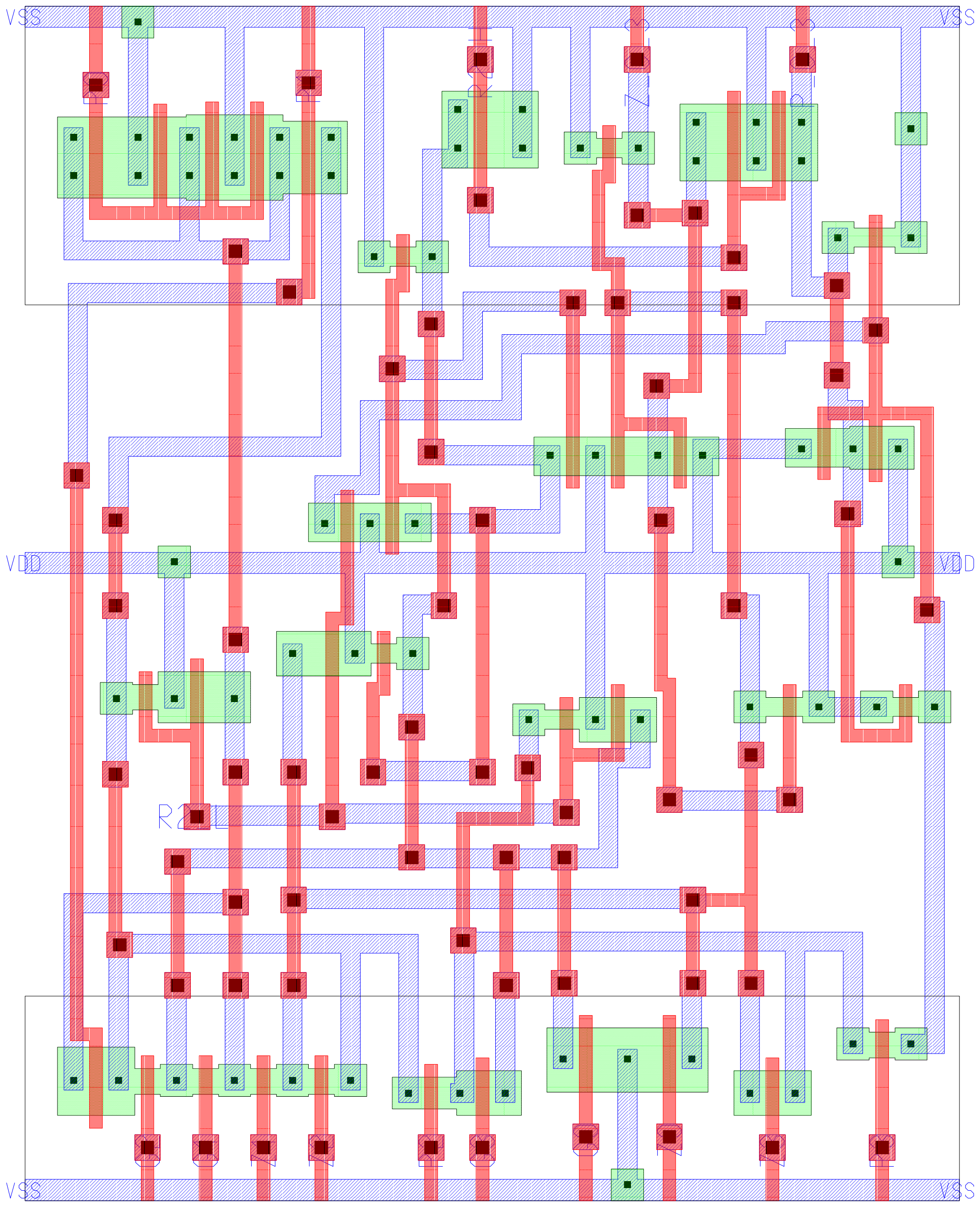
LAS layout: DPTL-42COMP (area = 7879μm^2)



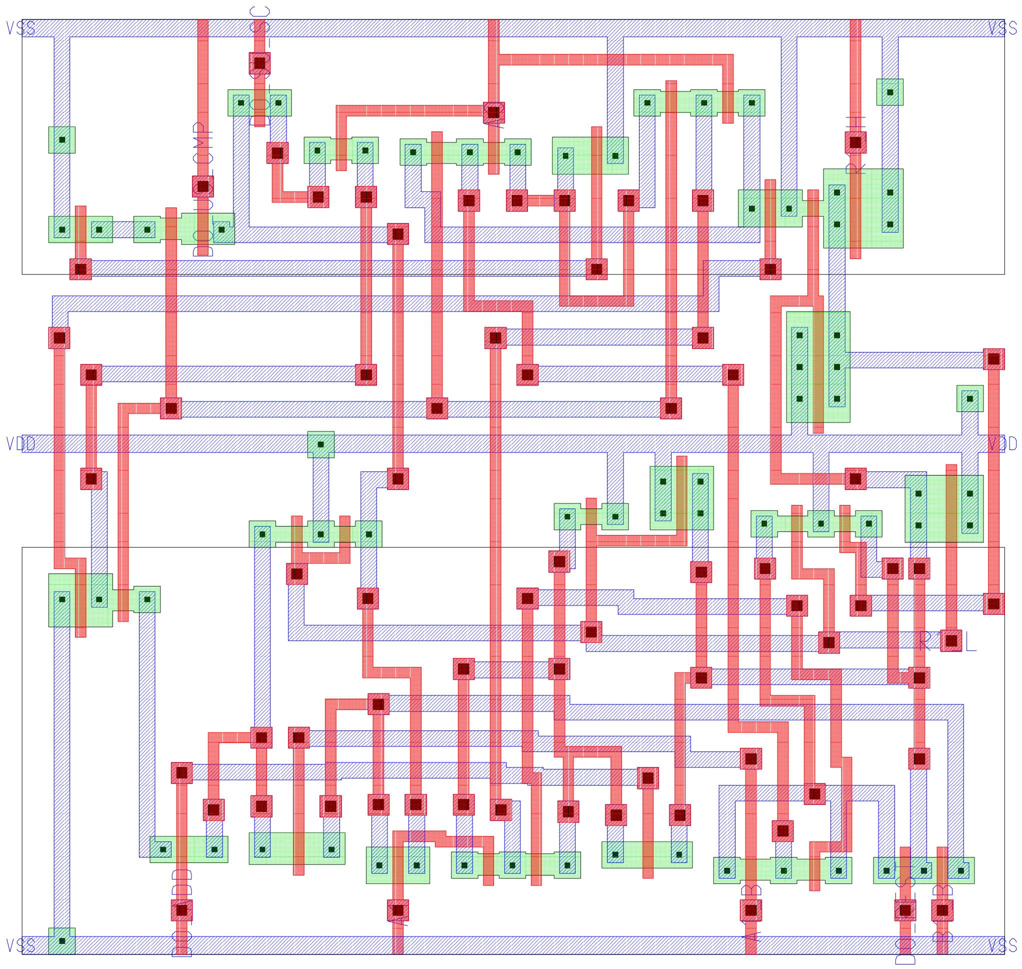
LAS layout: EMODL-CLA (area = 4027μm^2)



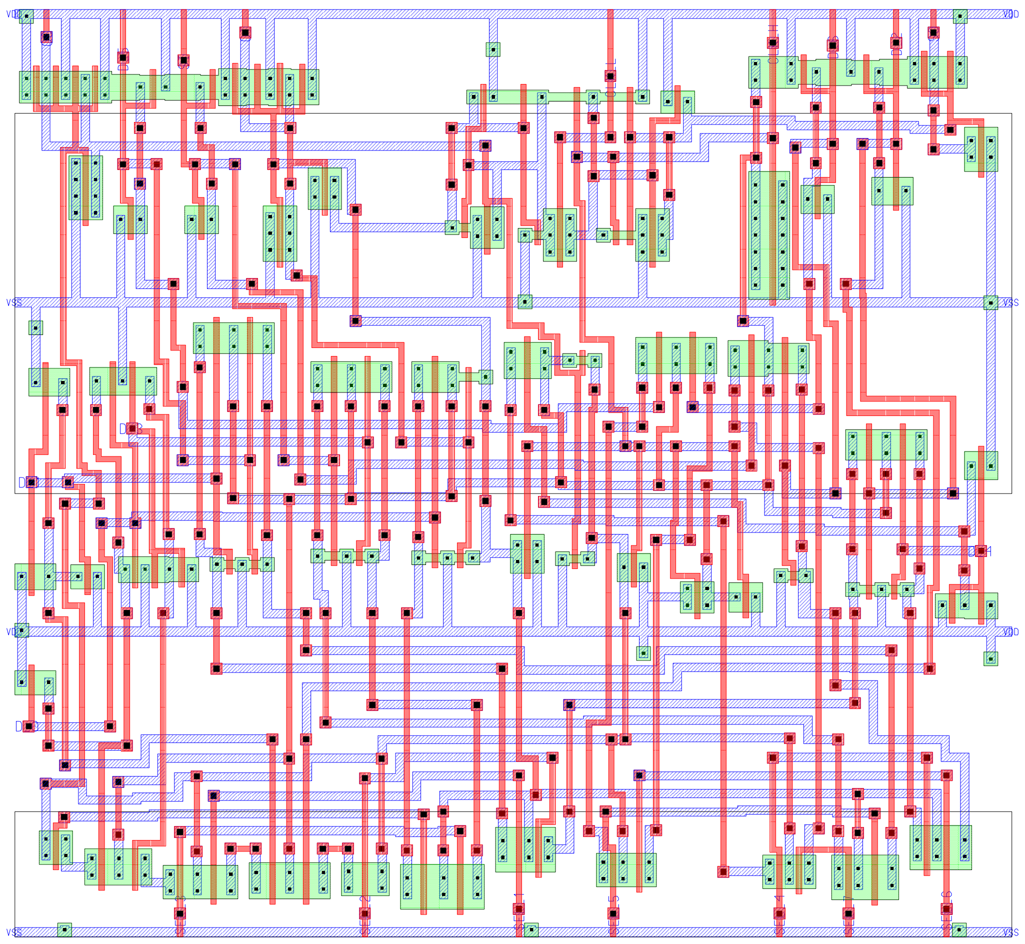
LAS layout: GHZ-CCU-MERGE (area = 3598μm^2)



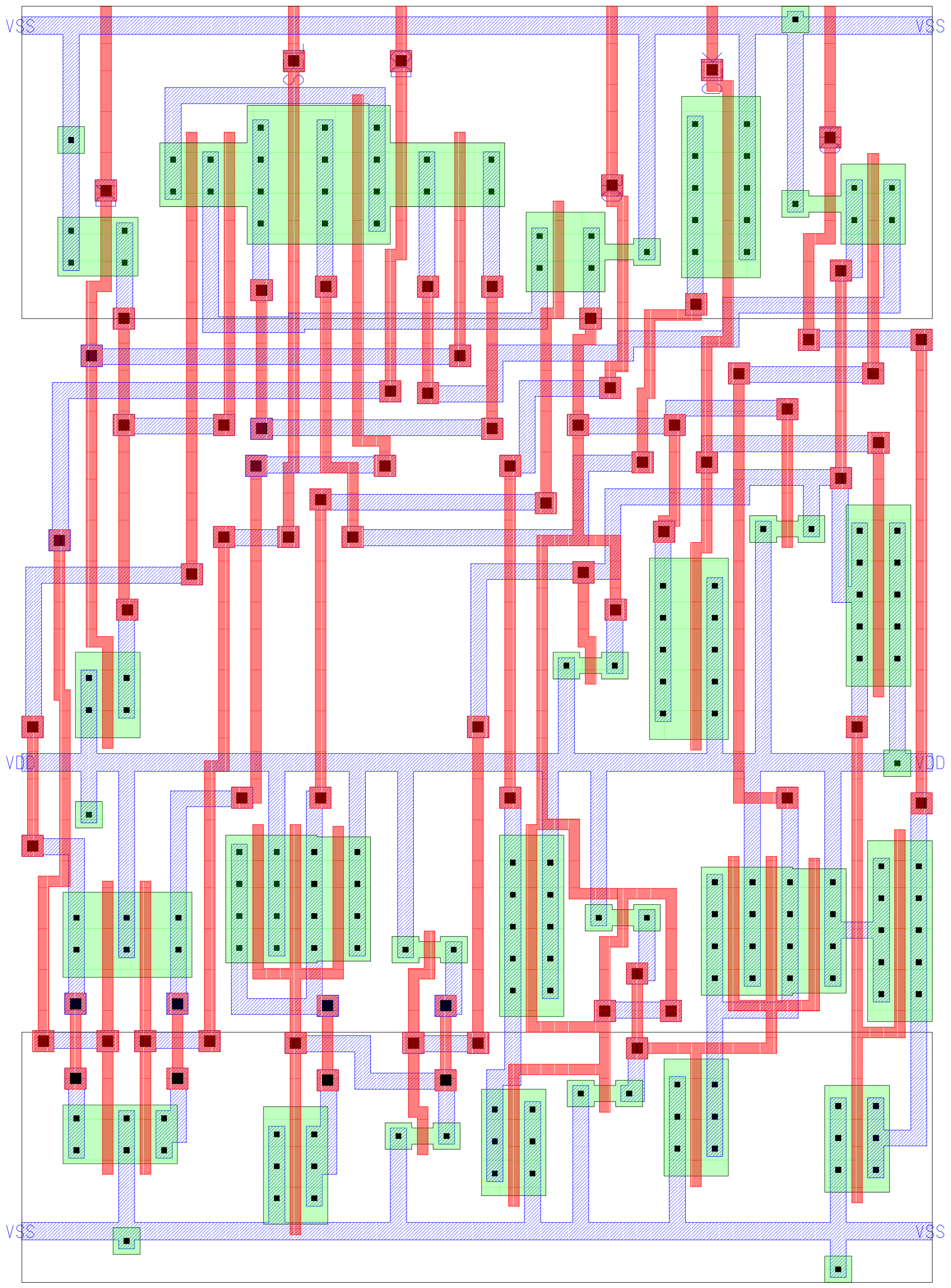
LAS layout: GHZ-CCU-PROP (area = 2939μm^2)



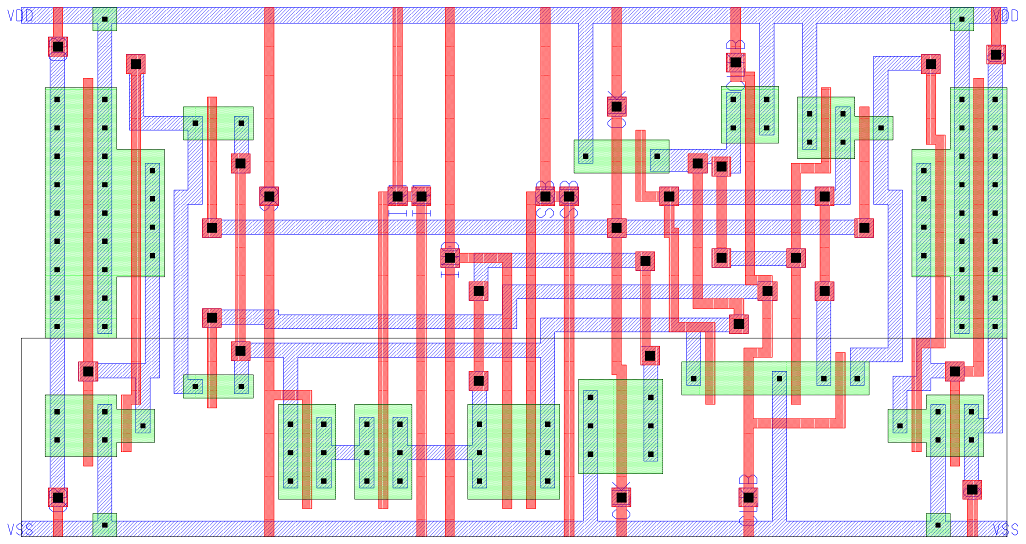
LAS layout: GHZ-MUX8-LA (area = 10680μm^2)



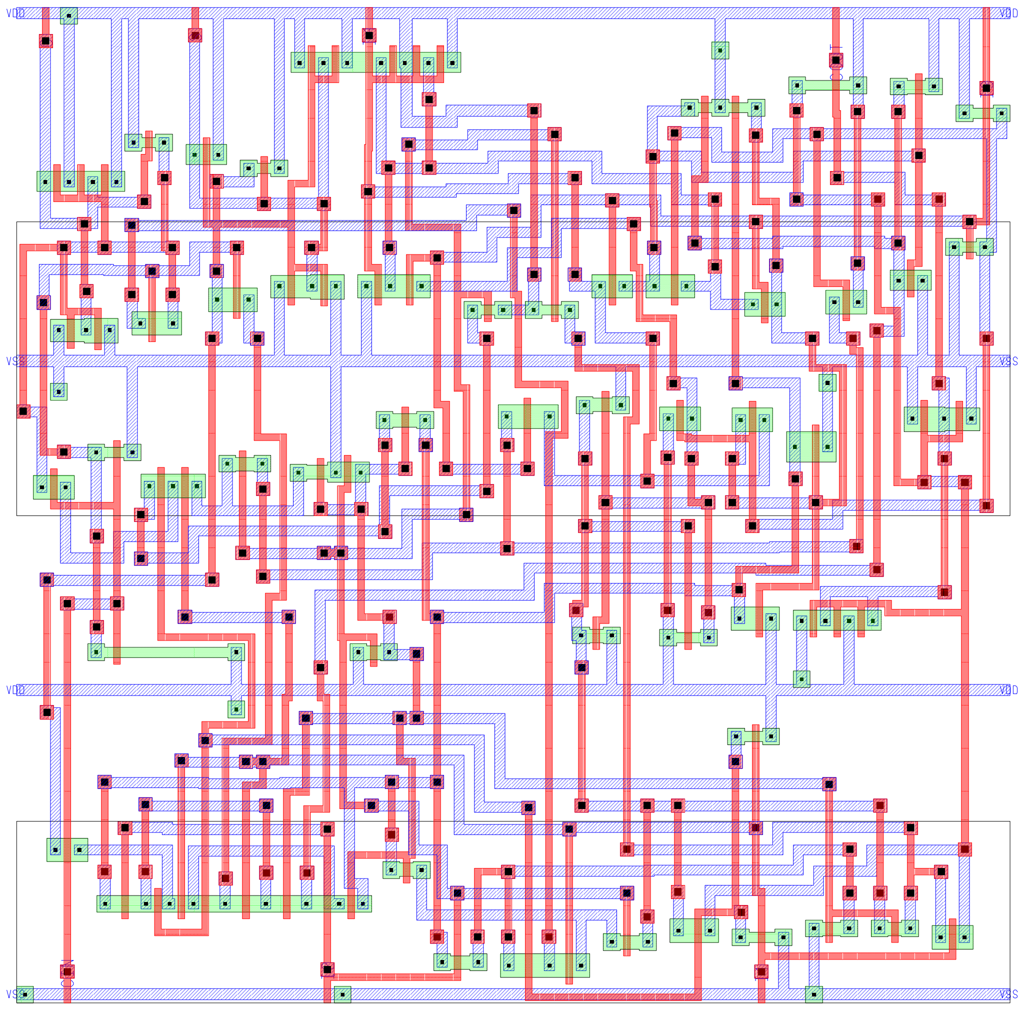
LAS layout: MUX2-SDFF (area = 3632μm^2)



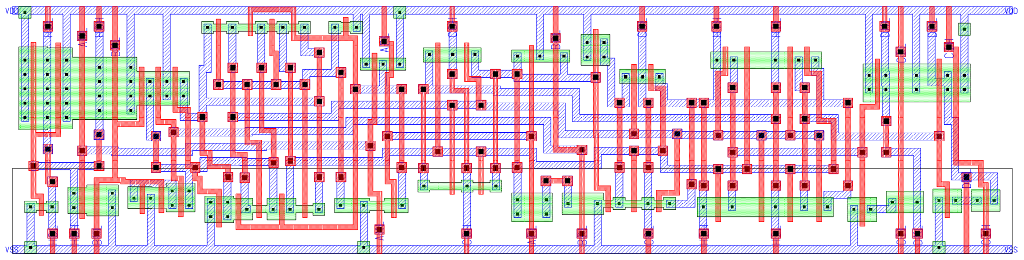
LAS layout: MUXFF (area = 1770μm^2)



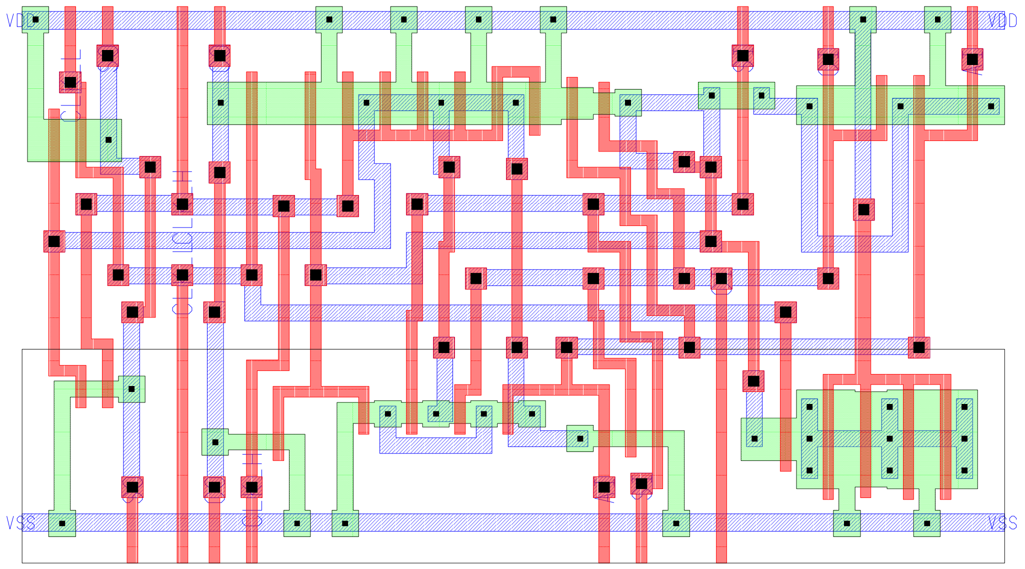
LAS layout: PTL-42COMP (area = 7792μm^2)



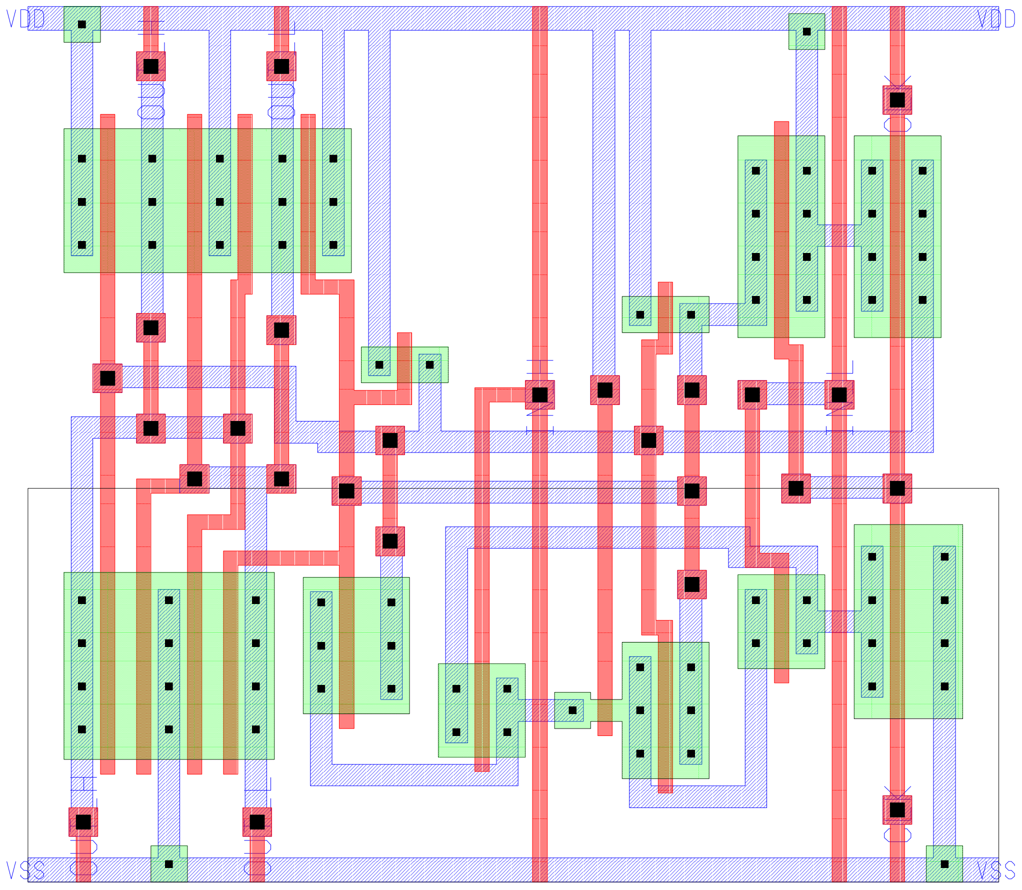
LAS layout: PTL-RBA (area = 3863μm^2)



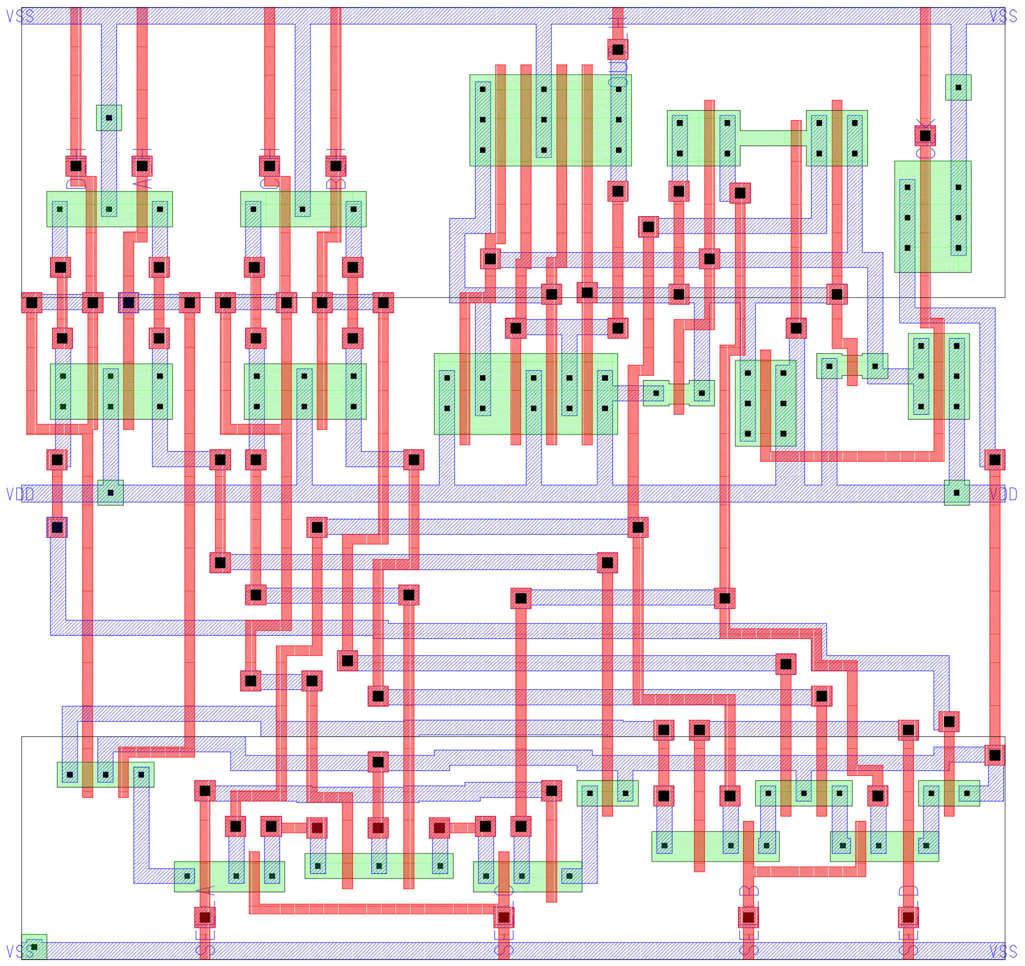
LAS layout: QNP-FA (area = 1736μm^2)



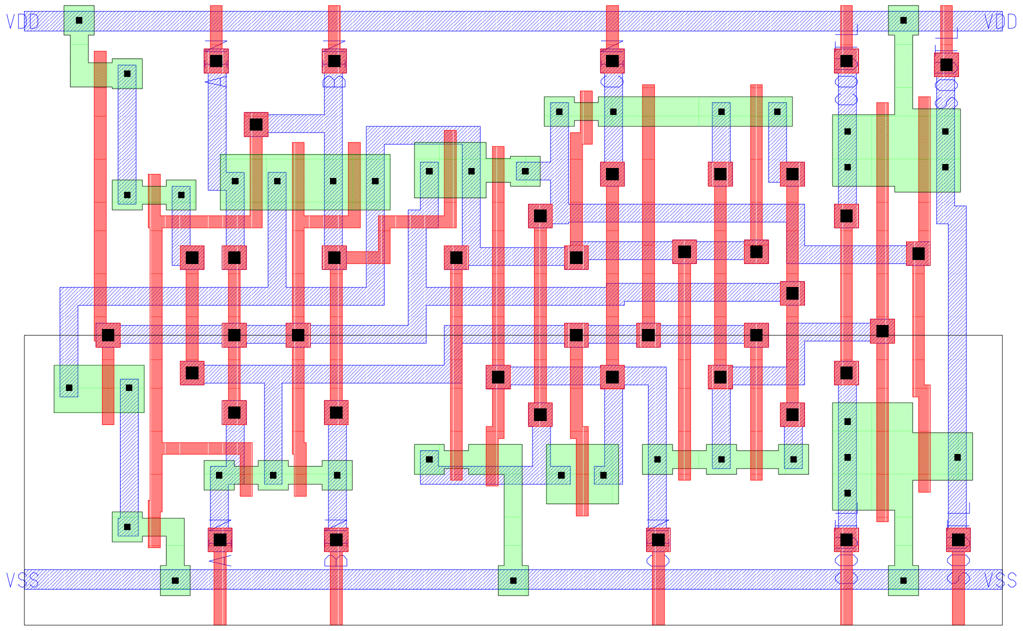
LAS layout: SA-FF (area = 1465μm^2)



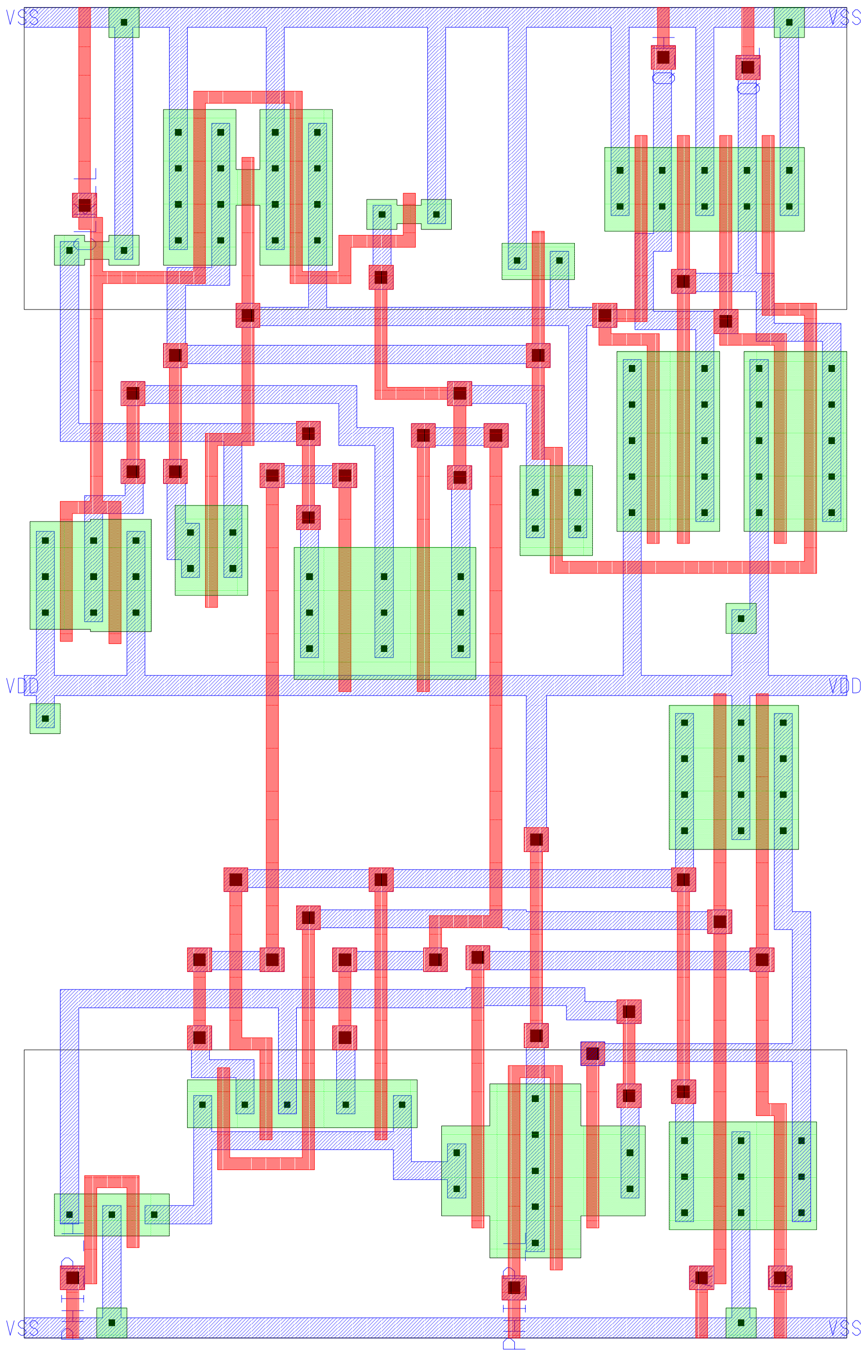
LAS layout: SA-MUX-FF (area = 3266μm^2)



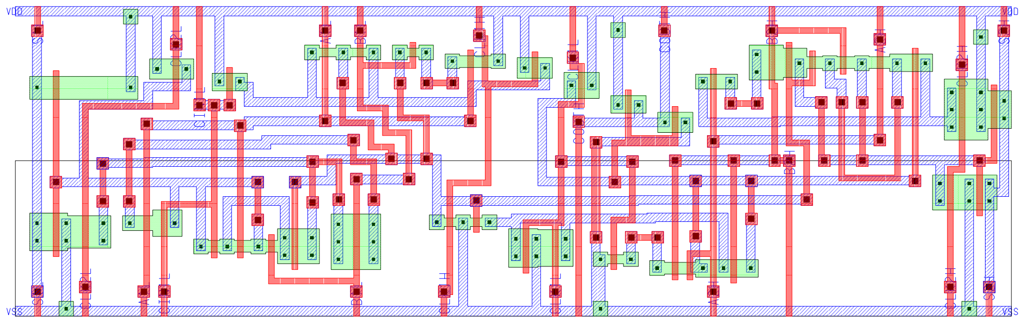
LAS layout: T17-FA (area = 1623μm^2)



LAS layout: XOR-FF (area = 2713μm^2)



LAS layout: ZIP-FA2 (area = 3253μm^2)

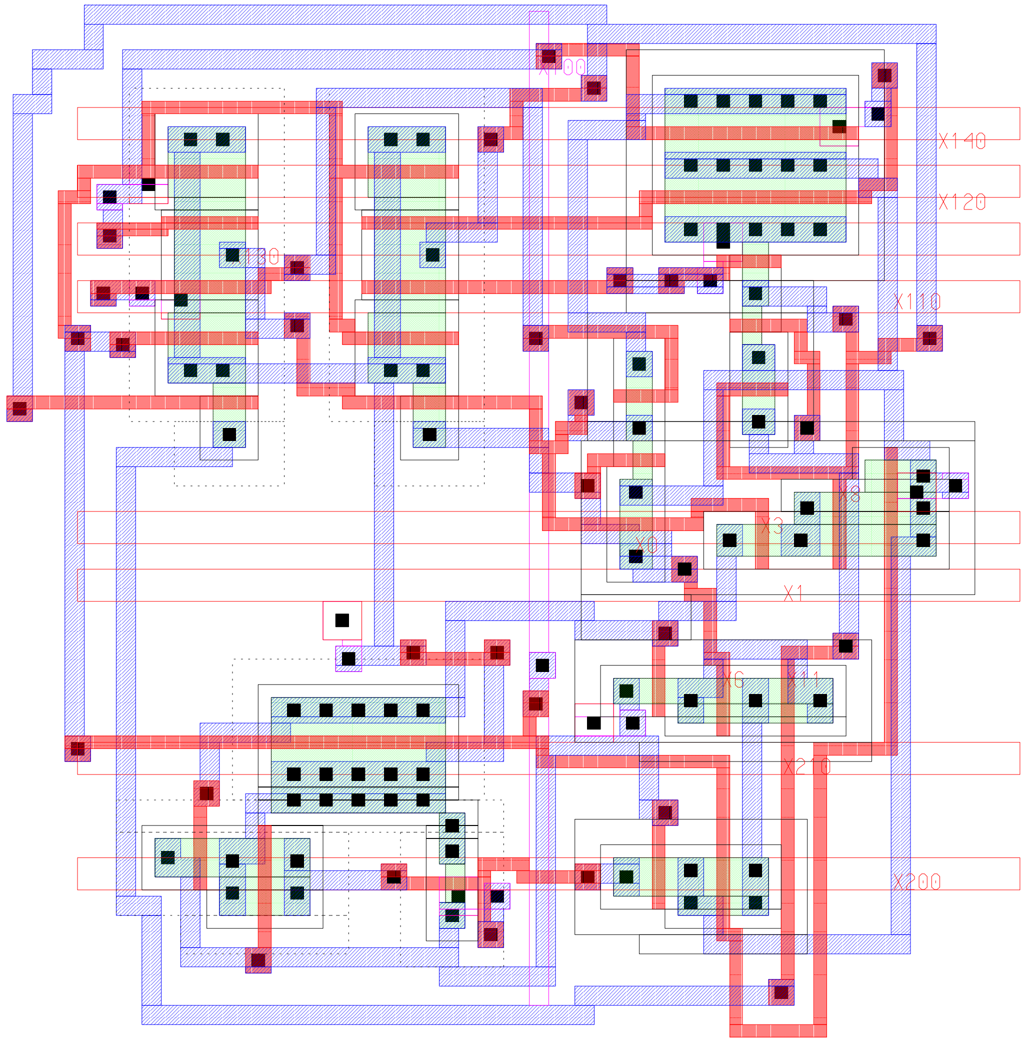


APPENDIX D

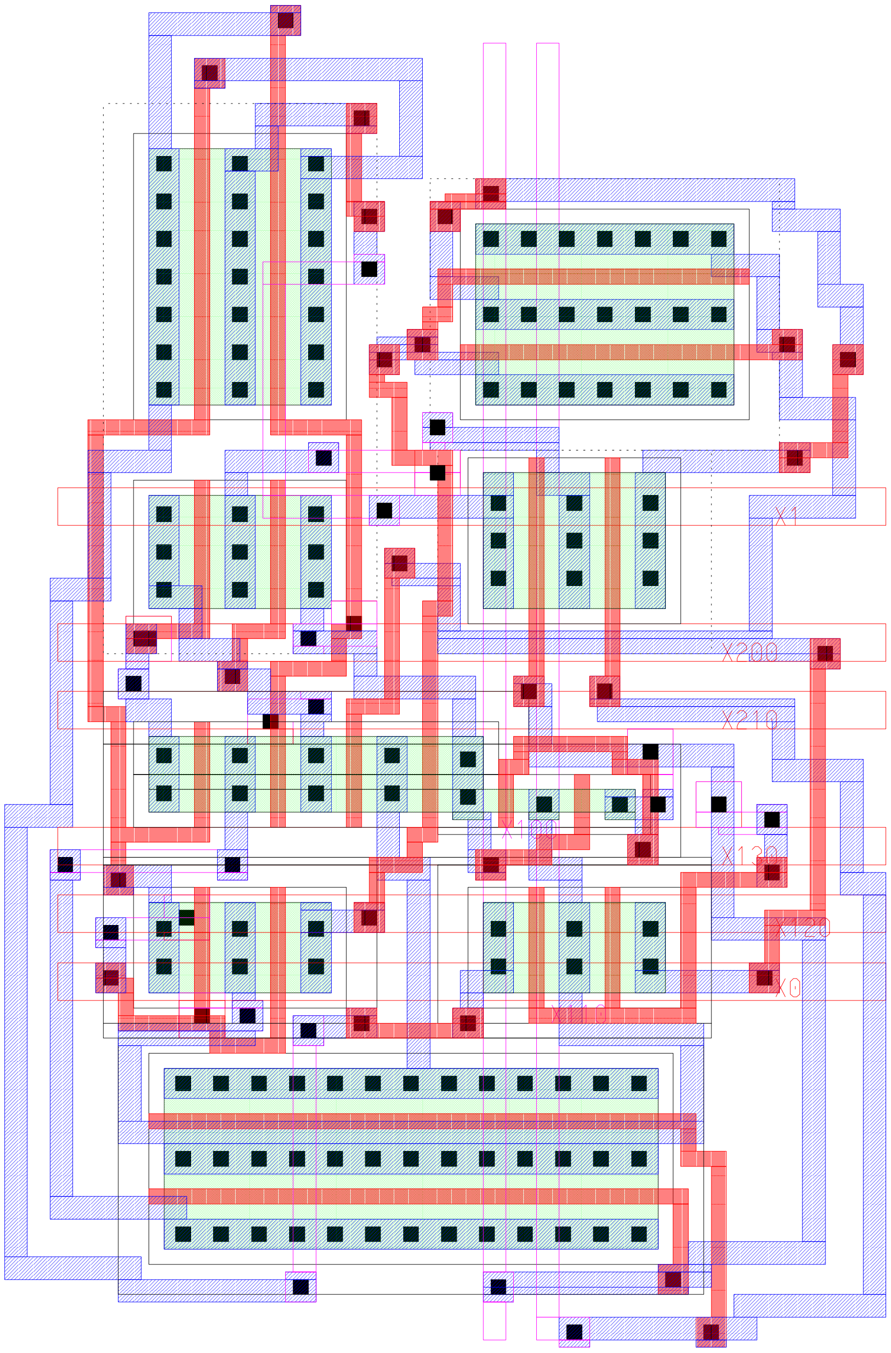
TEMPO Layouts (Pre-Compaction)

This appendix contains the benchmark layouts produced by TEMPO before the compaction step.

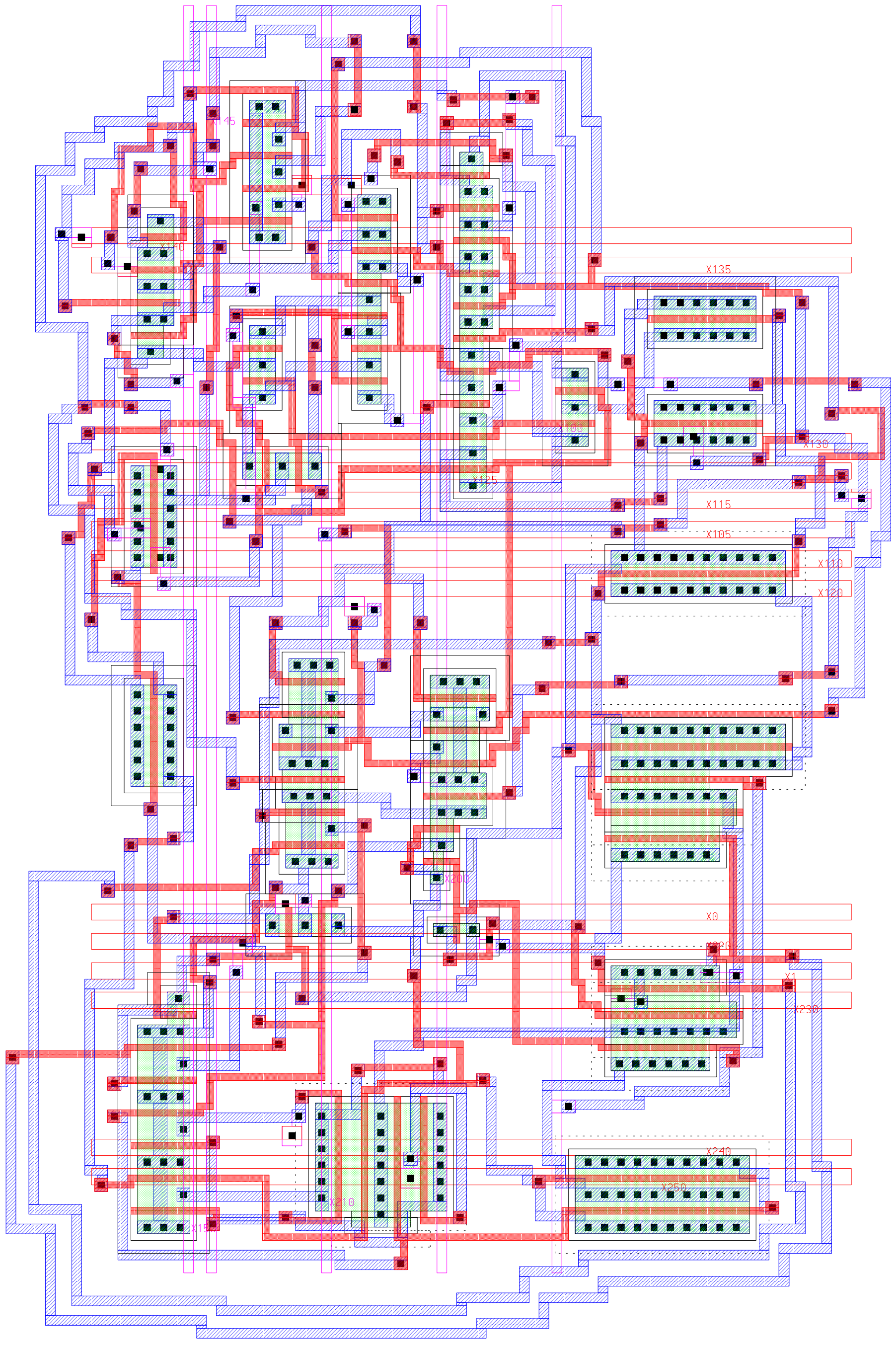
TEMPO layout: AOI-LFF (area = 2275μm^2)



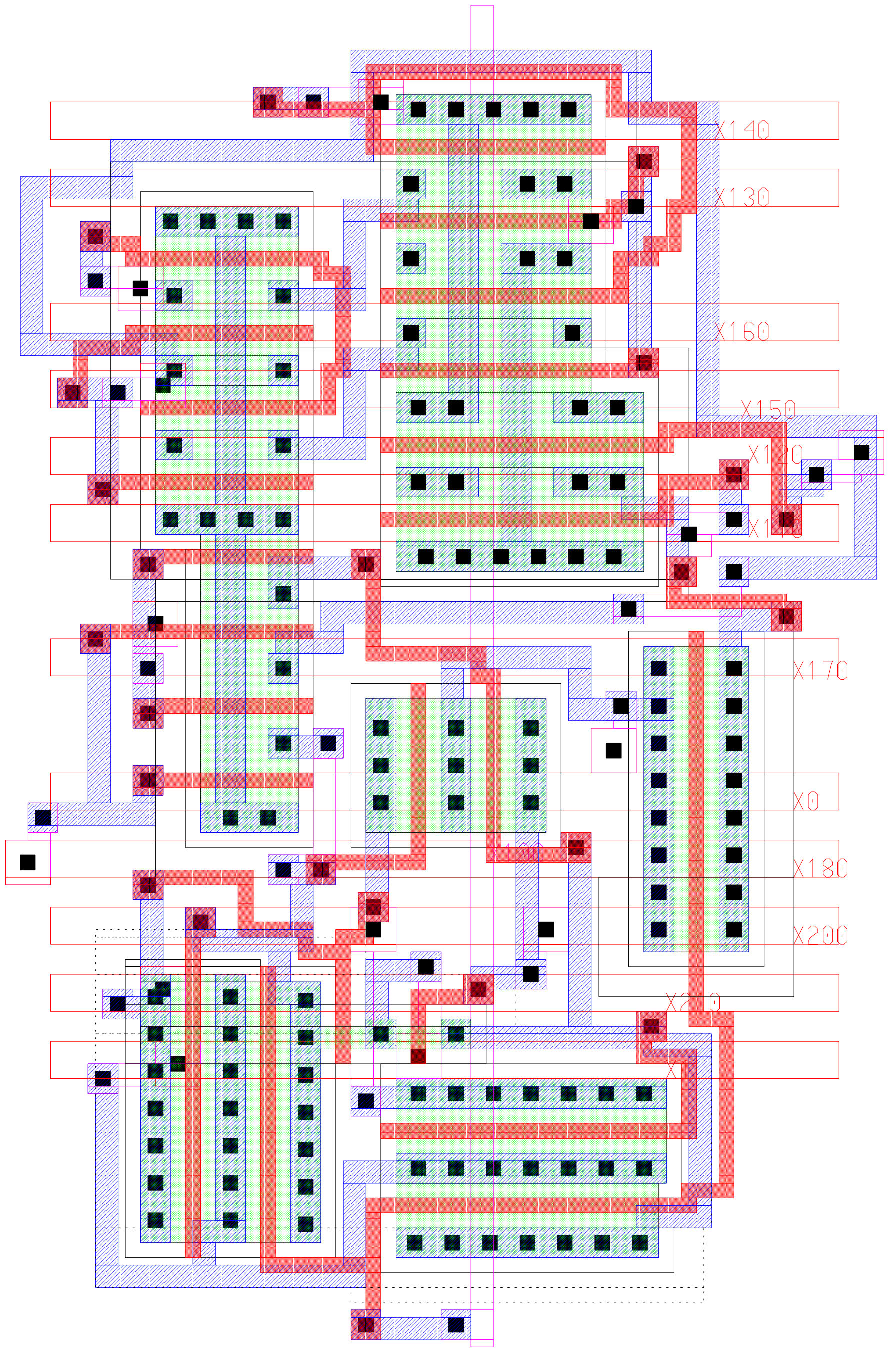
TEMPO layout: BLAIR-FF (area = 1858μm^2)



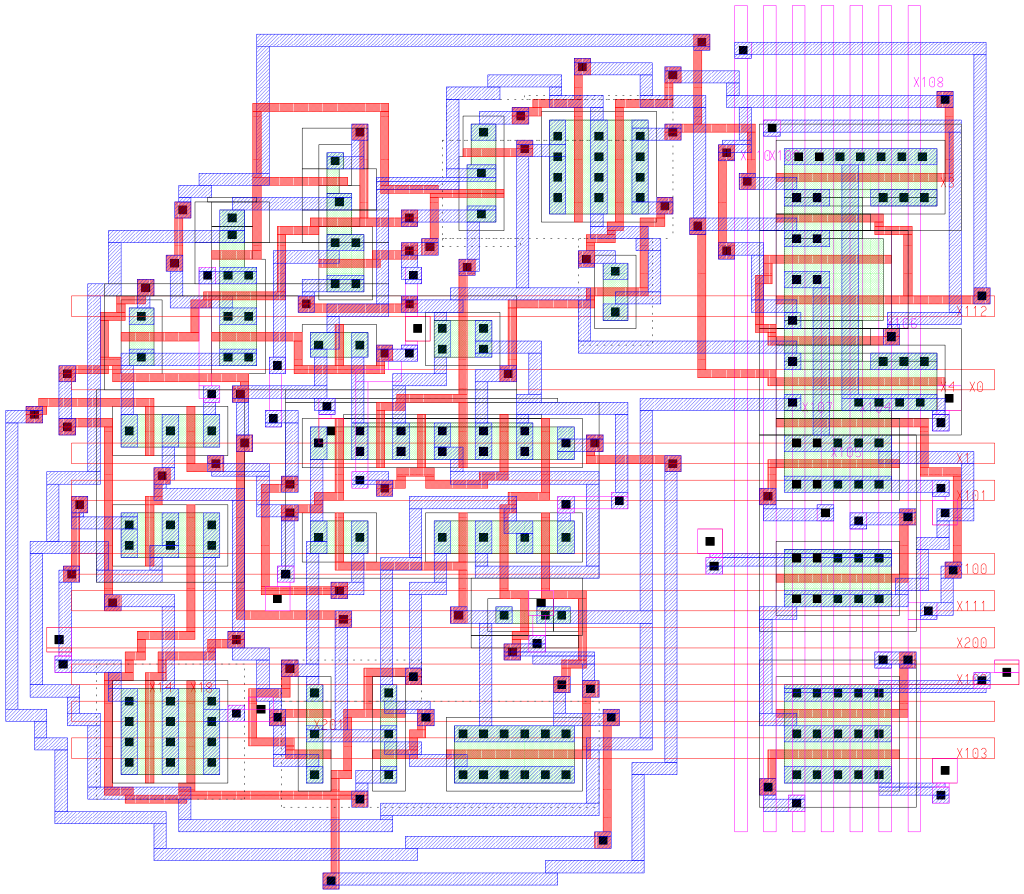
TEMPO layout: DCSL3-42COMP (area = 9878μm^2)



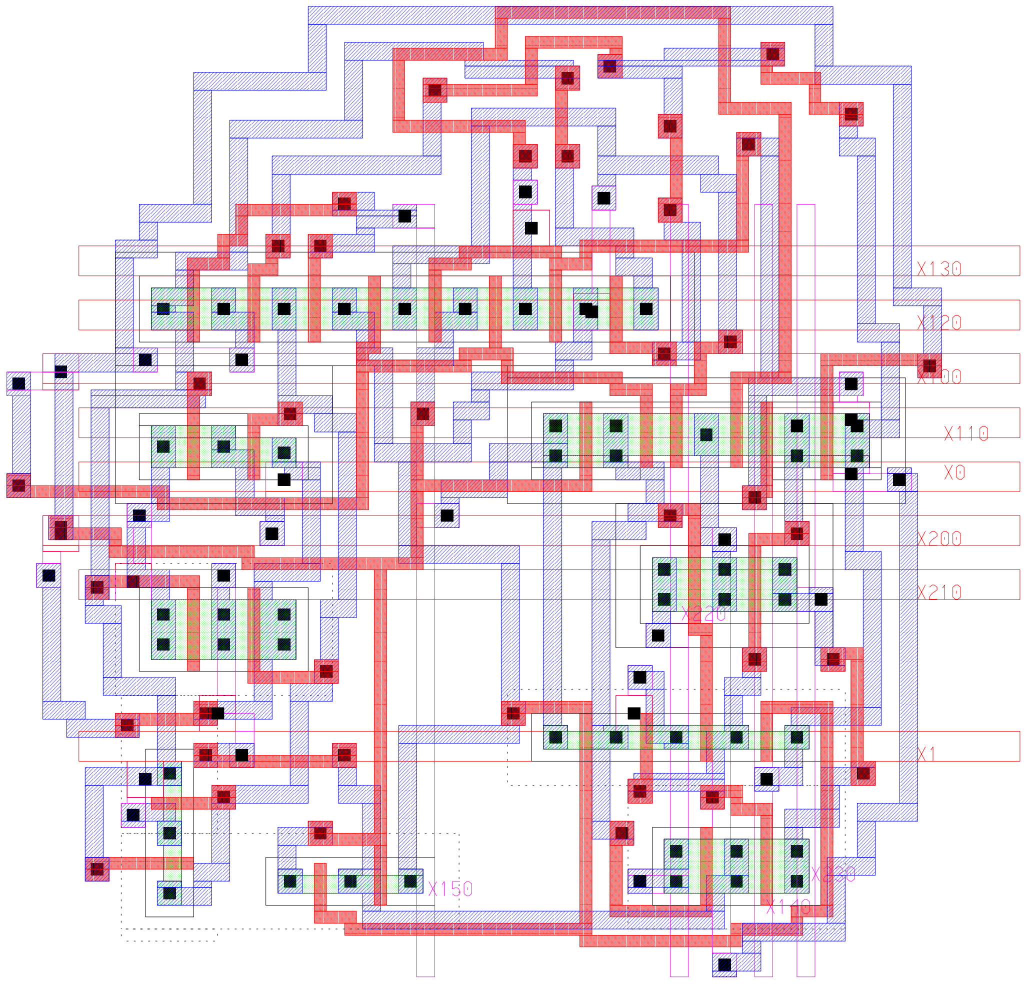
TEMPO layout: DCVSL-XOR4 (area = 1895μm^2)



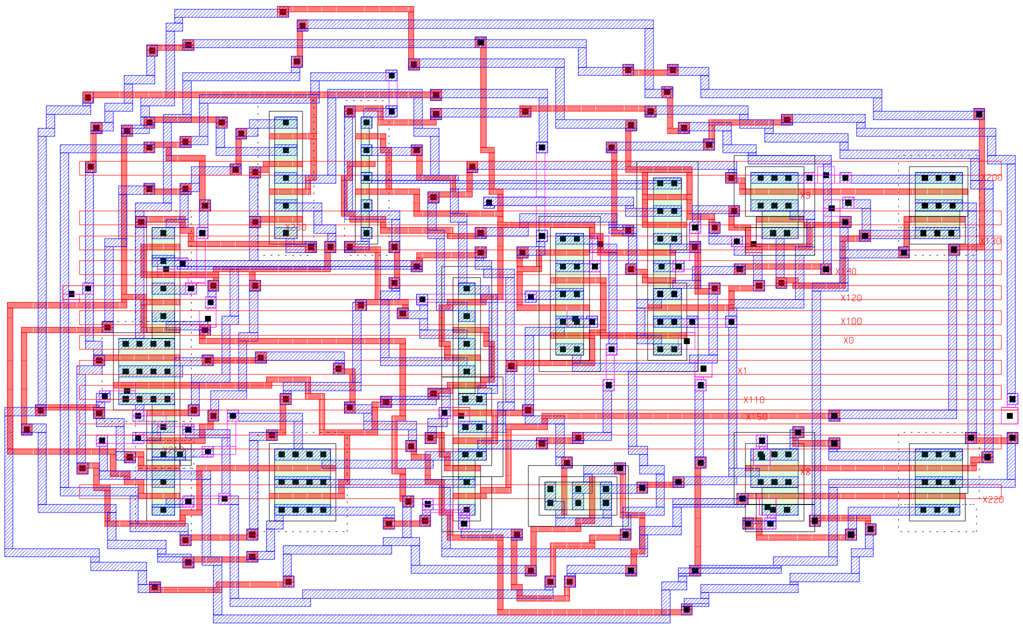
TEMPO layout: DEC-CSA-MUX (area = 4782μm^2)



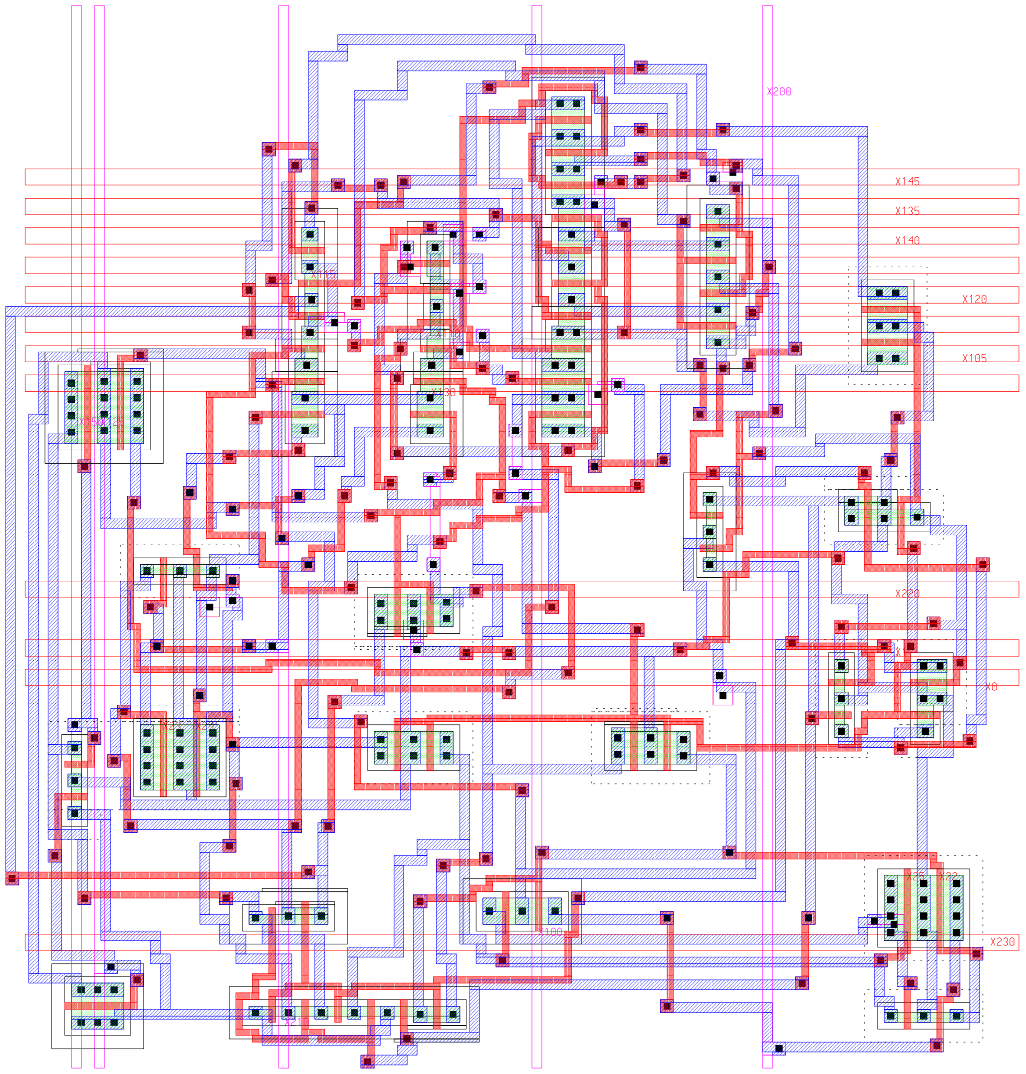
TEMPO layout: DIFF-FA (area = 2449μm^2)



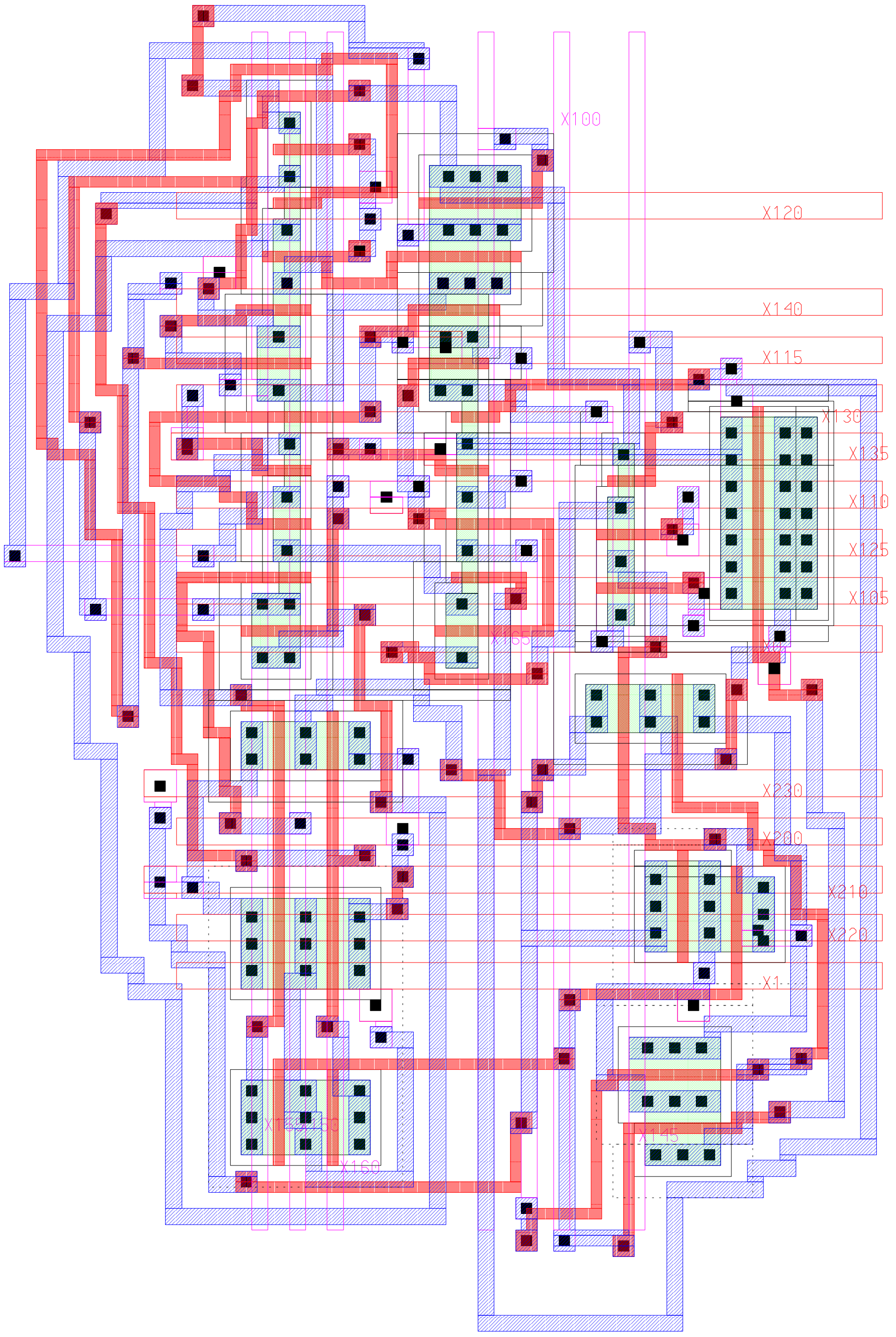
TEMPO layout: DPL-FA (area = 7305μm^2)



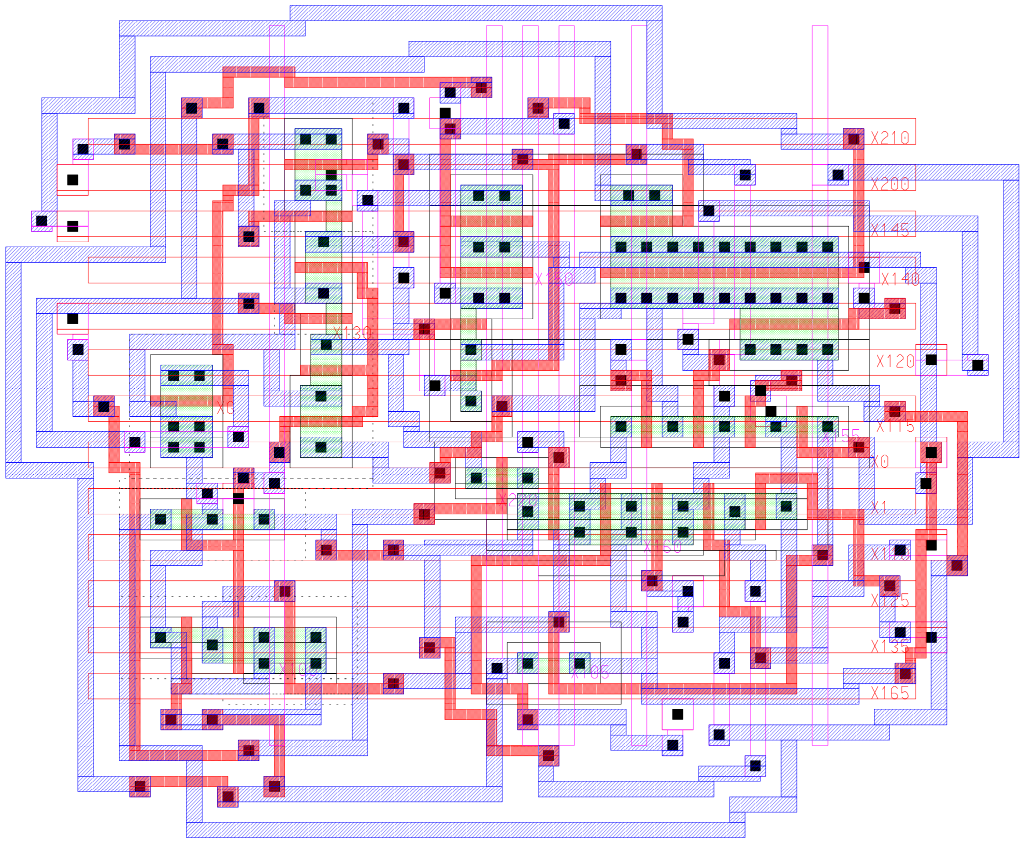
TEMPO layout: DPTL-42COMP (area = 9009μm^2)



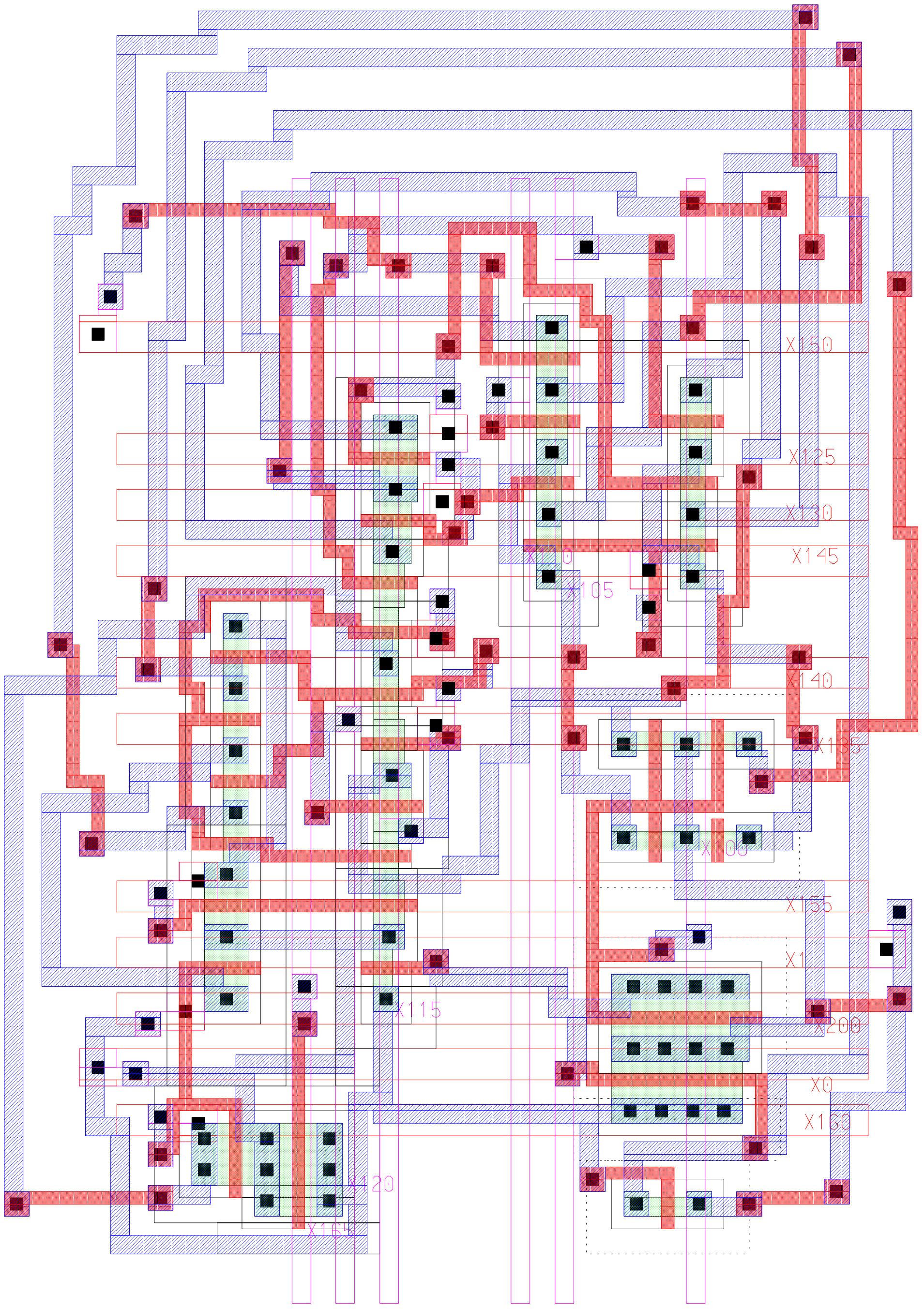
TEMPO layout: EMODL-CLA (area = 3638μm^2)



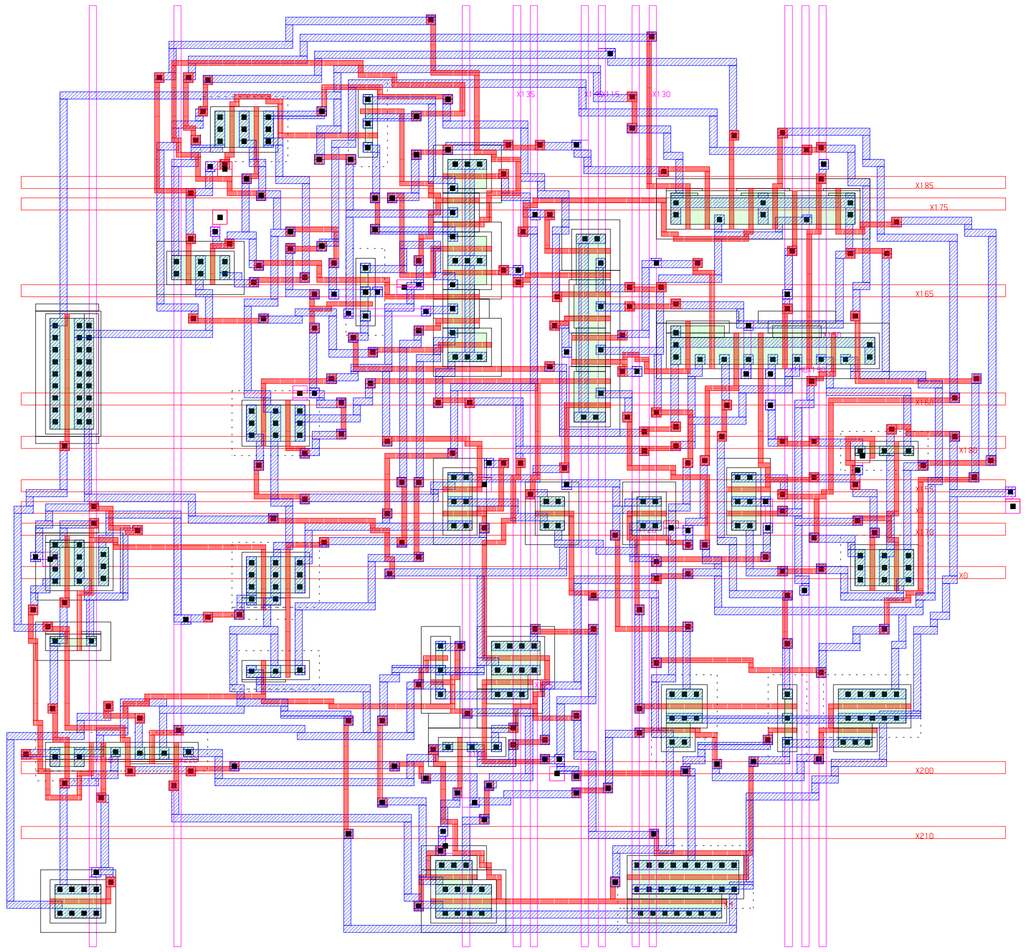
TEMPO layout: GHZ-CCU-MERGE (area = 2858μm^2)



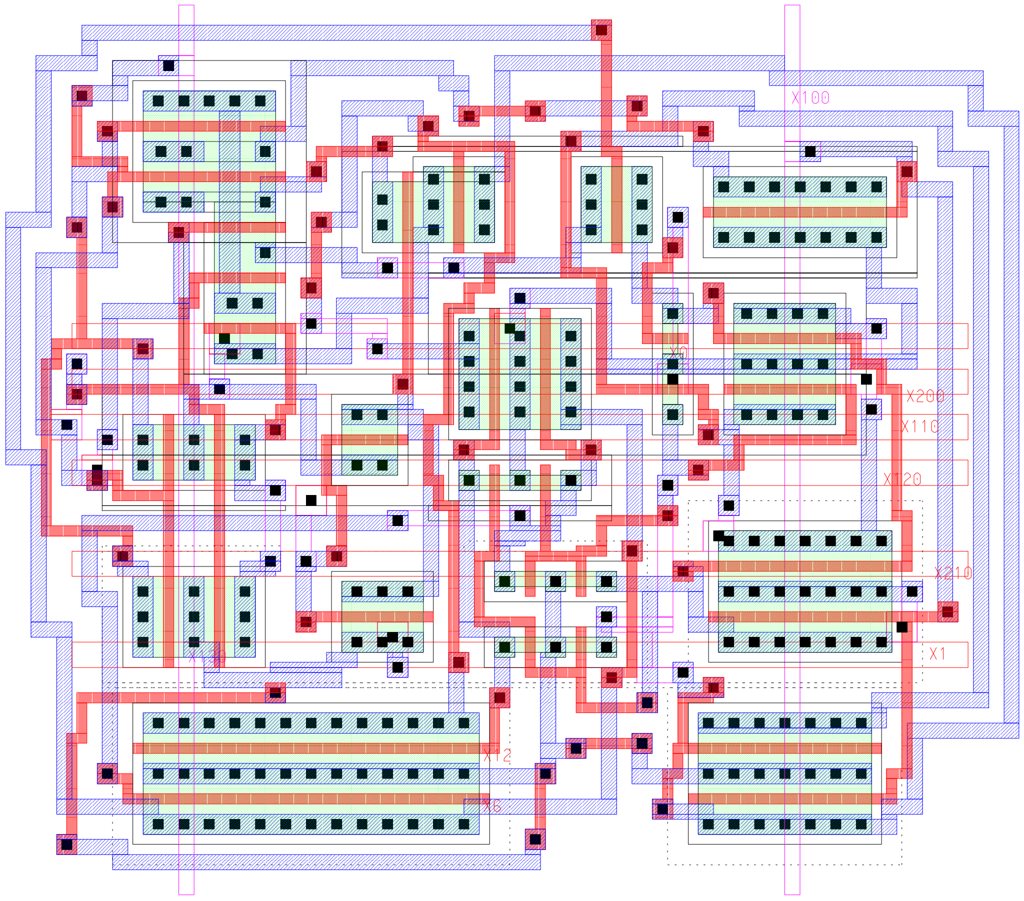
TEMPO layout: GHZ-CCU-PROP (area = 2746μm^2)



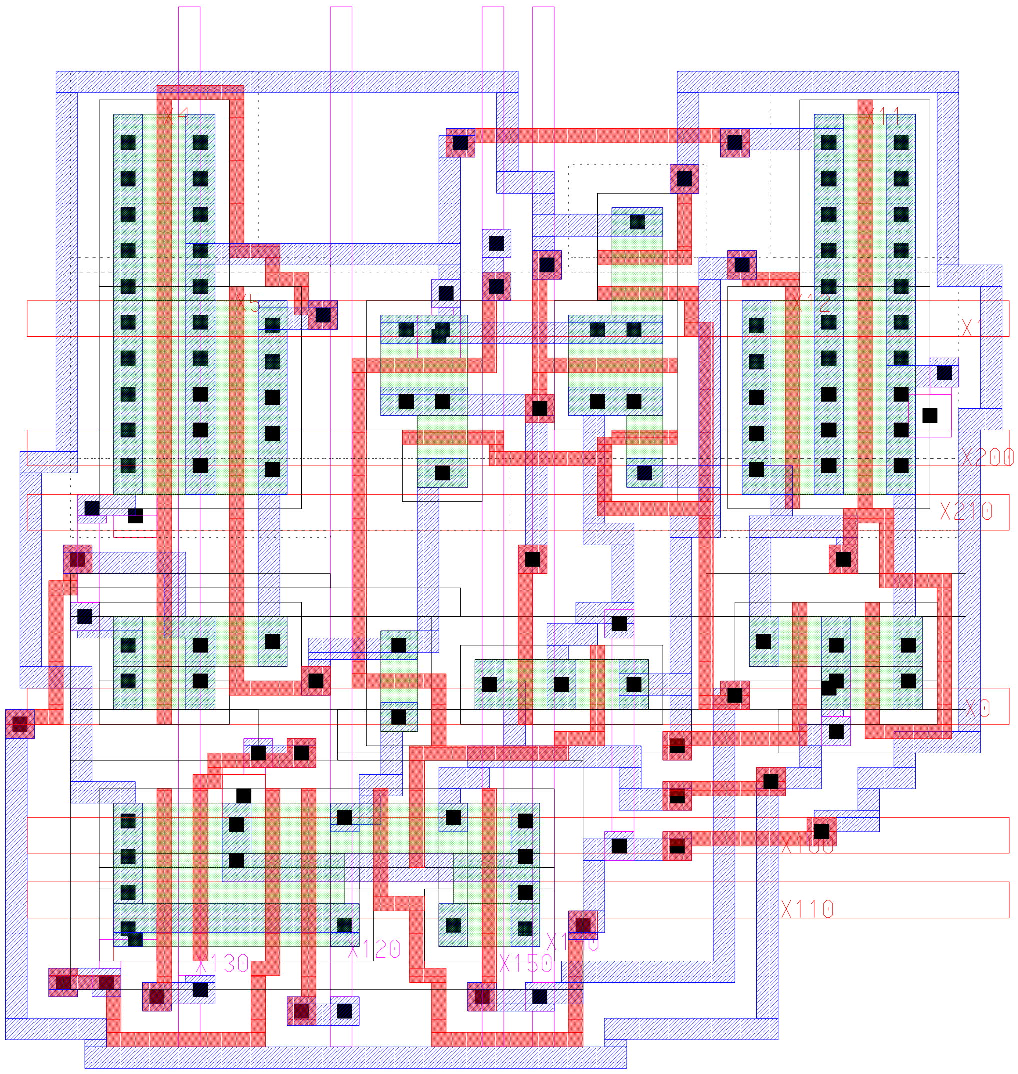
TEMPO layout: GHZ-MUX8-LA (area = 14709μm^2)



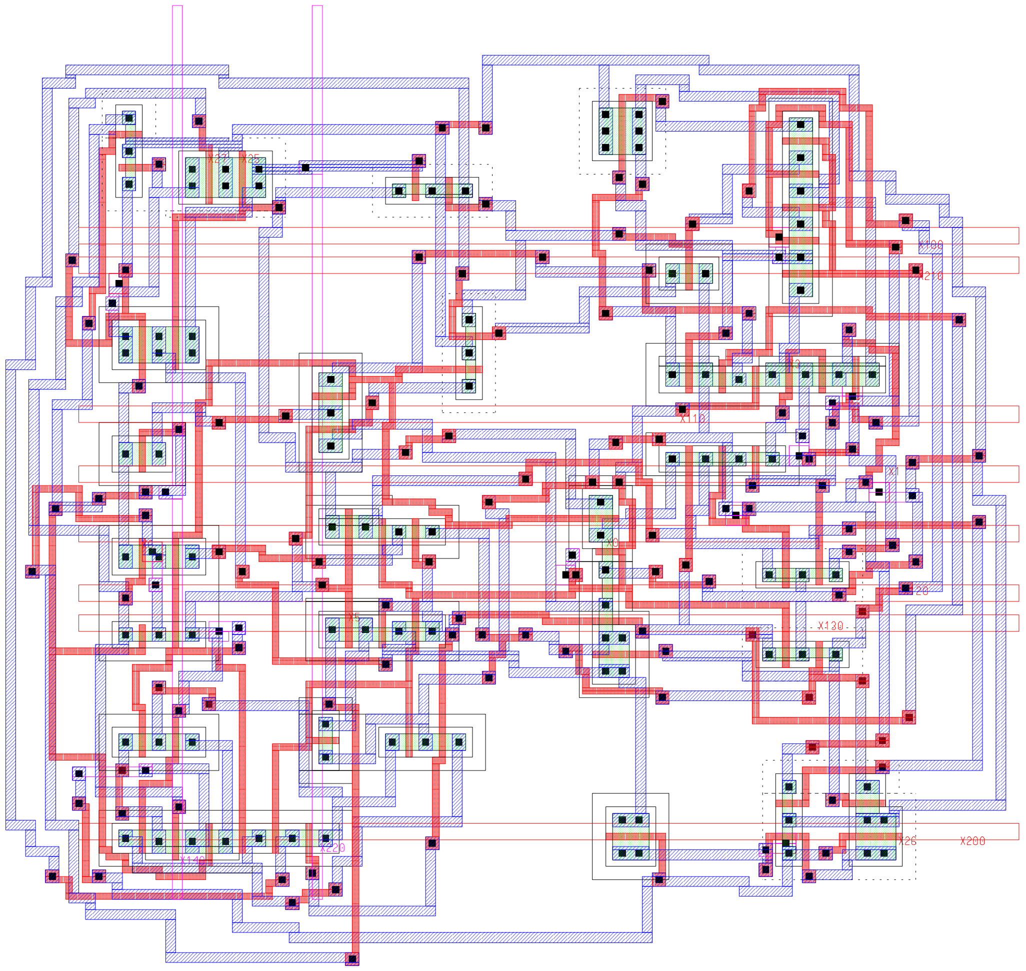
TEMPO layout: MUX2-SDFF (area = 3152μm^2)



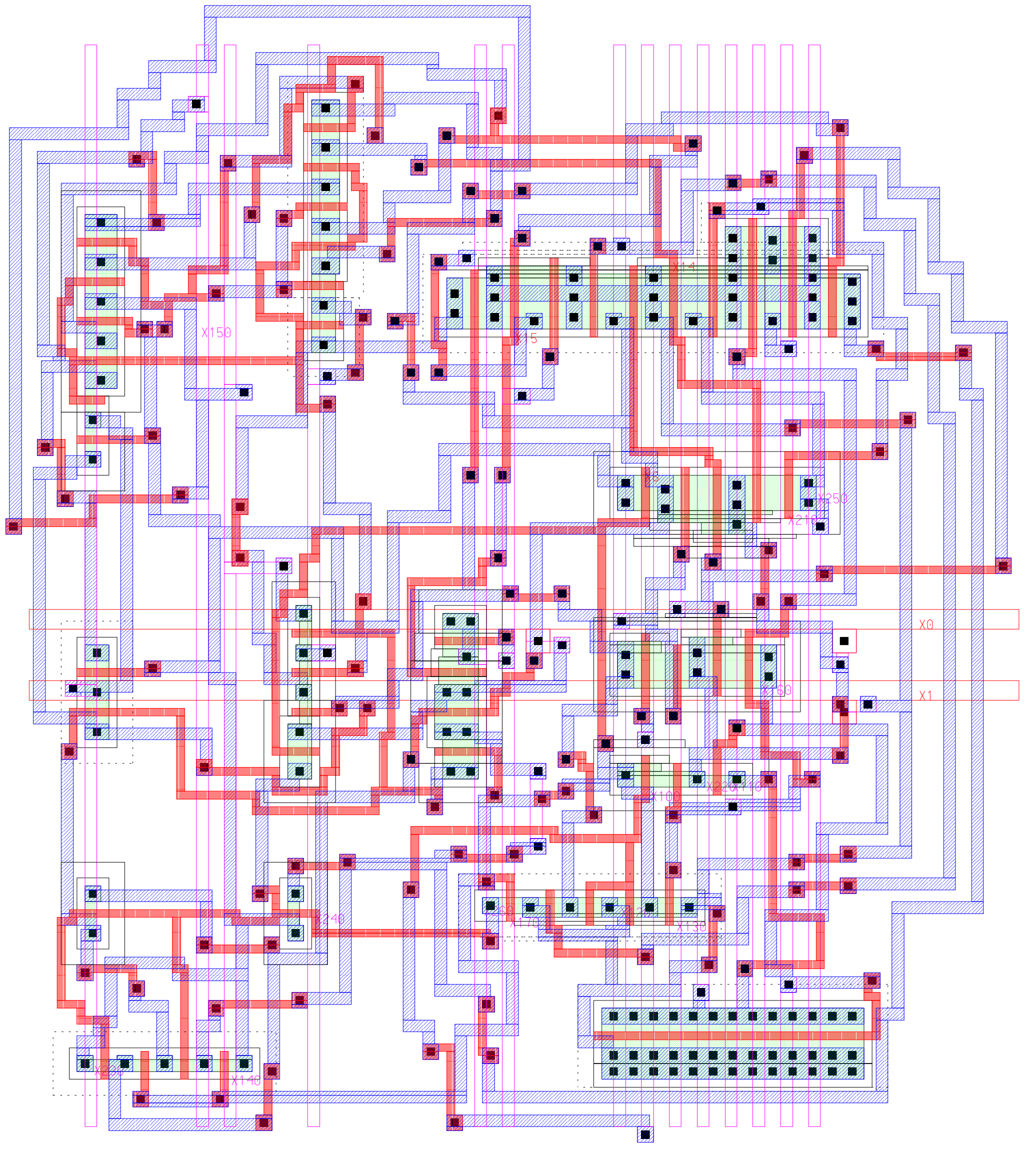
TEMPO layout: MUXFF (area = 1851μm^2)



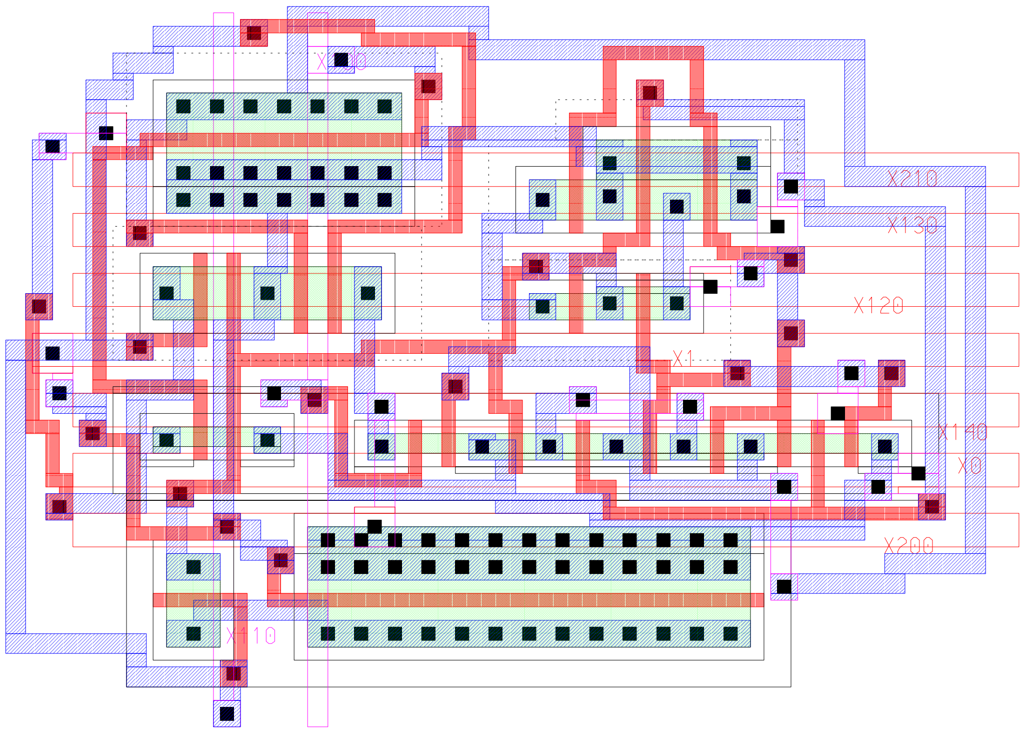
TEMPO layout: PTL-42COMP (area = 7934μm^2)



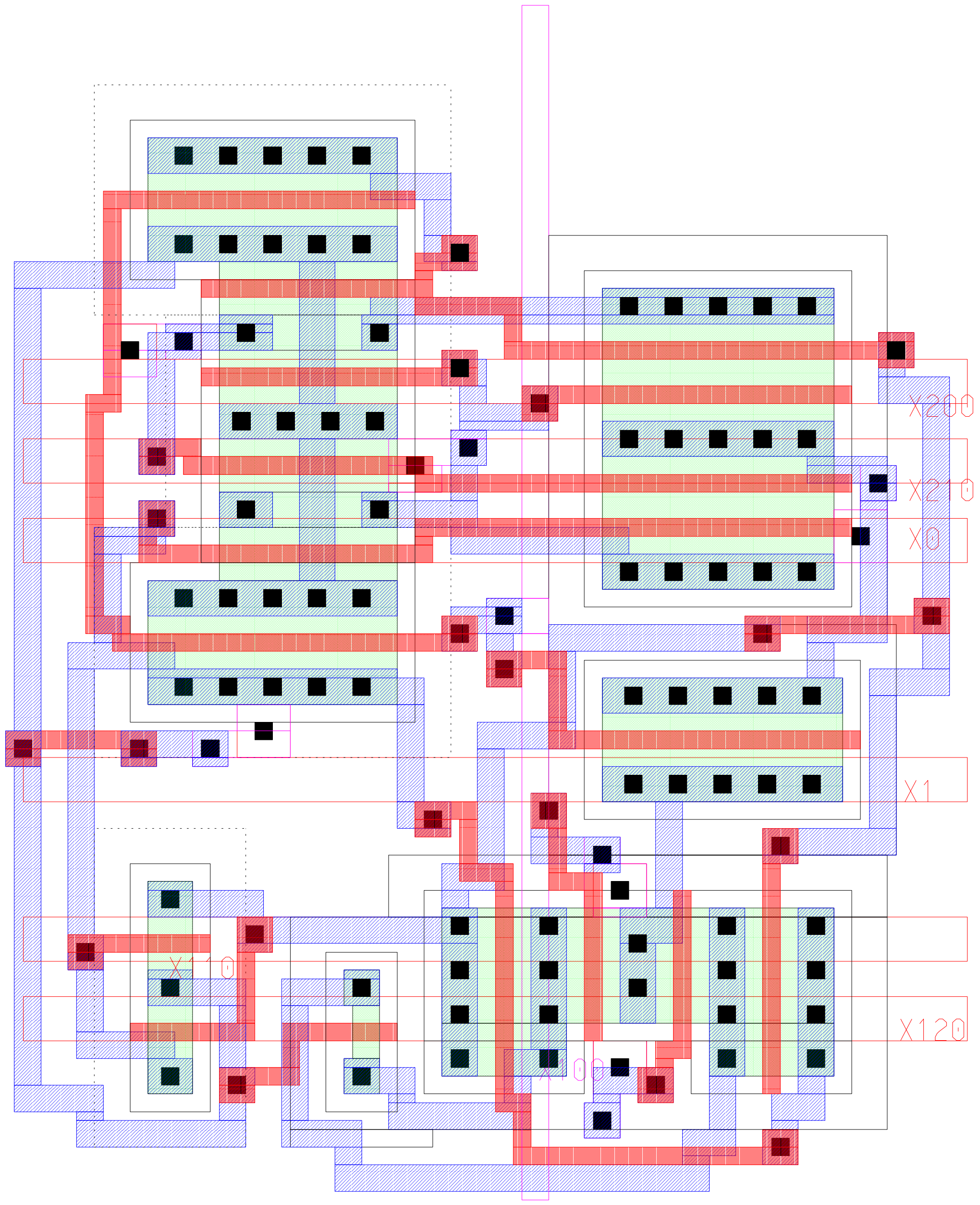
TEMPO layout: PTL-RBA (area = 6610μm^2)



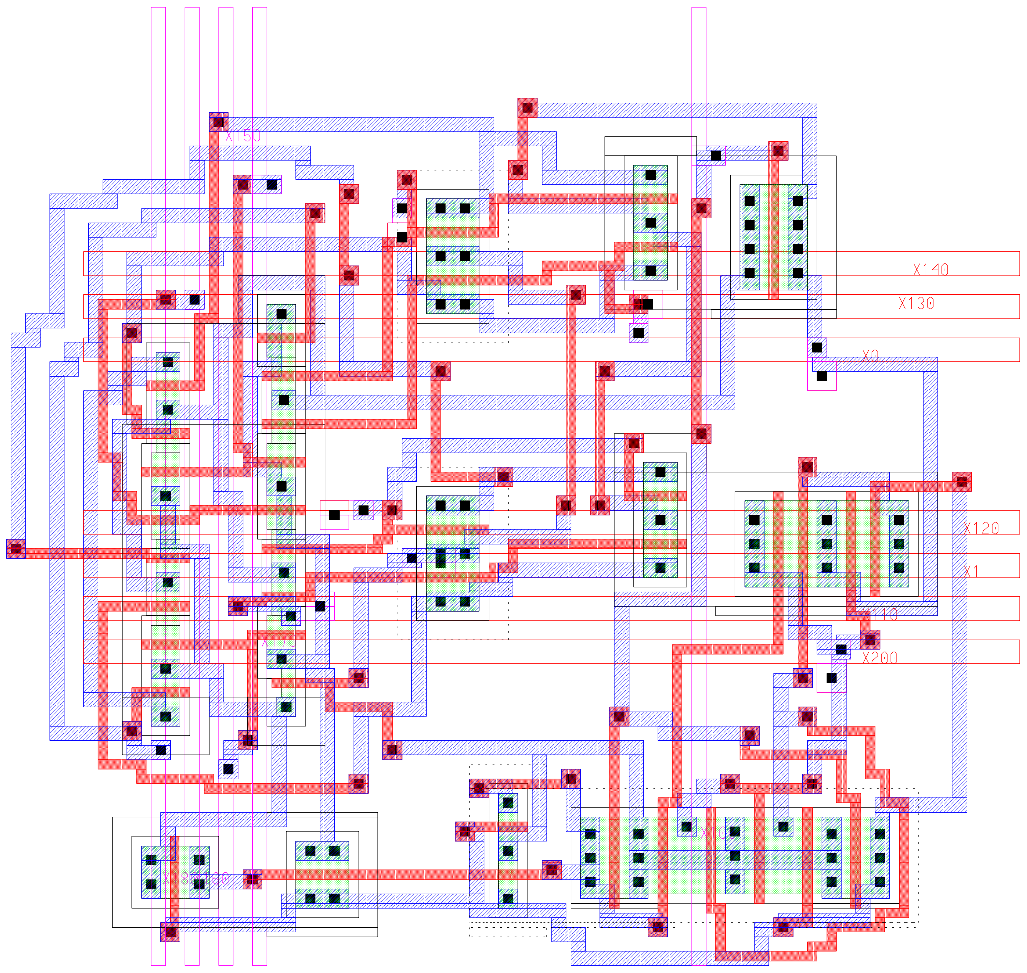
TEMPO layout: QNP-FA (area = 1468μm^2)



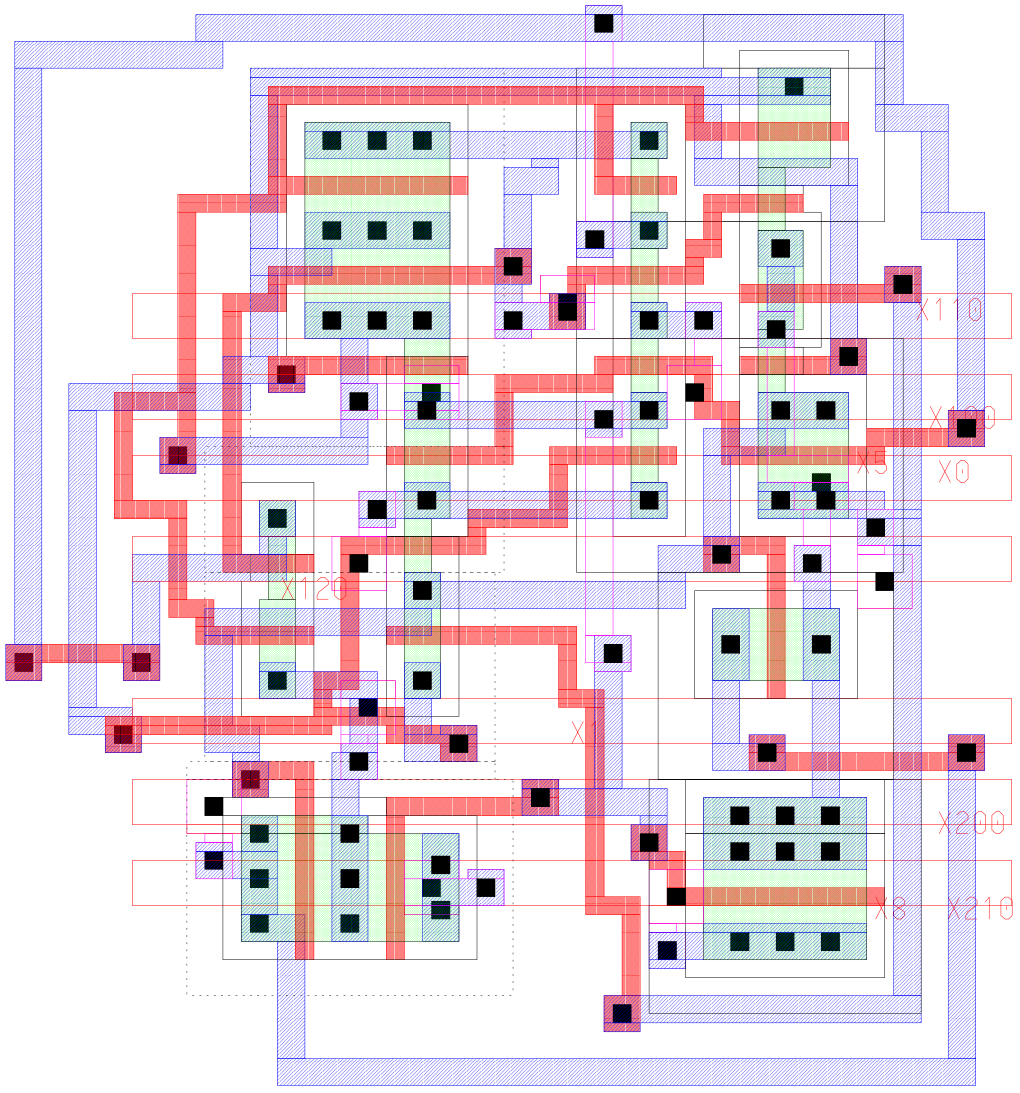
TEMPO layout: SA-FF (area = 1312μm^2)



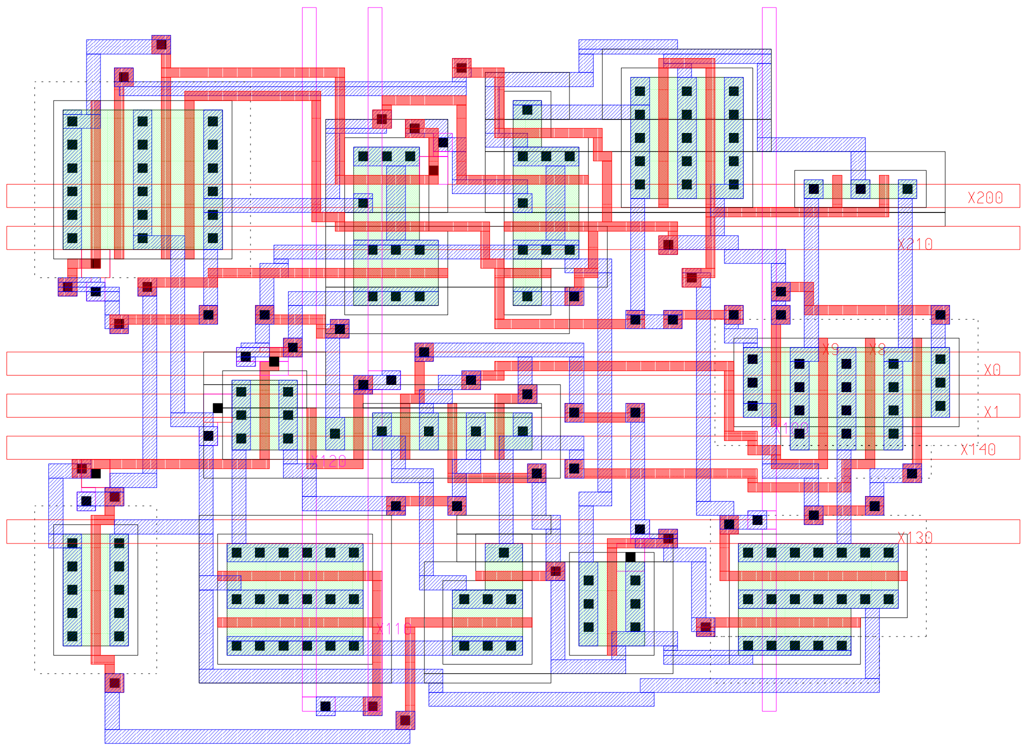
TEMPO layout: SA-MUX-FF (area = 3780μm^2)



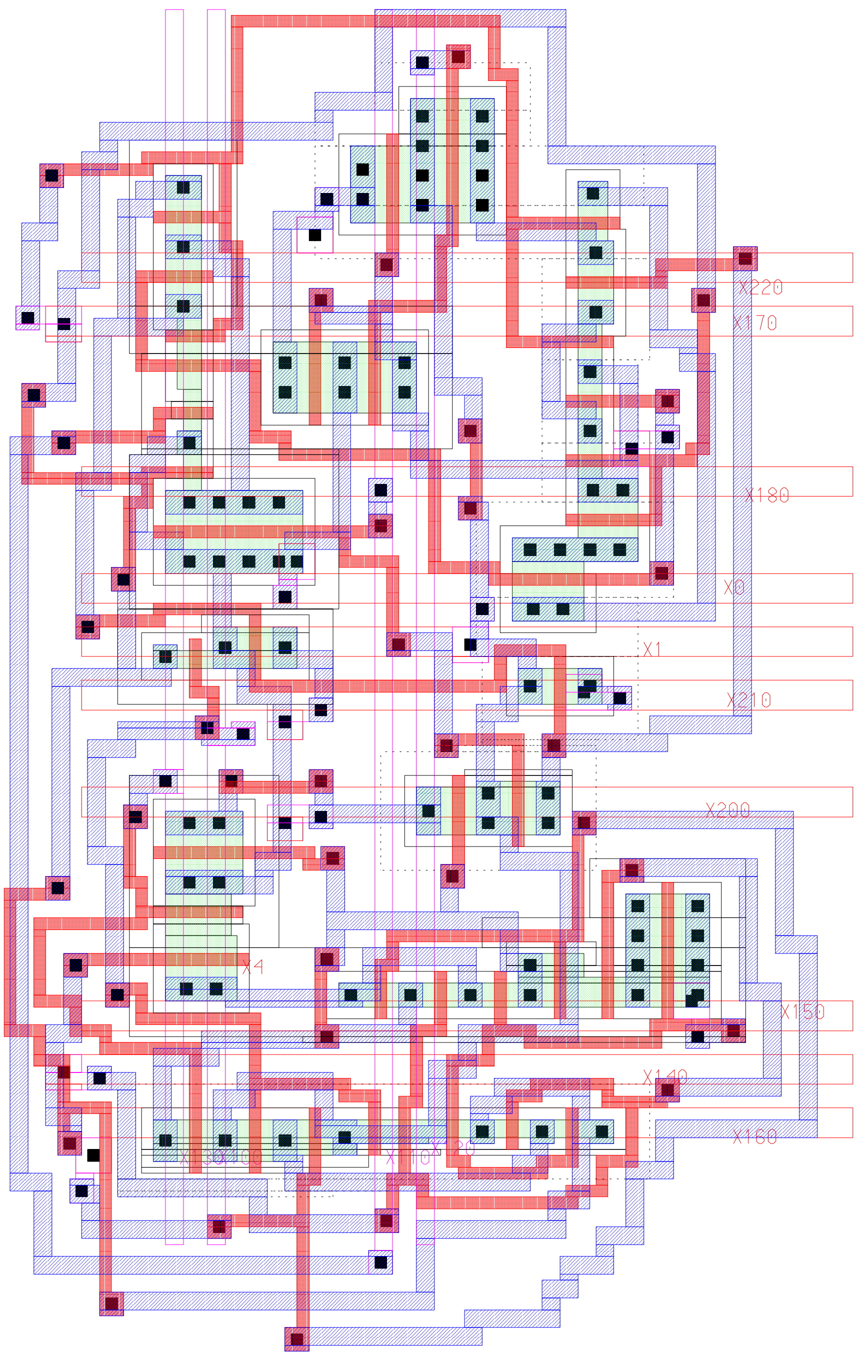
TEMPO layout: T17-FA (area = 1199μm^2)



TEMPO layout: XOR-FF (area = 3072μm^2)



TEMPO layout: ZIP-FA2 (area = 2888μm^2)

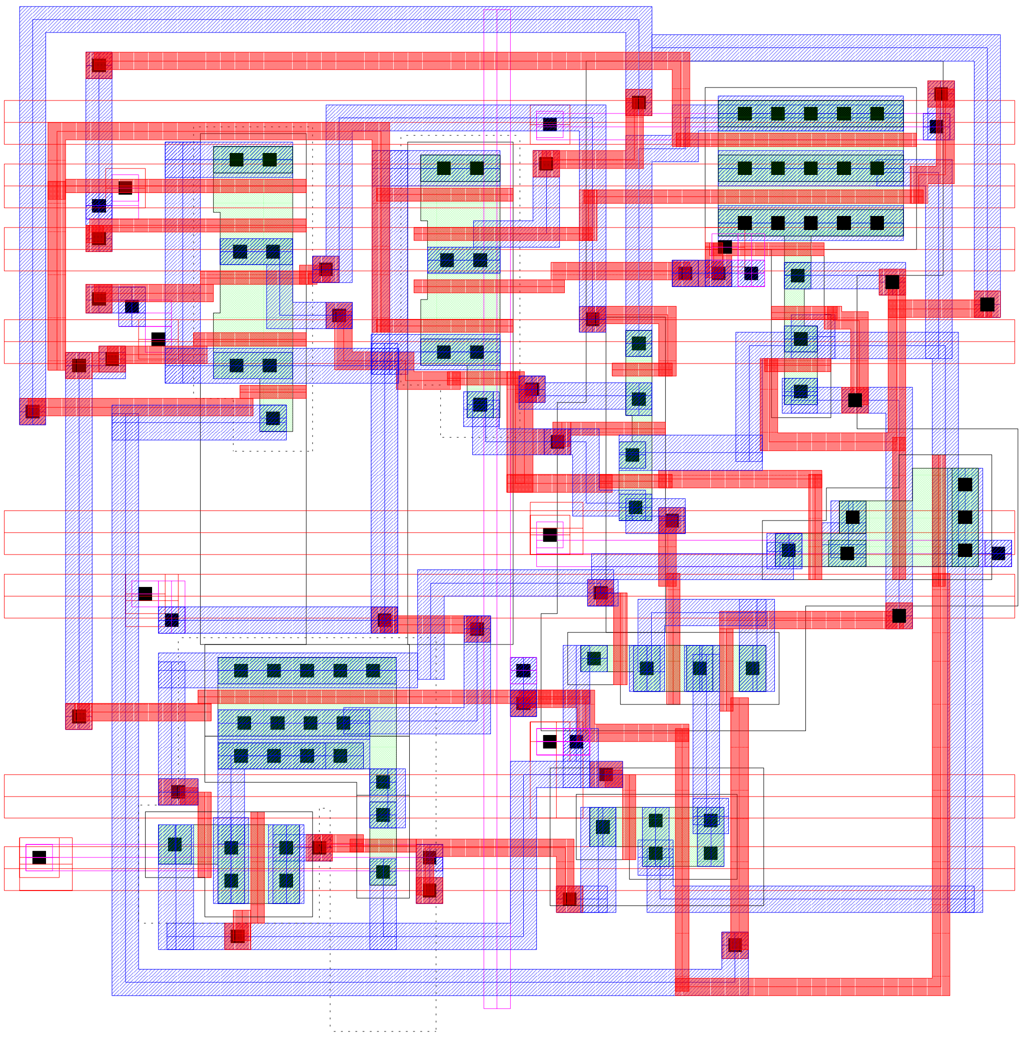


APPENDIX E

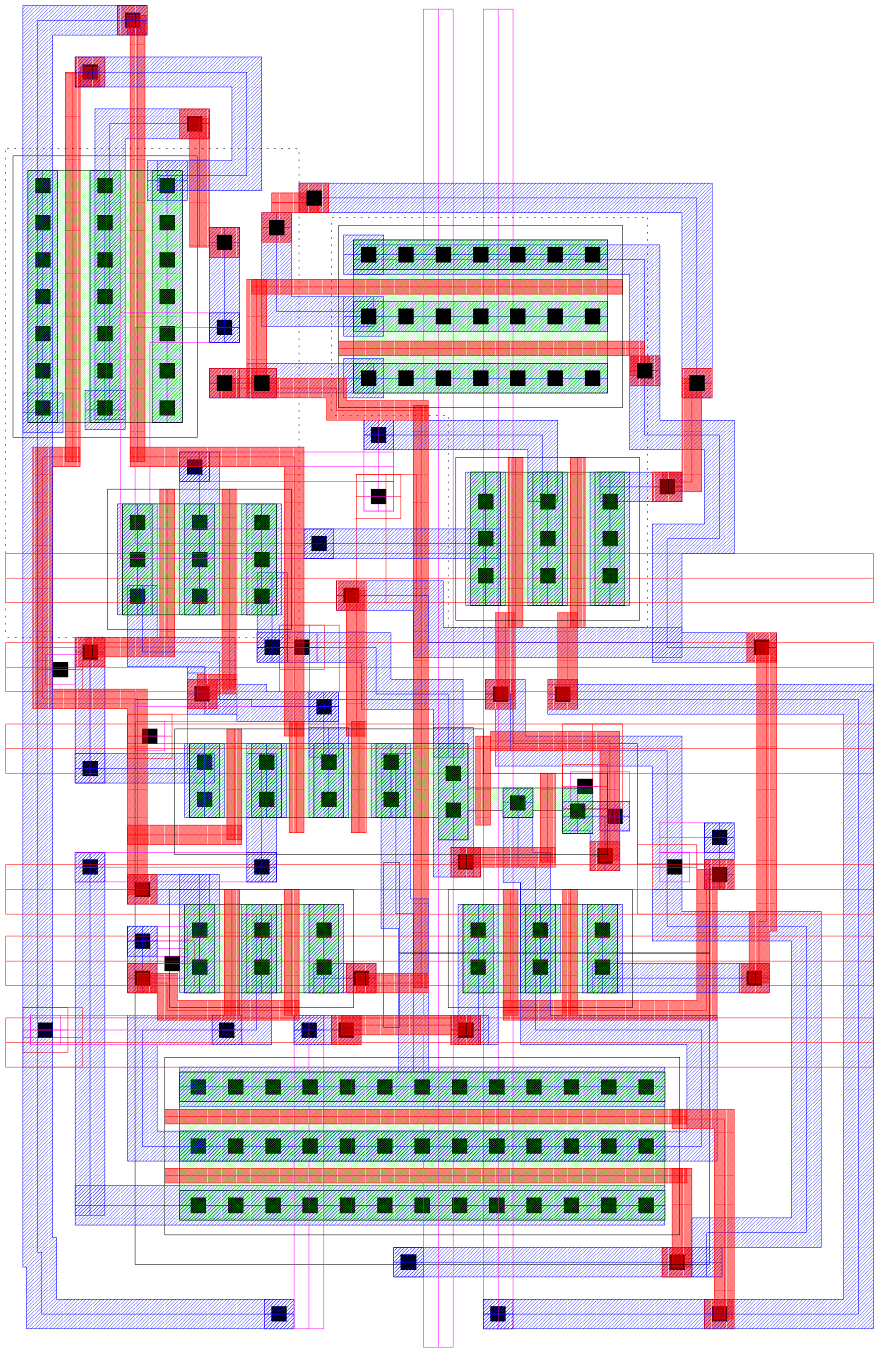
TEMPO Layouts (Post-Compaction)

This appendix contains the final benchmark layouts produced by TEMPO at the conclusion of the compaction step.

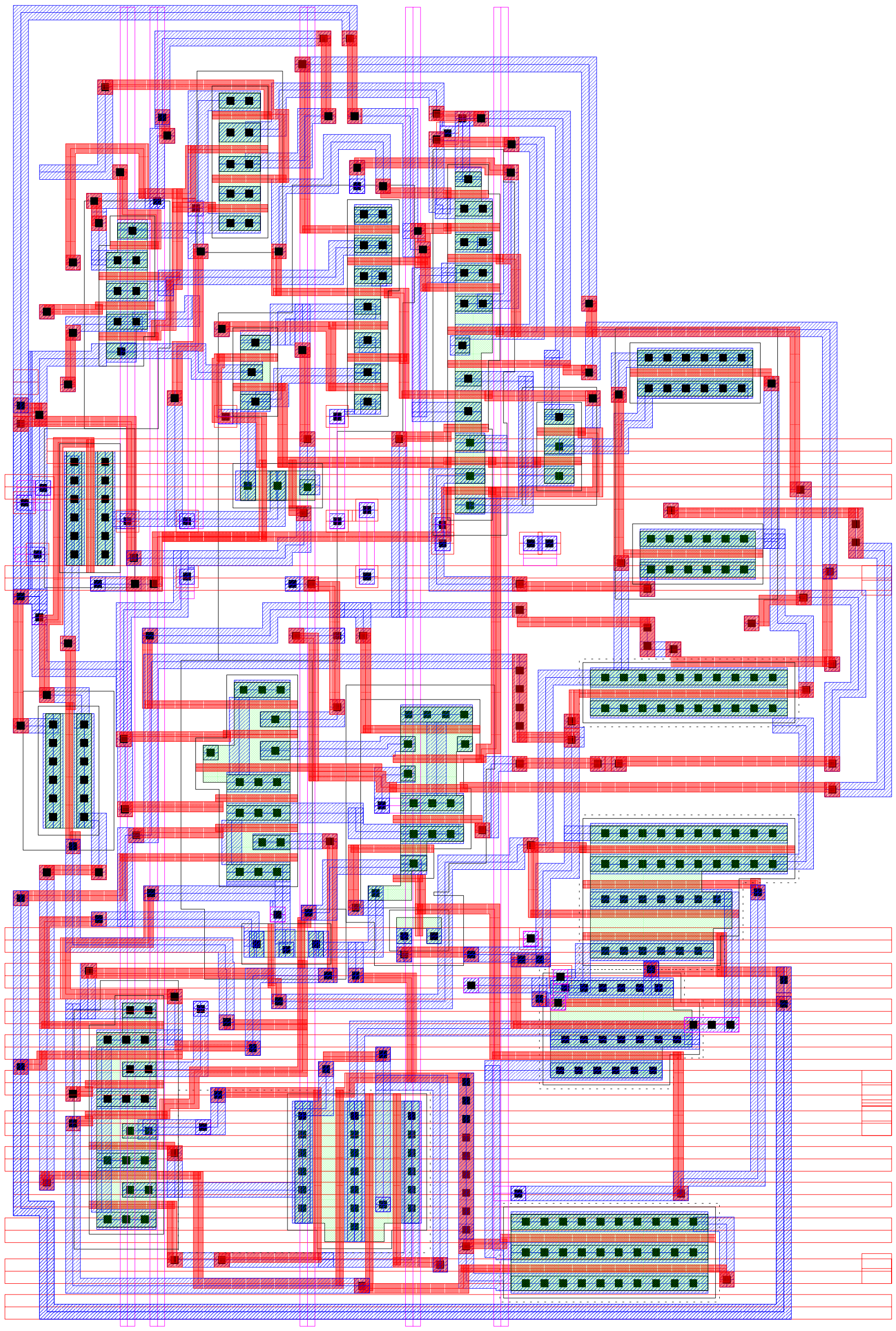
Final layout: AOI-LFF (area = 2100μm^2)



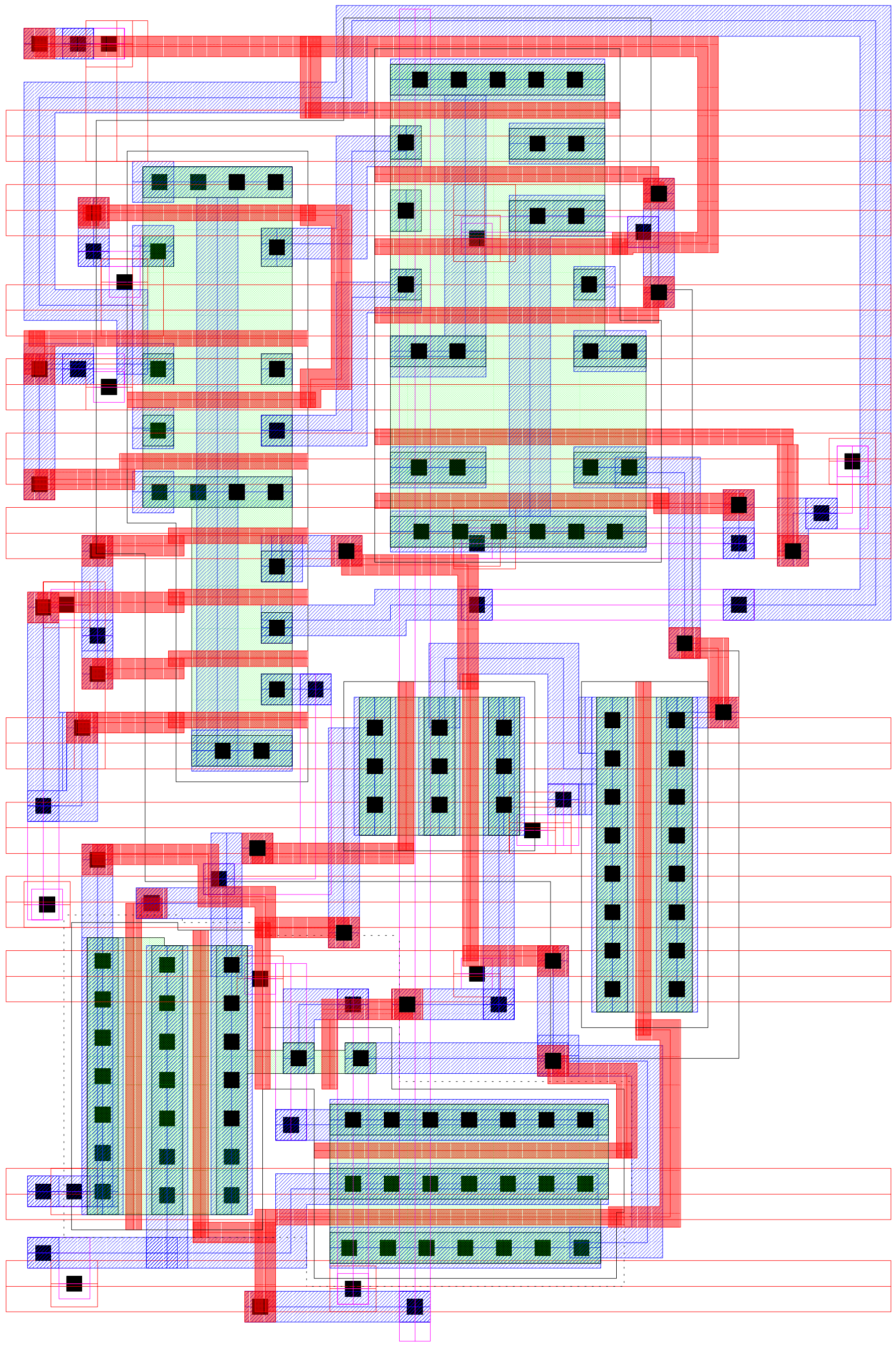
Final layout: BLAIR-FF (area = 1897μm^2)



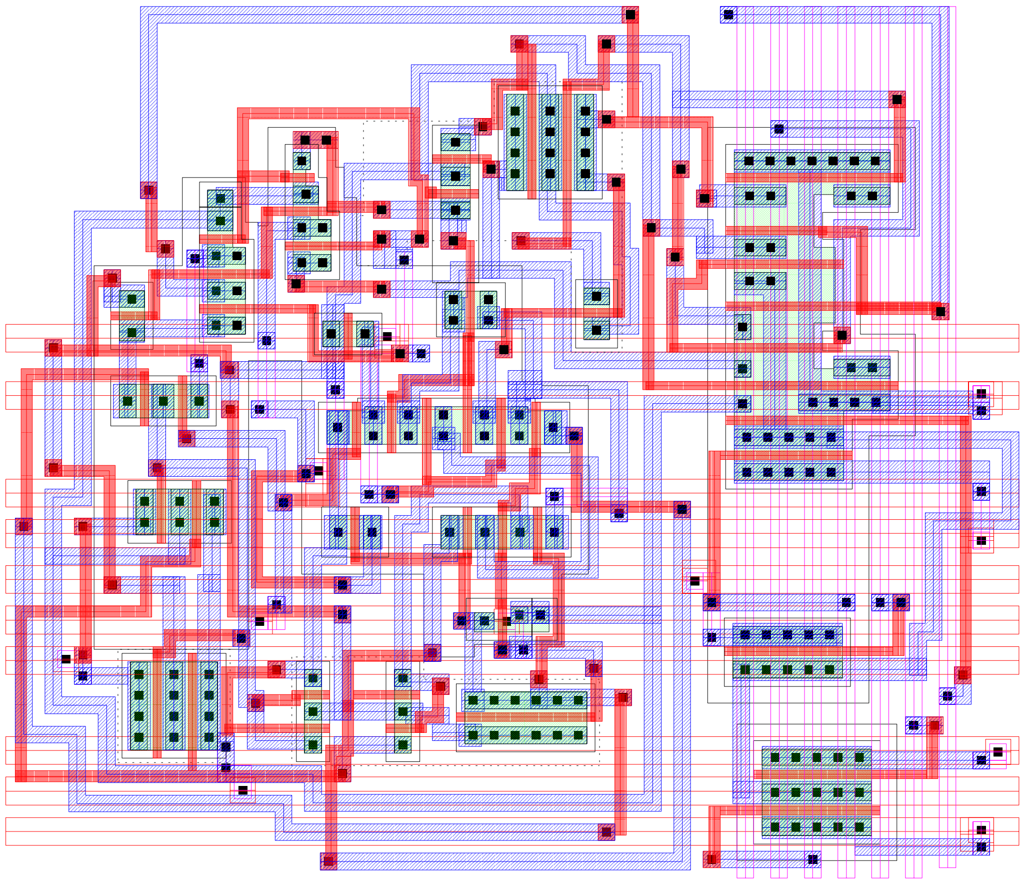
Final layout: DCSL3-42COMP (area = 7672μm^2)



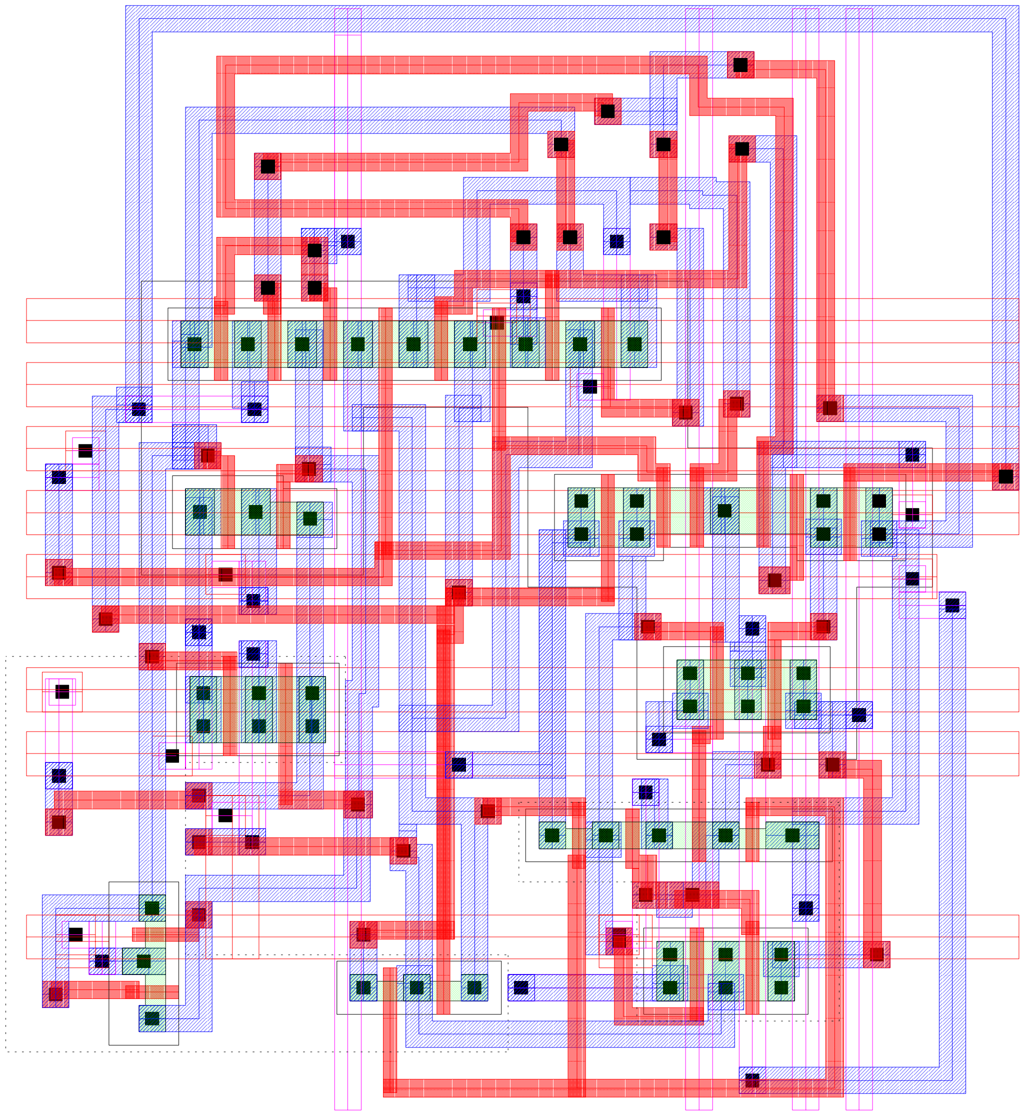
Final layout: DCVSL-XOR4 (area = 1789μm^2)



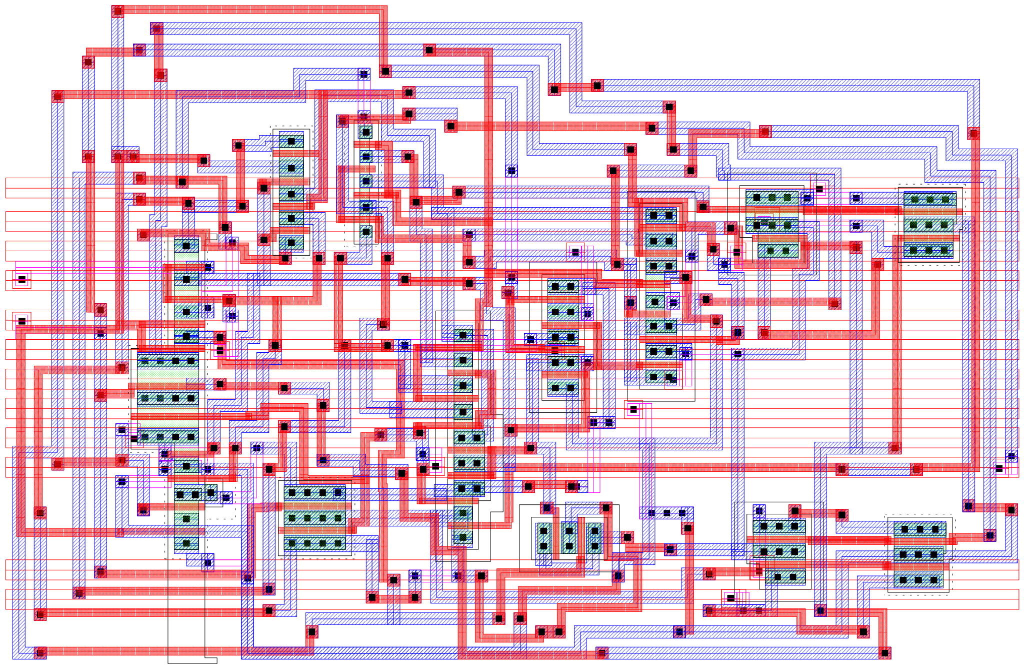
Final layout: DEC-CSA-MUX (area = 4511μm^2)



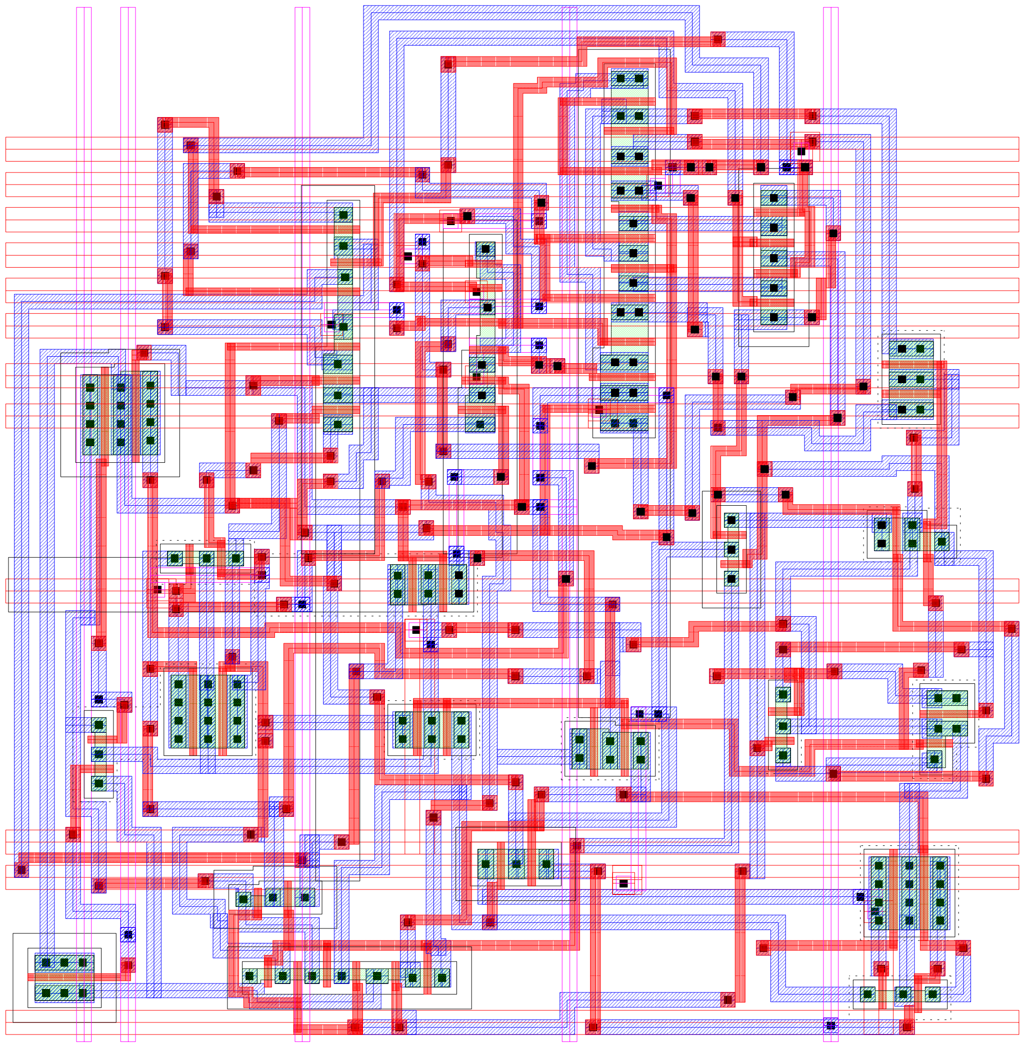
Final layout: DIFF-FA (area = 2224μm^2)



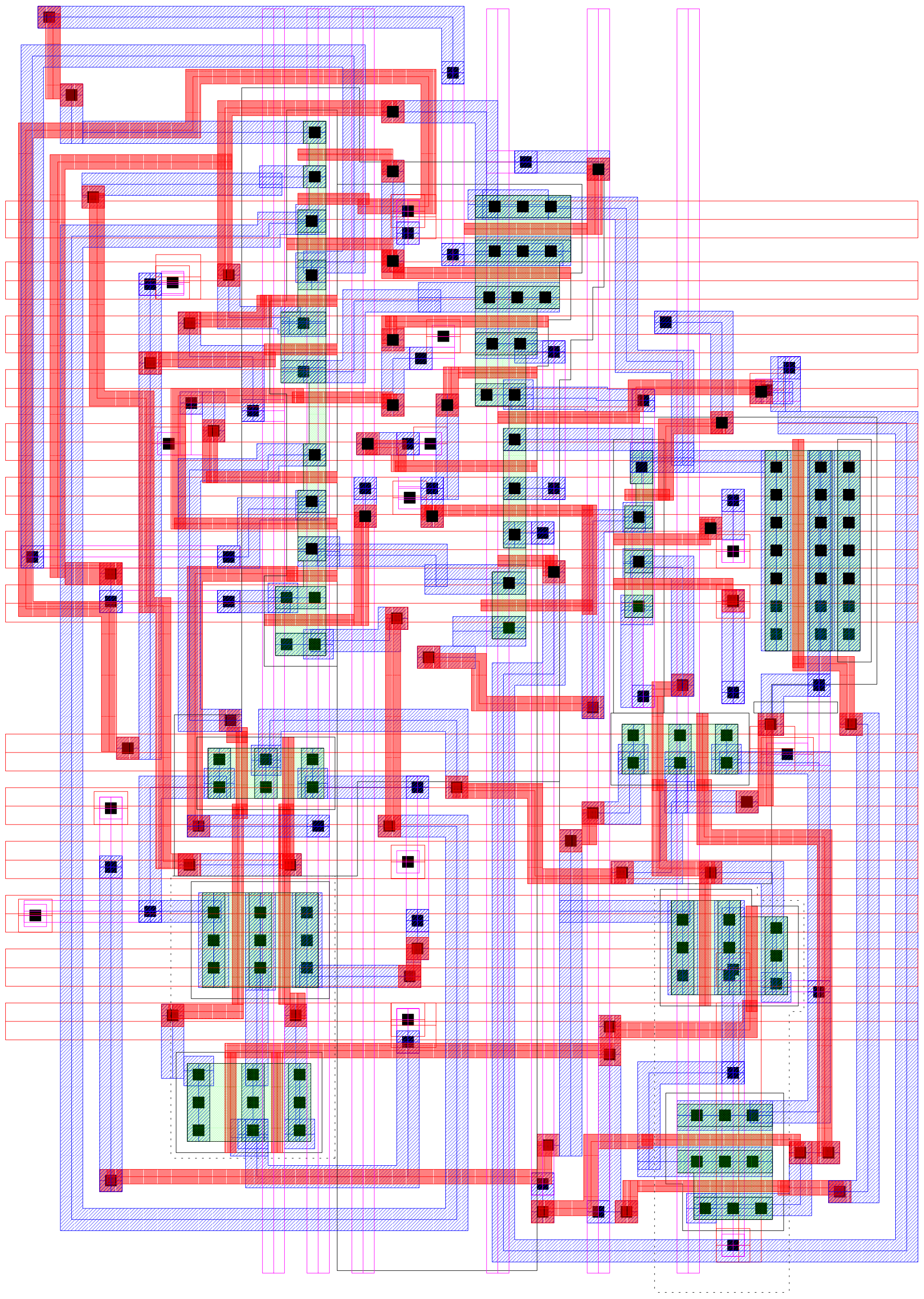
Final layout: DPL-FA (area = 6392μm^2)



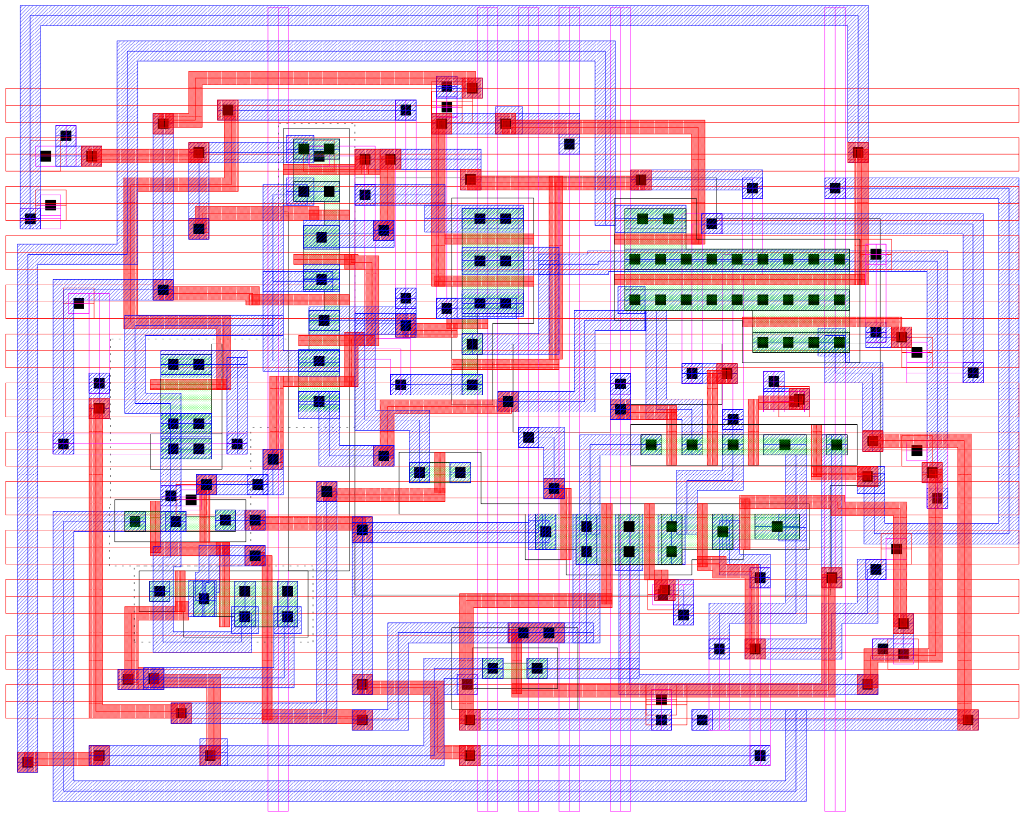
Final layout: DPTL-42COMP (area = 7003μm^2)



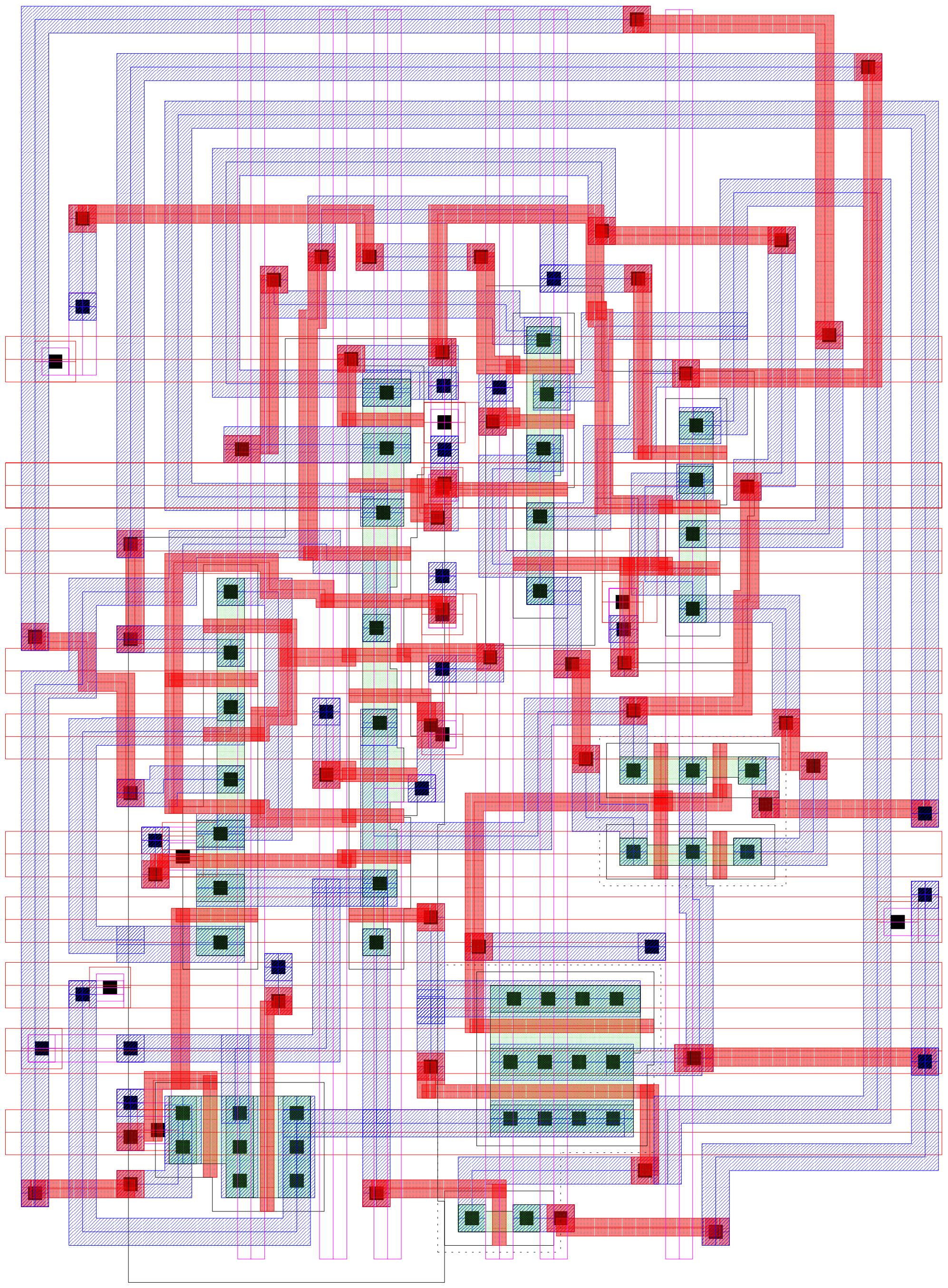
Final layout: EMODL-CLA (area = 3336μm^2)



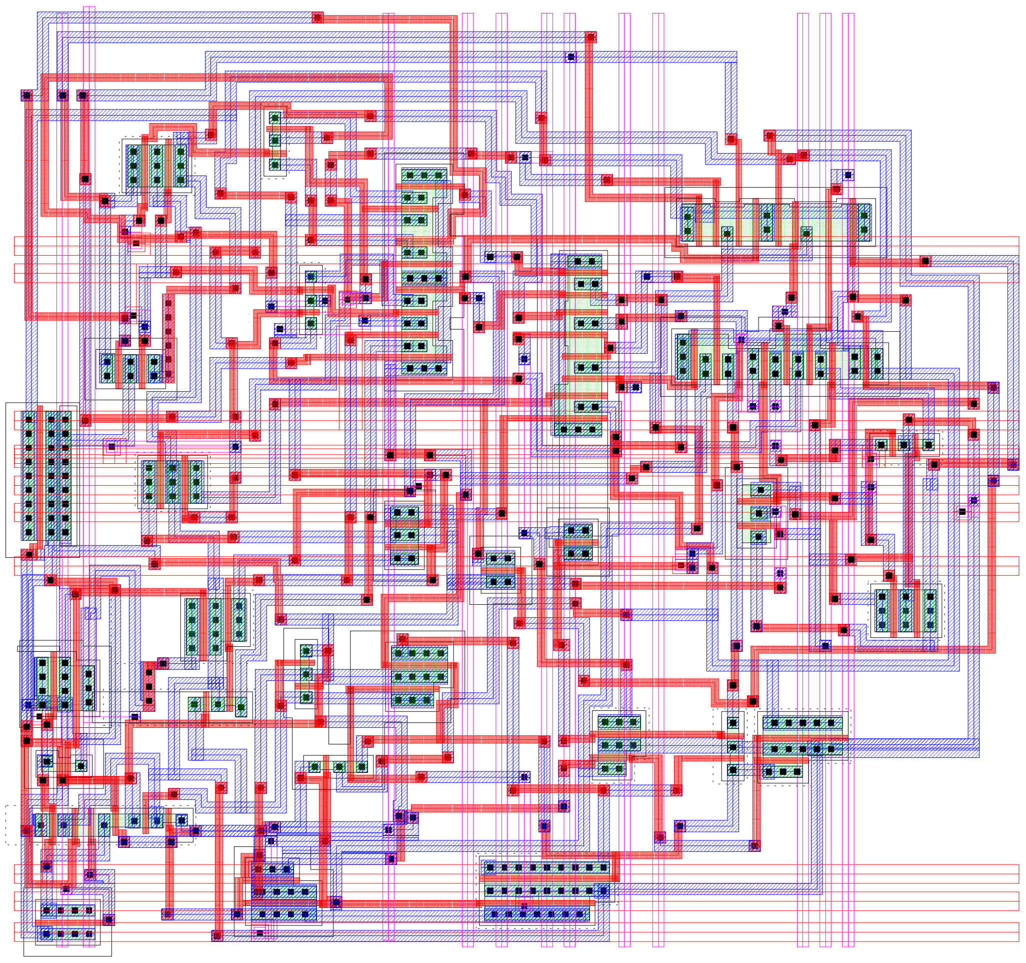
Final layout: GHZ-CCU-MERGE (area = 2832μm^2)



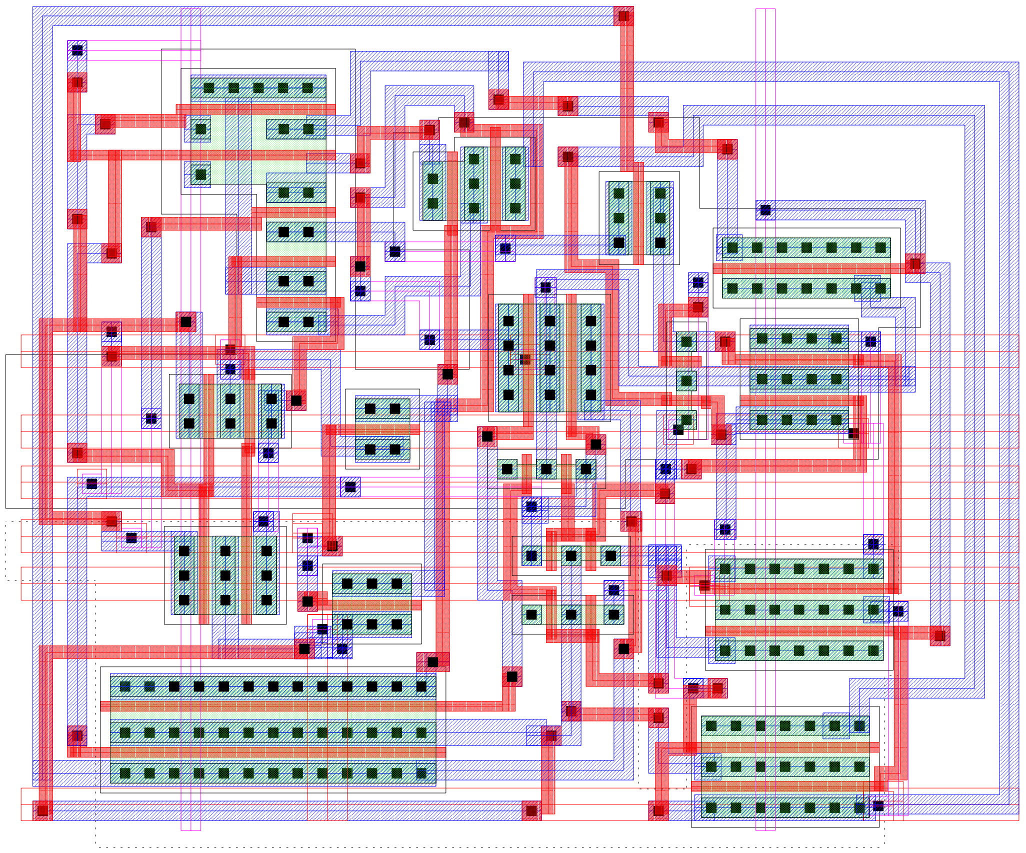
Final layout: GHZ-CCU-PROP (area = 2280μm^2)



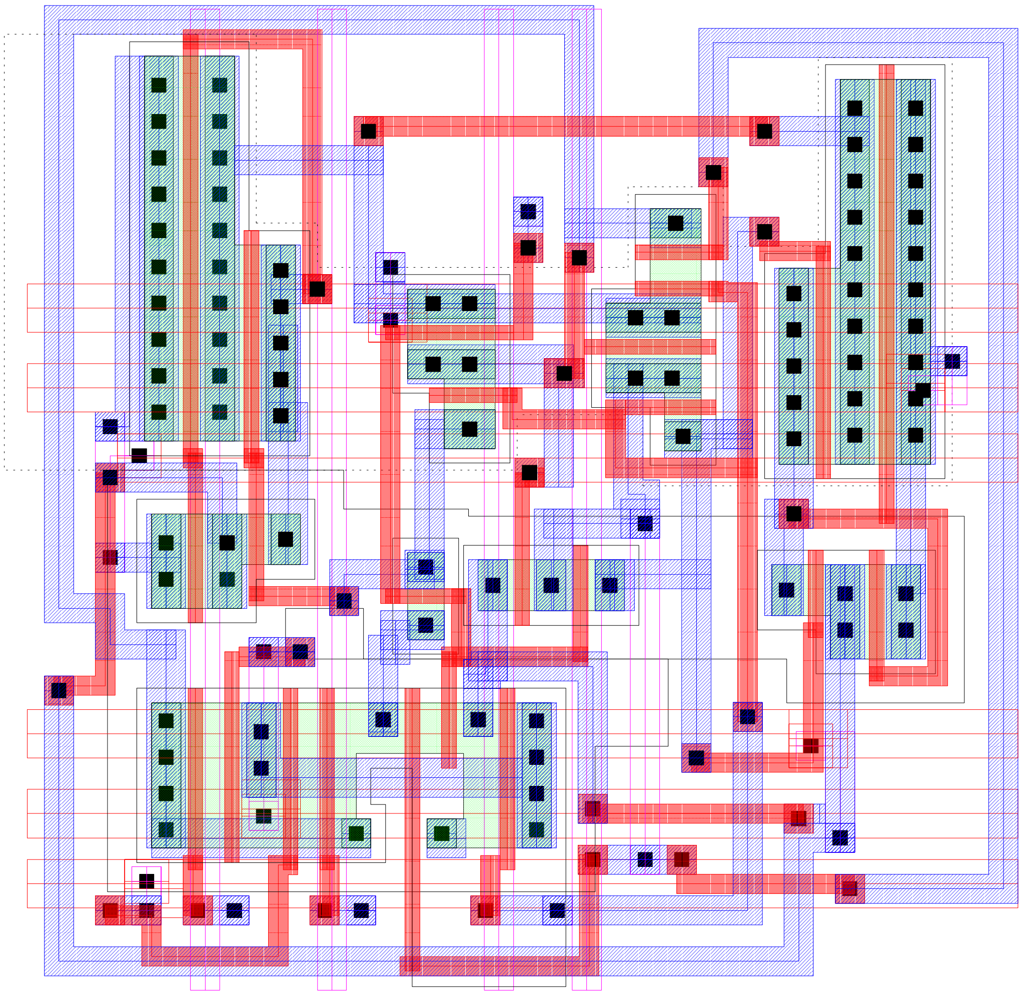
Final layout: GHZ-MUX8-LA (area = 10670μm^2)



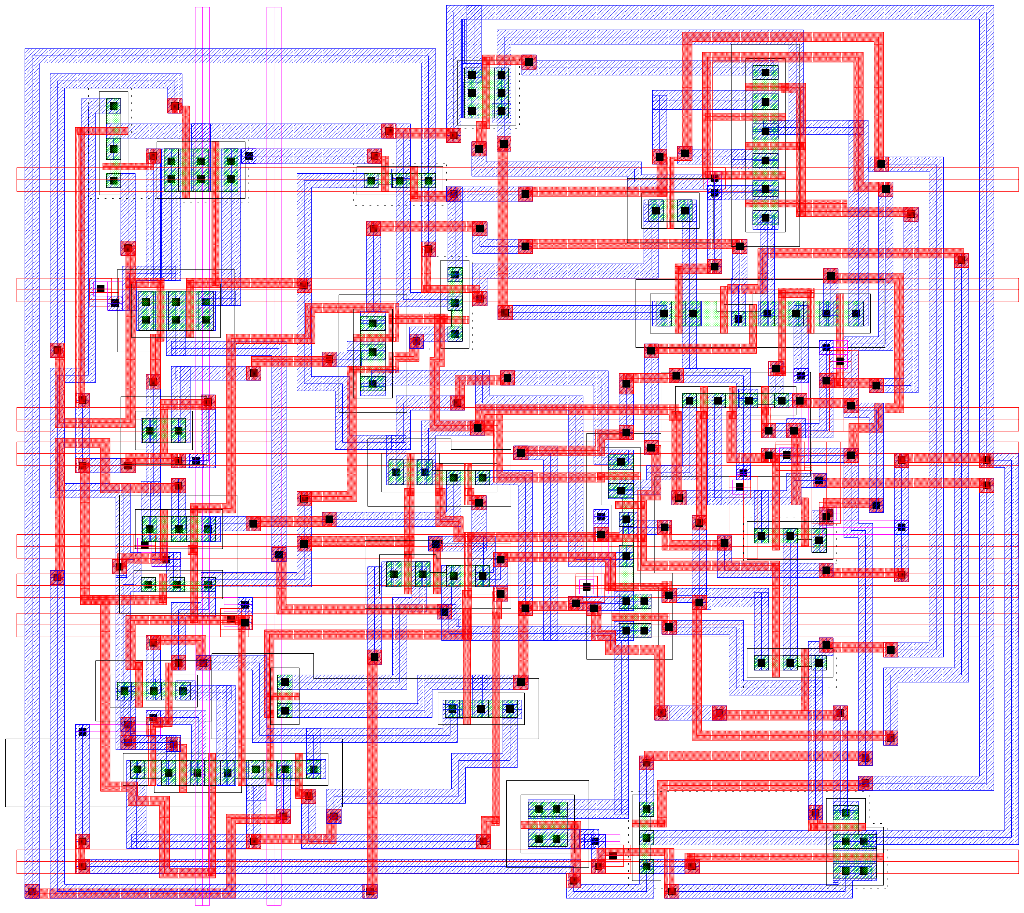
Final layout: MUX2-SDFF (area = 3054μm^2)



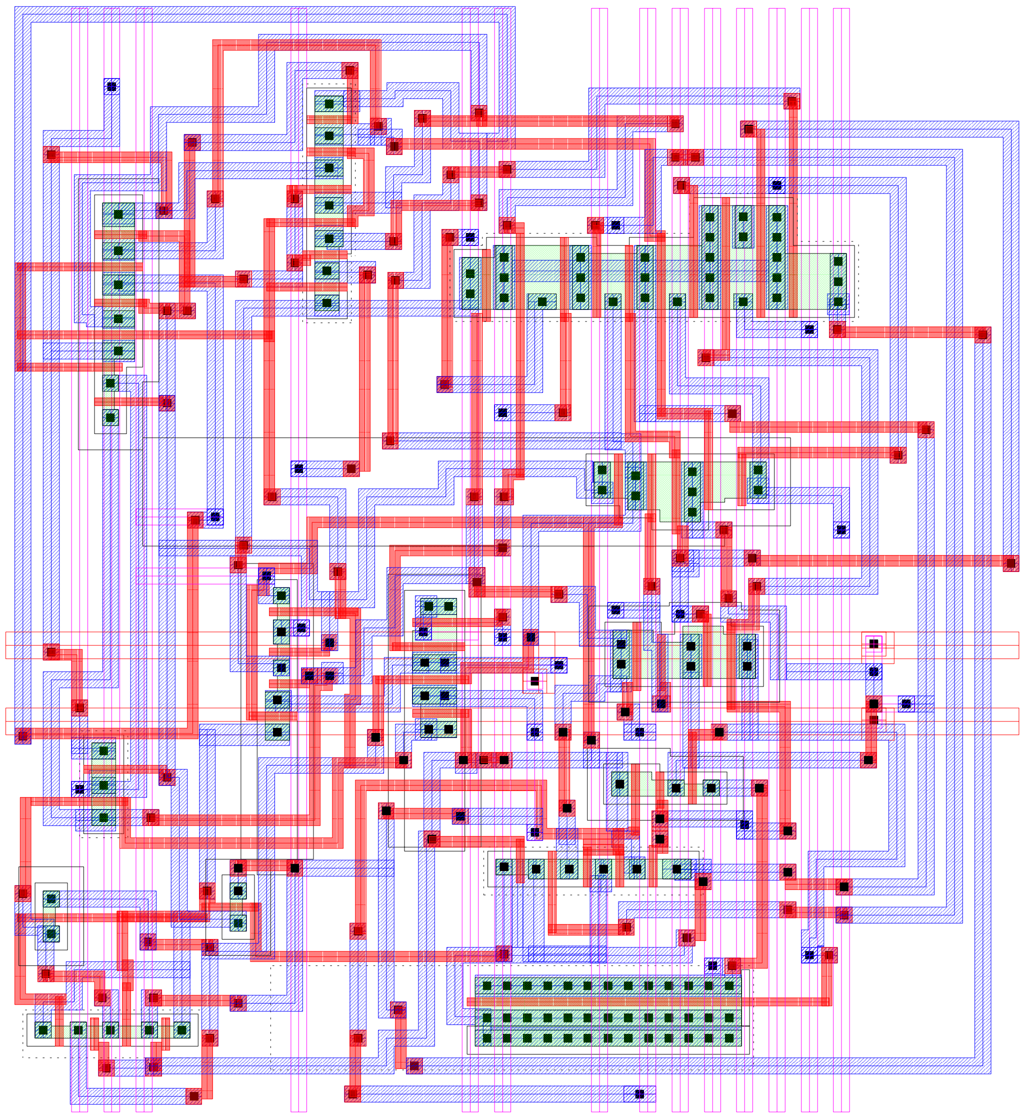
Final layout: MUXFF (area = 1654μm^2)



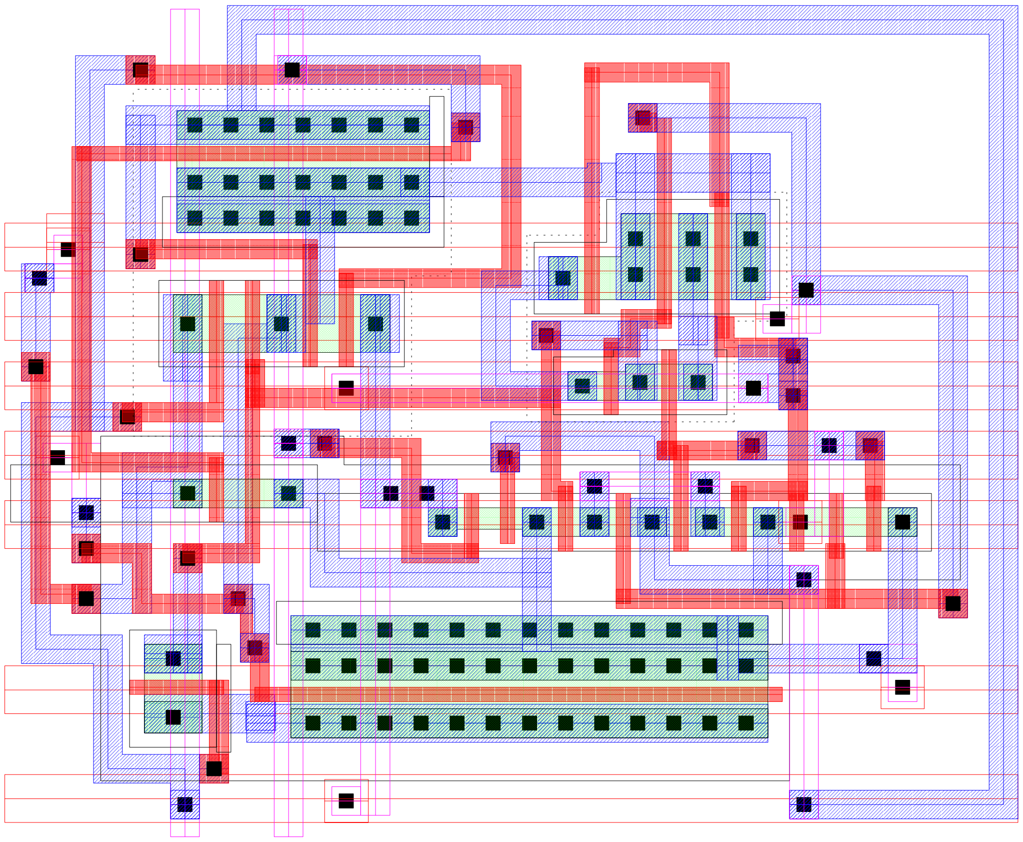
Final layout: PTL-42COMP (area = 6339μm^2)



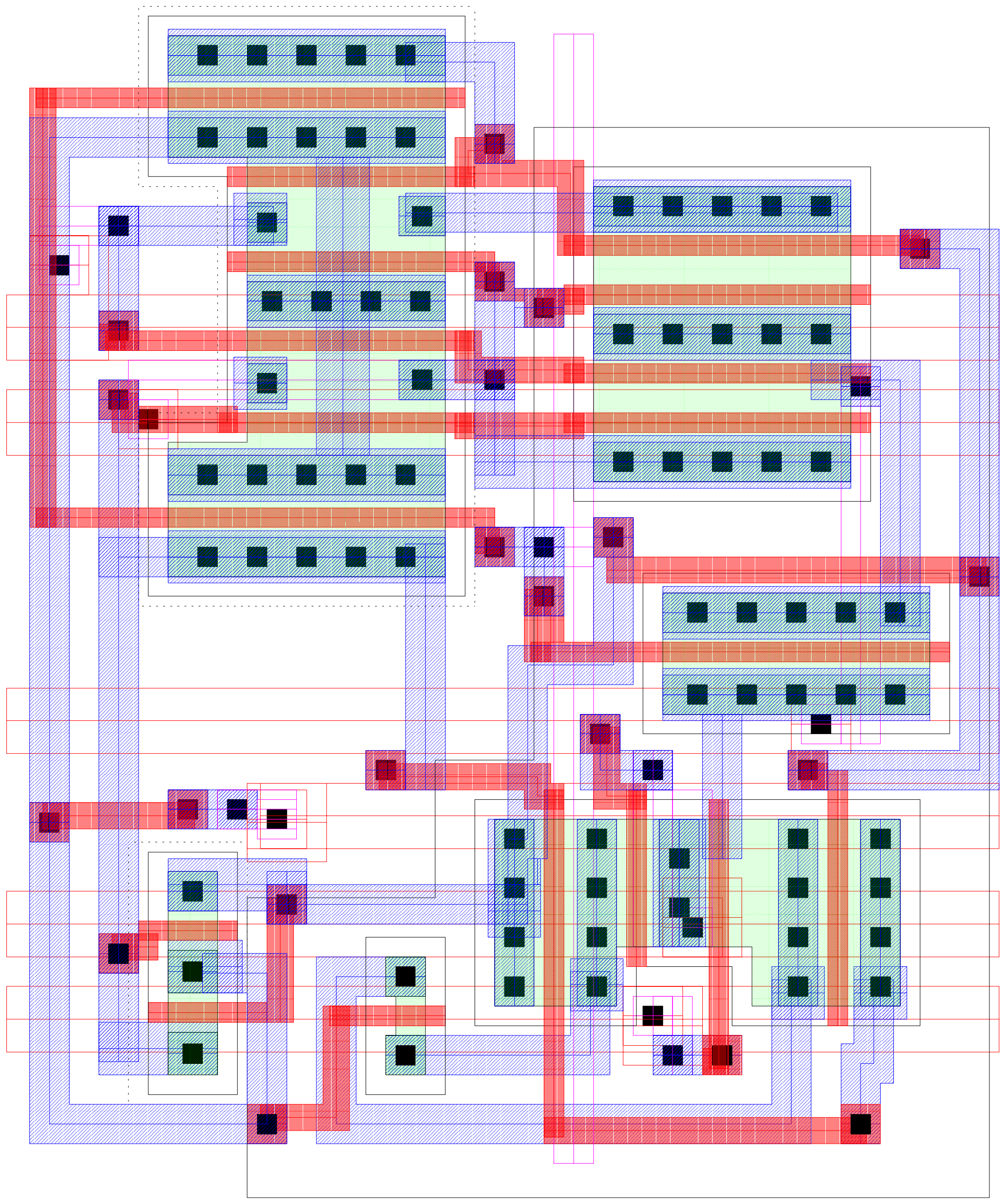
Final layout: PTL-RBA (area = 6237μm^2)



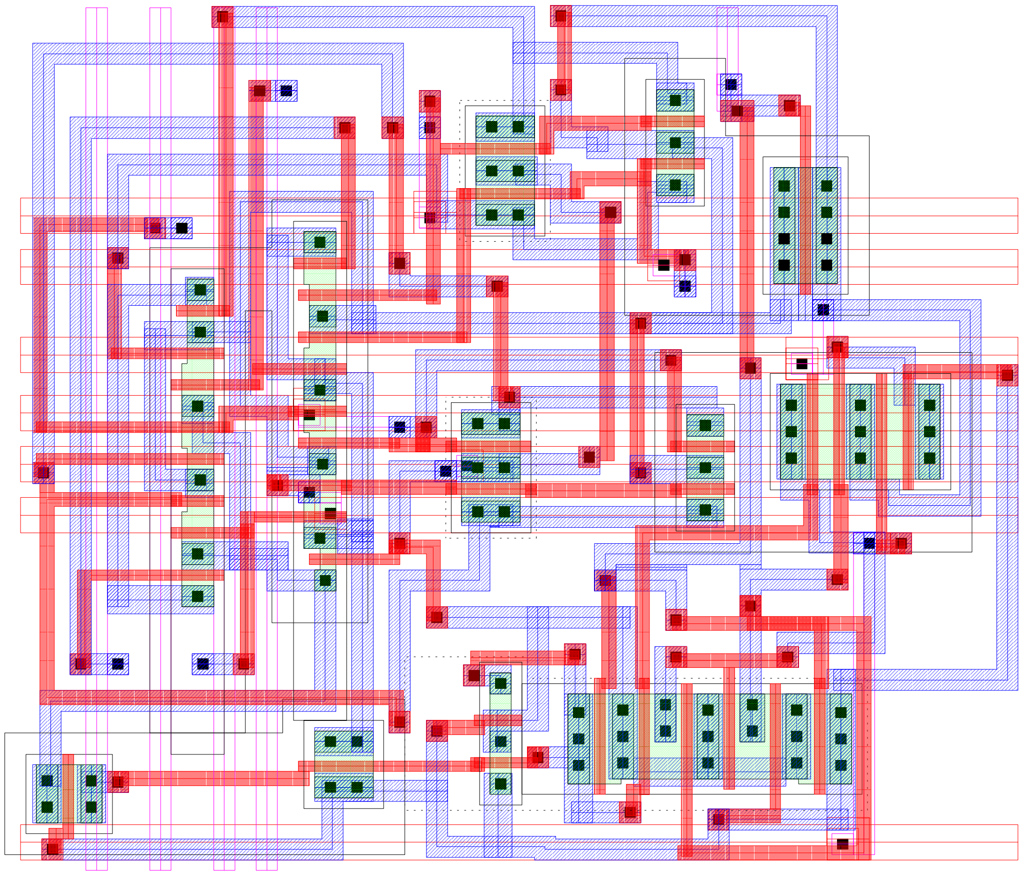
Final layout: QNP-FA (area = 1463μm^2)



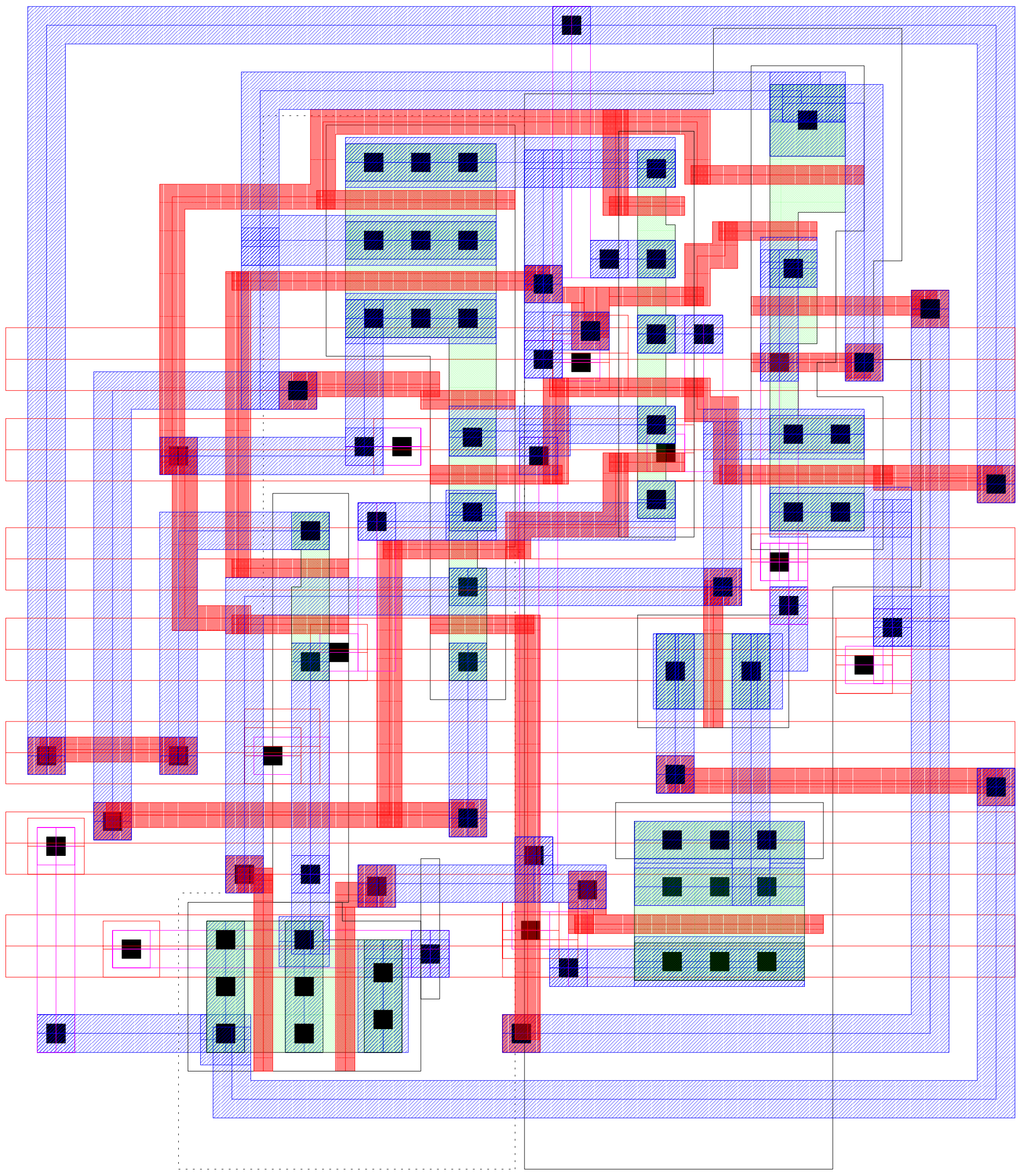
Final layout: SA-FF (area = 1041μm^2)



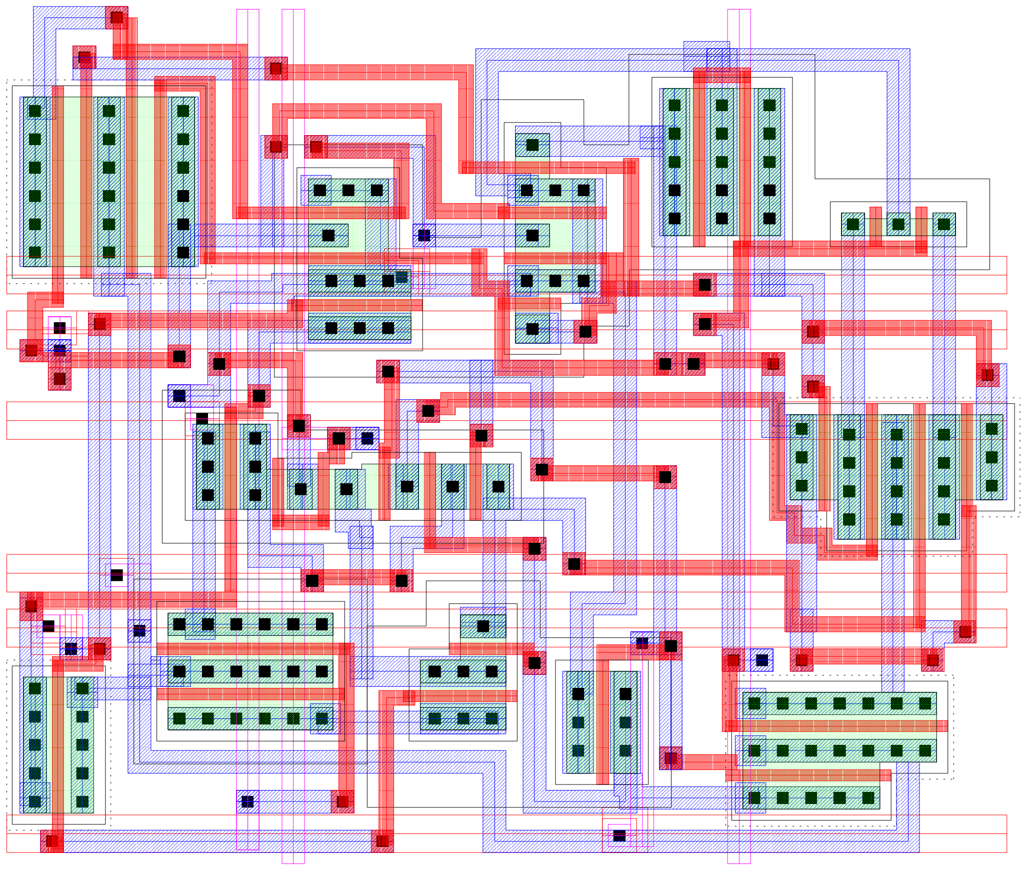
Final layout: SA-MUX-FF (area = 2765μm^2)



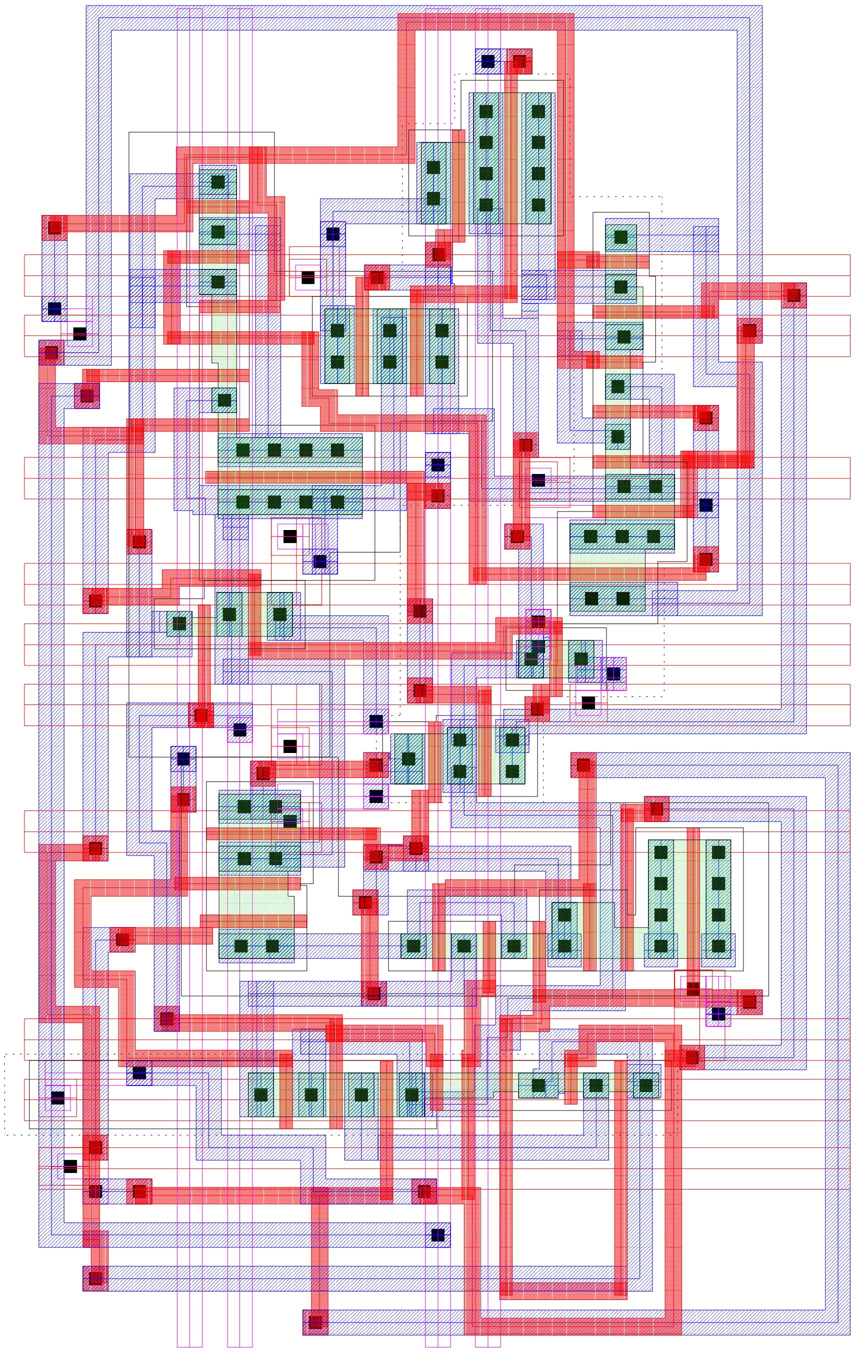
Final layout: T17-FA (area = 1143μm^2)



Final layout: XOR-FF (area = 2394μm^2)



Final layout: ZIP-FA2 (area = 2546μm^2)



BIBLIOGRAPHY

BIBLIOGRAPHY

[1] C. Bamji, R. Varadarajan, “Leaf cell and Hierarchical Compaction Techniques,” *Kluwer Academic Publishers*, May 1997.

[2] P. Bannon, “Alpha 21364: A Scalable Single-Chip SMP,” in proc. *1998 MicroProcessor Forum*, http://www.digital.com/alphaoem/present/sld001.htm.

[3] P. Barth, A Davis-Putnam Based Enumeration Algorithm for Linear Pseudo-Boolean Optimization”, Technical Report MPH-95-2-003, Max Planck Institut Fur Informatik, Saarbrucken, Germany, January 1995.

[4] B. Basaran, R. Rutenbar, L. Carley, “Latchup-Aware Placement and Parasitic-Bounded Routing of Custom Analog Cells,” in proc. *1993 CICC*, pp. 415–421.

[5] B. Basaran, “Optimal Diffusion Sharing in Digital and Analog CMOS Layout”, Ph.D. Dissertation, Carnegie Mellon University, CMU Report No. CMUCAD-97-21, May 1997.

[6] B. Bernhardt et al, “Complementary GaAs (CGaAs™): A High Performance BiCMOS Alternative”, in proc. *1995 GaAs IC Symp.*, pp. 18–21.

[7] G. Blair, “Comments on ‘New Single-Clock CMOS Latches and Flip-Flops with Improved Speed and Power Savings,” *IEEE J. Solid State Circ.*, 32(10), Oct. 1997, p. 1611.

[8] G. Bois and E. Cerny, “Lazy Constraint Generation for 2D Compaction,” in proc. *1990 MCNC International Workshop on Layout Synthesis*, pp. 1–16.

[9] J. Burns and J. Feldman, “C5M—A Control Logic Layout Synthesis System for High-Performance Microprocessors,” *IEEE Trans. on CAD*, 17(1), Jan. 1998, pp. 14–23.

[10] J. Burns and K. Nowka, “Parallel Condition-Code Generation for High-Frequency PowerPC Microprocessors,” in proc. *1998 Symp. on VLSI Circ.*, p. 115.

[11] S. Chakravarty, X. He and S. Ravi, “On Optimizing nMOS and Dynamic Functional Cells,” in proc. 1990 ISCAS, pp. 1701–1704.

[12] E. Charbon, E. Malavesi, A. Sangiovanni-Vincentelli, “Generalized Constraint Generation for Analog Circuit Design,” in proc. *1993 ICCAD*, pp. 408–414.

[13] E. Charbon, E. Malavesi, D. Pandini, A. Sangiovanni-Vincentelli, “Simultaneous Placement and Module Optimization of Analog ICs,” in proc. *1994 DAC*, pp. 31–35.

[14] C. Cheng, “RISA: Accurate and Efficient Placement Routability Modeling”, in proc. *1994 ICCAD*, pp. 690–695.

[15] U. Choudhury, A. Sangiovanni-Vincentelli, “Automatic Generation of Parasitic Constraints for Performance-Constrained Physical Design of Analog Circuits,” *IEEE Trans. on CAD*, 12(2), Feb. 1993, pp. 208–224.

[16] S. Chow, H. Chang, J. Lam, and Y. Liao, “The Layout Synthesizer: An Automatic Block Generation System,” in proc. *1992 CICC*., pp. 11.1.1–11.1.4.

[17] J. Cohn, “Automatic Device Placement for Analog Cells in KOAN,” Ph.D. Dissertation, Carnegie Mellon University, CMUCAD-92-07, 1992.

[18] J. Cohn, D. Garrod, R. Rutenbar and L. R. Carley, “Analog Device-Level Layout Automation”, Kluwer Academic Publishers, Boston MA., 1994.

[19] A. Conn, P. Coulman, R. Haring, G. Morrill and C. Visweswariah, “Optimization of Custom MOS Circuits by Transistor Sizing,” in proc. *1996 ICCAD*, pp. 174–180.

[20] T. Corman, C. Leiserson and R. Rivest, “An Introduction to Algorithms,” The MIT Press, Cambridge, 1990.

[21] S. Devadas, “Optimal Layout Via Boolean Satisfiability,” in proc. *1989 ICCAD*, pp. 294–297.

[22] Duet Technologies inc., “Masterport Compaction Users Manual”

[23] W. Elmore, “The Transient Response of Damped Linear Networks with Particular Regard to Wide Band Amplifiers,” Journal of Applied Physics, vol. 19, Jan 1948.

[24] L. Euler, “Solutio Problematis ad Geometriam Situs Pertinentis,” *Commentarii Academiae Petropolitanae*, vol. 8, 1736, pp. 128–140 (in latin).

[25] C. Fiduccia and R. Mattheyses, “A Linear-Time Heuristic for Improving Network Partitions,” in proc. *19th DAC*, 1982, pp. 241–247.

[26] J. Fishburn and A. Dunlop, “TILOS: A Posynomial Approach to Transistor Sizing,” in proc.

1985 ICCAD, pp. 326–328.

[27] V. Friedman and S. Liu, “Dynamic Logic CMOS Circuits,” *IEEE J. Solid State Circ.*, 19(2), Apr. 1984, p. 265.

[28] M. Fukui, N. Shinomiya, T. Akino, “A New Layout Synthesis for Leaf Cell Design”, in proc. *1995 ASP-DAC*, pp. 259-263.

[29] H. N. Gabow and R. E. Tarjan, “Faster Scaling Algorithms for Network Problems,” *SIAM Journal on Computing*, vol. 18, pp. 1013-1036, 1989.

[30] M. Garey and D. Johnson, “The Rectilinear Steiner Tree Problem is NP-Complete,” *SIAM Journal of Applied Mathematics*, vol. 32, 1977 pp. 826–834.

[31] D. Garrod, “Device-Level Routing of Analog Cells in Anagram II,” Ph.D. Dissertation, Carnegie Mellon University, CMUCAD-91-59, 1991.

[32] M. Garey, D. Johnson, “Computers and Intractability: A Guide to the Theory of NP-Completeness,” W. H. Freeman, 1979.

[33] B. Guan, C. Sechen, “Efficient Standard Cell Generation When Diffusion Strapping is Required,” in proc. 1996 CICC, pp. 501–504.

[34] A. Gupta and J. Hayes, “Width Minimization of Two-Dimensional CMOS Cells Using Integer Linear Programming,” in proc. *1996 ICCAD*, pp. 660–667.

[35] A. Gupta and J. Hayes, “CLIP: An Optimizing Layout Generator for Two-Dimensional CMOS Cells,” in proc. *34th DAC*, 1997, pp. 452–455.

[36] A. Gupta and J. Hayes, “Optimal 2-D Cell Layout with Integrated Transistor Folding,” in proc. 1998 ICCAD, pp. 128–135.

[37] M. Guruswamy, R. Maziasz, D. Dulitz, S. Raman, V. Chiluvuri, A. Fernandez and L. Jones, “CELLERITY: A Fully Automatic Layout Synthesis System for Standard Cell Libraries,” in proc. 1997 DAC, pp. 327–332.

[38] L. Gwennap, “Merced Slips to Mid-2000, Delay Jeopardizes Attempt to Gain Performance Lead,” *MicroProcessor Report*, 12(8), June 22, 1998.

[39] S. Hambrusch and H. Tu, “Minimizing Total Wire Length During 1-Dimensional Compac-

tion," INTEGRATION, 14(2), 1992, pp. 113–144.

[40] M. Hanawa, K. Kaneko, T. Kawashimo and H. Maruyama, "A 4.3ns 0.3µm CMOS 54x54b Multiplier Using Precharged Pass-Transistor Logic," in proc. *1996 ISSCC*, p. 265.

[41] N. Hedenstierna and K. Jeppson, "CMOS Circuit Speed and Buffer Optimization," IEEE *Trans. on CAD*, 6(2), March 1987, pp. 270–281.

[42] L. Heller, W. Griffin, J. Davis and N. Thoma, "Cascode Voltage Switch Logic: A Differential CMOS Logic Family," in proc. *1984 ISSCC*, p. 17.

[43] Y. Hsieh, C. Huang, Y. Lin and Y. Hsu, "LiB: A CMOS Cell Compiler," *IEEE Trans. on CAD*, 10(8), August 1991, pp. 994–1005.

[44] M. Huang, F. Romeo, and A. Sangiovanni-Vincentelli, "An Efficient General Cooling Schedule for Simulated Annealing," in proc. 1986 *ICCAD*, pp. 381–384.

[45] F. Hwang, "On Steiner Minimal Trees with Rectilinear Distance," *SIAM Journal of Applied Mathematics*, 30(1), Jan. 1976, pp. 104–114.

[46] S. Hustin and A. Sangiovanni-Vincentelli, "TIM, a New Standard Cell Placement Program Based on the Simulated Annealing Algorithm," unpublished Master's Thesis, University of California at Berkeley.

[47] M. Kang, W. Dai, "General floorplanning with L-shaped, T-shaped and soft blocks based on bounded slicing grid structure", in proc. *1997 ASP-DAC*, pp. 267–270.

[48] M Kang, W. Dai, "Arbitrary Rectilinear Block Packing Based on Sequence Pair," in proc. *1998 ICCAD*, pp. 259–266.

[49] J. Keller, "The 21264: A Superscalar Alpha Processor with Out-of-Order Execution," in proc. 1996 MicroProcessor Forum, http://www.digital.com/info/semiconductor/
a264up1/index.html

[50] S. Kirkpatrick, C. Gelatt and M Vecchi, "Optimization by Simulated Annealing," *Scienc*e, 220(4598), May 1983, pp. 671–680.

[51] F. Klass, "Semi-Dynamic and Dynamic Flip-Flops with Embedded Logic," in proc. *1998 Symp. on VLSI circ.*, p. 109.

[52] J. Kleinhaus, G. Sigl, F. Johannes, and K. Antreich, “GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization,” IEEE Trans. on CAD, 10(3), March 1991, pp. 356–365.

[53] J. Kowaleski, G. Wolrich, T. Fischer, R. Dupcak, P. Kroesen, T. Pham and A. Olesin, “A Dual-Execution Pipelined Floating-Point CMOS Processor”, in proc. *1996 ISSCC*, p. 359.

[54] J. Kruskal, “On the Shortest Spanning Subtree of a Graph and the Travelling Salesman Problem,” *Proc. of the American Mathematical Society*, 7(1), 1956, pp. 48–50.

[55] G. Lakhani and R. Varadarajan, “A Wire-Length Minimization Algorithm for Circuit Layout Compaction,” in proc. 1987 ISCAS, pp. 276–279.

[56] J. Lam and J. Delosme, “An Efficient Simulated Annealing Schedule: Derivation,” Yale University, New Haven, Connecticut, *Technical Report 8816*, Sept. 1988.

[57] J. Lam and J. Delosme, “An Efficient Simulated Annealing Schedule: Implementation and Evaluation,” Yale University, New Haven, Connecticut, *Technical Report 8817*, Sept. 1988.

[58] K. Lampaert, G. Gielen, and W. Sansen, “A Performance-Driven Placement Tool for Analog Integrated Circuits,” *IEEE J. Solid State Circ.*, 30(7), July 1995, pp. 773–780.

[59] E. Lawler, “Combinatorial Optimization: Networks and Matroids,” Holt, Reinhart and Winston, 1976.

[60] M. Lefebvre and C. Chan, “Optimal Ordering of Gate Signals in CMOS Complex Gates,” in proc. 1989 CICC, pp. 17.5.1–17.5.4.

[61] M. Lefebvre, C. Chan and G. Martin, “Transistor Placement and Interconnect Algorithms for Leaf Cell Synthesis,” in proc. *1990 European DAC*, pp. 119–123.

[62] M. Lefebvre and C. Liem, “Cell Generator-Based Technology Mapping by Constructive Tree-Matching and Dynamic Covering,” VLSI Design, 3(1), 1995, pp. 1–12.

[63] M. Lefebvre, D. Marple and C. Sechen, “The Future of Custom Cell Generation in Physical Synthesis,” in proc. *1997 Design Automation Conference*, pp. 446–451.

[64] M. Lefebvre and D. Skoll, “PicassoII: A CMOS Leaf Cell Synthesis System,” in proc. *1992 MCNC Intl. Workshop on Layout Synth.*, vol. 2, pp. 207–219.

[65] T. Lengauer, “Combinatorial Algorithms for Integrated Circuit Layout,” John Wiley and Sons, Chichester, 1990.

[66] F. Lu and H. Samueli, “A 200-MHz CMOS Pipelined Multiplier-Accumulator Using a Quasi-Domino Dynamic Full-Adder Cell Design,” *IEEE J. Solid State Circ.*, 28(2) Feb. 1993, p. 125.

[67] H. Makino, H. Suzuki, H. Morinaka, Y. Nakase, K. Mashiko and T. Sumi, “A 286 MHz 64-b Floating Point Multiplier with Enhanced CG Operation,” *IEEE J. Solid State Circ.*, 31(4), Apr. 1996, p. 510.

[68] C. Makris and C. Toumazou, “Analog IC Design Automation: Part II—Automated Circuit Correction by Qualitative Reasoning,” *IEEE Trans. on CAD*, 14(2), Feb. 1995, pp. 239–254.

[69] E. Malavasi and D. Pandini, “Optimum CMOS Stack Generation with Analog Constraints,” *IEEE Trans. on CAD*, 14(1), Jan. 1995, pp. 107–122.

[70] M. Matsui, H. Hara, Y. Uetani, L.-S. Kim, T. Nagamatsu, Y. Watanabi, A. Chiba, K. Matsuda and T. Sakurai, “A 200 MHz 13 mm^2 2-D DCT Macrocell Using Sense-Amplifying Pipeline Flip-Flop Scheme, “*IEEE J. Solid State Circ.*, 29(12), Dec. 1994, p. 1484.

[71] R. Maziasz and J. Hayes, “Layout Optimization of Static CMOS Functional Cells,” *IEEE Trans. on CAD*, 9(7), July 1990, pp. 708–719.

[72] R. Maziasz and J. Hayes, “Exact Width and Height Minimization of CMOS Cells,” in proc. *28th DAC*, 1991, pp. 487–493.

[73] R. Maziasz and J. Hayes, “Layout Minimization of CMOS Cells,” Kluwer Academic Publishers, Boston, 1992.

[74] C. Mead and L. Conway, “Introduction to VLSI Systems,” Addison-Wesley, Reading, Mass., 1980.

[75] N. Metropolis, A. Rosenbluth, E. Teller and A. Teller, “Equation of State Calculations by Fast Computing Machines,” *Journal of Chemical Physics*, vol. 21, 1953.

[76] J. Montanaro, R. Witek, K. Anne, A. Black, E. Cooper, D. Dobberpuhl, P. Donahue, J. Eno, A. Farell, G. Hoeppner, D. Kruckemyer, T. Lee, P. Lin, L. Madden, D. Murray, M. Pearce, S. Santhanam, K. Snyder, R. Stephany and S. Thierauf, “A 160MHz 32b 0.5W CMOS RISC Microprocessor,” in proc. *1996 ISSCC*, p. 215.

[77] H. Murata, K. Fujiyoshi, S. Nakatake and Y. Kajitani, “Rectangle Packing Based Module Placement,” in proc. *1995 ICCAD*, pp. 472–479.

[78] L. Nagel, “SPICE2: A Computer Program to Simulate Semiconductor Circuits,” Memo no. ERL-M520, Electronics Research Laboratory, University of California, Berkeley, CA, May 1975.

[79] G. Nemhauser, L. Wolsey, “Integer and Combinatorial Optimization,” John Wiley and Sons Publishers, New York, 1988.

[80] N. Nilsson, “Principals of Artificial Intelligence”, McGraw Hill, New York, 1971, chapter 2.

[81] W. Nye, D. Riley, A. Sangiovanni-Vincentelli and A. Tits, “DELIGHT.SPICE: An Optimization-Based System for the Design of Integrated Circuits,” *IEEE Trans. on CAD*, 7(4), April 1988, pp. 501–519.

[82] E. Ochotta, “Synthesis of High-Performance Analog Cells in ASTRX/OBLX,” Ph.D dissertation, *Carnegie Mellon University Research Report CMUCAD-94-17*, Feb. 1994.

[83] E. Ochotta, R. Rutenbar, L. Carley, “Synthesis of High-Performance Analog Circuits in ASTRX/OBLX,” *IEEE Trans. on CAD*, 15(3), March 1996, pp. 273–294.

[84] H. Onodera, Y. Taniguchi, and K. Tamaru, “Branch-and-Bound Placement for Building Block Layout,” in proc. *28th DAC*, 1991, pp. 433–439.

[85] R. Otten, L. Van Ginneken, “The Annealing Algorithm,” Kluwer Academic Publishers, Boston, 1989.

[86] C. Papadimitriou and K. Steiglitz, “Combinatorial Optimization Algorithms and Complexity,” Prentice-Hall, Englewood Cliffs, NJ. 1982, p. 413.

[87] C. Poirier, “Excellerator: Custom CMOS Leaf Cell Layout Generator,” *IEEE Trans. on CAD*, 8(7), July 1989, pp. 744–755.

[88] R. Prim, “Shortest Connection Networks and Some Generalizations,” *Bell System Technical Journal*, 1957.

[89] R. Rogenmoser and Q. Huang, “An 800-MHz 1-μm CMOS Pipelined 8-b Adder Using True Single-Phase Clocked Logic-Flip-Flops’,” *IEEE J. Solid State Circ.*, 31(3), Mar. 1996, p. 405.

[90] J. Rubenstein, P. Penfield and M. Horowitz, “Signal Delay in RC Tree Networks,” *IEEE Trans. on CAD*, 2(7), July 1980, pp. 202–211.

[91] R. Rutenbar, “Simulated Annealing Algorithms: An Overview,” *IEEE Circ. and Dev. Mag.*, January 1989, pp. 19–26.

[92] R. Rutenbar, Carnegie Mellon University, private communication, 1998.

[93] R. Saigal, “Linear Programming: A Modern Integrated Analysis,” Kluwer Academic Publishers, Boston, 1995.

[94] R. Saigal, University of Michigan, Private Communication, 1997.

[95] S. Saika, M. Fukui, N. Shinomiya and T. Akino, “A Two-Dimensional Transistor Placement Algorithm for Cell Synthesis and its Application to Standard Cells”, *IEICE Trans. Fund.*, E80–A(10), Oct. 1997, pp. 1883–1891.

[96] S. Sait and H. Youssef, “VLSI Physical Design Automation, Theory and Practice,” McGraw Hill Book Company, London, 1995.

[97] S. Sapatnekar, V. Rao, P. Vaidya and S.-M. Kang, “An Exact Solution to the Transistor Sizing Problem for CMOS Circuits Using Convex Optimization,” *IEEE Trans. on CAD*, 12(11), Nov. 1993, pp. 1621–1643.

[98] L. Scales, “Introduction to Non-Linear Optimization,” Springer-Verlag, New York, 1985.

[99] W. Schiele, “Improved Compaction by Minimized Length of Wires,” in proc. 20th DAC, 1983, pp. 121–127.

[100] M. Schlag, Y. Liao and C. Wong, “An Algorithm for Optimal Two-Dimensional Compaction of VLSI Layouts,” *Integration* 1(2,3), 1983, 179-209.

[101] C. Sechen and A. Sangiovanni-Vincentelli, “The TimberWolf Placement and Routing Package”, *IEEE J. Solid State Circ.*, SC-20(2), Apr. 1985, pp. 510–522.

[102] C. Sechen and A. Sangiovanni-Vincentelli, “TimberWolf 3.2: A New Standard Cell Placement and Global Routing Package,” in proc. *23rd DAC*, 1986, pp. 432–439.

[103] C. Sechen, “Chip-Planning, Placement, and Global Routing of Macro/Custom Cell Integrated Circuits using Simulated Annealing,” in proc. *25th DAC*, 1988, pp. 73–80.

[104]Semiconductor Industry Assoc., “The National Technology Roadmap for Semiconductors”, SEMATECH Inc., 1997, http://www.sematech.org/public/roadmap/index.htm.

[105]A. Shams and M. Bayoumi, “A New Full Adder Cell for Low-Power Applications,” in proc. *1998 Great Lakes Symp. on VLSI*, p. 47.

[106]N. Sherwani, “Algorithms for VLSI Physical Design Automation,” Kluwer Academic Publishers, Boston, 1993.

[107]Y. Shimazu, T. Kengaku, T. Fujiyama, E. Teraoka, T. Ohno, T. Tokuda, O. Tomisawa and S. Tsujimichi, “A 50MHz 24b Floating-Point DSP,” in proc. *1989 ISSCC*, p. 45.

[108]J. Shyu, A. Sangiovanni-Vincentelli, J. Fishburn and A. Dunlop, “Optimization-Based Transistor Sizing,” *IEEE J. Solid State Circ.*, 23(2), Apr. 1988, pp. 400–409.

[109]J. Silberman, N. Aoki, D. Boerstler, J. Burns, S. Dhong, A. Essbaum, U. Ghosal, D. Heidel, P. Hofstee, K. Lee, D. Meltzer, H. Ngo, K. Nowka, S. Posluszny, O. Takahashi, I. Vo and B. Zoric, “A 1.0GHz Single-Issue 64b PowerPC Integer Processor,” in proc. *1998 ISSCC*, p. 231.

[110]D. Somasekhar and K. Roy, “Differential Current Switch Logic: A Low Power DCVS Logic Family,” *IEEE J. Solid State Circ.*, 31(7), July 1996, p. 987.

[111]R. Statnikov and J. Matusov, “Multicriteria Optimization and Engineering,” Chapman & Hall, New York, 1995.

[112]R. Stephany, K. Anne, J. Bell, G. Cheney, J. Eno, G. Hoeppner, G. Joe, R. Kaye, J. Lear, T. Litch, J. Meyer, J. Montanaro, K. Patton, T. Pham, R. Reis, M. Silla, J. Slaton, K. Snyder and R. Witek, “A 200MHz 32b 0.5W CMOS RISC Microprocessor,” in proc. *1998 ISSCC*, p. 239.

[113]S. Stetson, University of Michigan, private communication, 1997.

[114]L. Stockmeyer, “Optimal Orientations of Cells in Slicing Floorplan Designs,” Information and Control, vol. 57, pp. 91-101, 1983.

[115]S. Sutanthavibul, E. Shragowitz and J. Rosen, “An Analytical Approach to Floorplan Design and Optimization,” in proc. *27th DAC*, 1990, pp. 187–192.

[116]M. Suzuki, N. Ohkubo, T. Yamanaka, A. Shimizu and K. Sasaki, “A 1.5ns 32b CMOS ALU in Double Pass-Transistor Logic,” in proc. *1993 ISSCC*, p. 91.

[117]K. Tani, K. Izumi, M. Kashimura, T. Matsuda and T. Fujii, “Two-Dimensional Layout Synthesis for Large-Scale CMOS Circuits”, in proc. *1991 ICCAD*, pp. 490–493.

[118]C. Toumazou and C. Makris, “Analog IC Design Automation: Part I—Automated Circuit Generation: New Concepts and Methods,” *IEEE Trans. on CAD*, 14(2), Feb. 1995, pp. 218–238.

[119]T. Uehara and W.M. VanCleemput, “Optimal Layout of CMOS Functional Arrays,” *IEEE Transactions on Computers*, C-30(5), May 1981, pp. 305–312.

[120]G. Vijayan and C. Cheng, “A New Method for Floorplanning Using Topological Constraint Reduction,” IEEE. Trans on CAD, 10(12), Dec. 1991, pp. 1494–1501.

[121]C. Visweswariah, “Optimization Techniques for High-Performance Digital Circuits,” in proc. *1997 ICCAD*, pp. 198–205.

[122]D.Wang and E. Kuh, “Performance-Driven Interconnect Global Routing,” in proc. *The Sixth Great Lakes Symposium on VLSI*, 1996, pp. 132-136.

[123]T. Wang and D. Wong, “An Optimal Algorithm for Floorplan Optimization,” in proc. 27th DAC, 1990, pp. 180–186.

[124]Z. Wang, G. Jullien, W. Miller, J. Wang and S. Bizzan, “Fast Adders Using Enhanced Multiple-Output Domino Logic,” *IEEE J. Solid State Circ.*, 32(2) Feb. 1997, p. 209.

[125]S. White, “Concepts of Scale in Simulated Annealing,” in proc. 1984 *ICCD*, pp. 646–651.

[126]D. Wong and C. Liu, “A New Algorithm for Floorplan Design,” in proc. *23rd DAC*, 1986, pp. 101–107.

[127]H. Xia, M. Lefebvre and D. Vinke, “Optimization-Based Placement Algorithm for BiCMOS Leaf Cell Generation”, *IEEE J. Solid State Circ.*, 29(10), Oct. 1994, pp. 1227–1237.

[128]K. Yano, Y. Sasaki, K. Rikino and K. Seki, “Top-Down Pass-Transistor Logic Design,” *IEEE J. Solid State Circ.*, 31(6) June 1996, p. 797.