ANALOG LAYOUT GENERATION

by

Ender YILMAZ

B.S., Electronical Engineering, Yeditepe University, 2004

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Electronical Engineering

Boğaziçi University

2006

ACKNOWLEDGEMENTS

First of all and the foremost, I wish to thank to my family, for their infinite patience, support and love. Besides, I thank them very much for everything they sacrificed for me.

I am very much grateful of working with Prof. Günhan Dündar, whom always encouraged me during this project. He always appreciated every little progress have achieved. I would like to express my graduate to him for his trust and guidance through this project.

I would like to thank to Aslı Ünlügedik, expressing my deep appreciation for her suggestions, criticisms and encouragement.

Finally, I would like to thank for the support of TÜBİTAK.

ABSTRACT

ANALOG LAYOUT GENERATION

In this work an analog layout automation tool, ALGv3, is presented. ALGv3 is successor of ALGv2, which was also designed at Bogazici University. ALGv3 is composed of four parts; module generator, partitioner, placer and router.

Module generator is capable of generating simple transistors as well as complex structures such as merged, interdigitized and common centroid transistors. Modules can be generated in a performance oriented manner. Parameters of modules that are extracted in the generator such as parasitic values, matching and aspect ratio values are combined in a cost function. This cost function is used to find the optimum generation parameters such as fold number or quad number for module generator.

Transistors are partitioned into groups in order to ease the job of placer. Well known structures are recognized in the circuit and transistors of the same group are kept together in placement step.

Router is not finished yet, the total currently routes the nodes, but these routes are not converted to actual layers.

ÖZET

ANALOG TÜMDEVRE İÇİN SERİM OLUŞTURUMU

Bu çalışmada, tümdevre serim oluşturum aracı olan ALGv3 sunulmuştur. ALGv3, kendisi gibi Boğaziçi Üniversitesinde geliştirilen ALGv2'nin devamıdır. ALGv3, dürt bölümden oluşmaktadır; modül üretimi, bölüştürme, yerleştirme ve yol atayıcı.

Modül üreticisi, basit tranzistörler oluşturabildiği gibi iç içe girmiş ve ortak merkezli transistor yapılarını da oluşturabilir. Modüller performans kriterleri gözetilerek üretilir. Oluşturulan modullerin parametreleri olan parasit değerler, eşleştirme ve en-boy değerleri bir maliyet fonksiyonunda bir araya getirilmiştir. Bu maliyet fonksiyonu, modüllerin üretim parametreleri olan katlanma sayısı veya iç içe girme sayısının en uygun değerlerini bulmakta kullanılır.

Tranzistörler, yerleştiricinin işini kolaylaştırmak için bölüştürülürler. Dervredeki iyi bilinen yapılar bulunur, bu yapıların tranzistörleri yerleşim aşamasında bir arada tutulur.

Bu aracın yol atayıcı kısmı henüz tamamlanmadı, düğümler birleştiriliyor fakat bu yollar katman seviyesine aktarılmıyor.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS/ABBREVATIONS

ALG	Analog Layout Generator
amp	Amplifier
AR	Aspect Ratio
BW	Bandwidth
C	Capacitance
ccNum	Number of quads
diff	Differential
FDP	Force directed placement
K	Cutset matrice
M	Mismatch
numFold	Number of folds
numQuad	Number of quads
NP	Non polynomial
λ	Minimum width in layout

1. INTRODUCTION

In modern technology, systems are getting larger and more complicated; many modules are placed in single chips with full functionality. The reason for this development is cost and productivity, the smaller the chip area, the more easily producable and cheaper it is. It is easier to design large digital systems that compared to their analog counterparts, so digital circuits dominate the market. However, that doesn't mean that analog design is needless, it is an unavoidable part of technology.

Analog design is unavoidable due to two reasons; nature is analog, at least interfaces between digital world and real world is analog. Further more high performance requirements dictate that some subsystems be designed in the analog domain. Analog design has some difficulties as given below.

Difficulties in analog design

- There are too many variables and also constraints to satisfy. These variables and even constraints are not mutually exclusive, they interact with each other.
- Design methodology is very poor compared to digital design
- Requires more qualified engineers
- Design and test time is long
- Mathematical modeling of digital design is plain and simple, but for analog it is complicated, and evolving with technology

Despite these difficulties, analog design is an inevitable part of a system. Almost all systems include an analog part, which means most systems require an analog design step. Analog design is done in a full custom manner, where every part of the layout is designed by hand, hence requiring expert knowledge for even simple circuits. If the design is a high performance circuit, design of the analog step dominates all other design steps.

Analog VLSI design automation is essential for time and labor efficiency. Analog design needs a systematic approach and automation; using these two concepts, huge

savings in labor can be achieved. One of the reasons for the relatively more rapid evolution of digital design is the existence of automation tools.

1.1. Analog VLSI Design Automation

Digital design is highly suitable for automation; problem specification is not difficult to state, there are well known steps that are repeated in every design and in general these steps do not differ too much. These steps and their order are illustrated on the well known Y-chart [33].

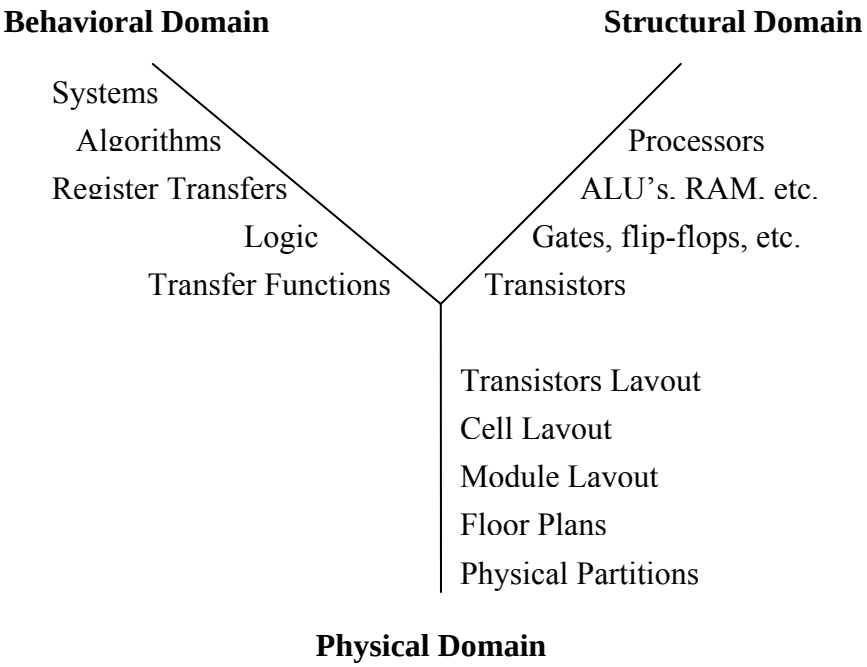


Figure 1.1. Y-Chart

Analog design also needs a similar approach in order to be automated. Circuit design is a long and error prone step, where design time and errors increase with complexity. Fortunately, in analog design there are also steps that are common for various designs. This enables us to define a chart similar to the Y-chart for analog design. There are several analog automation flows in contrast to digital automation [1][21]. One of the

flows [1] is shown in Figure 1.2. Dividing a big problem in small pieces eases the work of engineers, increasing productivity and decreasing design time.

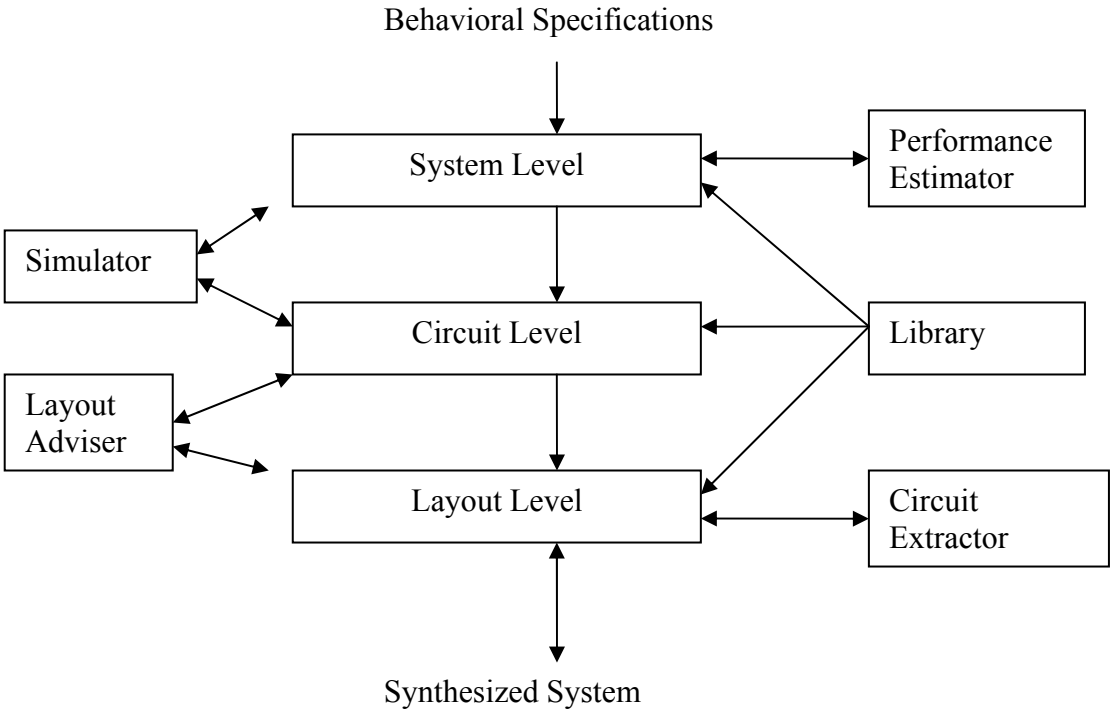


Figure 1.2. Flow of analog design

System level synthesis is the first step of automation. Behavioral specifications are accepted as inputs. These inputs may differ from design to design. For example while an amplifier requires BW, gain, slew rate parameters, a filter requires frequency specifications. Behavioral specifications are used to obtain a block diagram of the system. Performance estimation and simulation steps are used to select best building blocks and topologies. Blocks and their specifications are processed on the next step, circuit level synthesis.

Circuit level synthesis step accepts the outputs of the previous step as input. Output of the previous stage is building blocks of the system and their topologies. These inputs are optimized to give the netlist of each block and consequently of the whole system, where all transistors are sized and the netlist is complete. Most popular optimization algorithms used in this step are simulated annealing, genetic algorithms and simulated evolution

algorithms. These algorithms may differ in terms of performance from circuit to circuit. Parameters of these algorithms may also differ for each circuit.

Layout level synthesis is the last step. Spice file and performance specifications are accepted as input. Output of this step is the layout of the circuit. Layout level synthesis is hard to automate; design steps may differ for each implementation. One of the design flows is shown below. These steps do not occur successively, instead they interact with each other either to inform or to manipulate the related step.

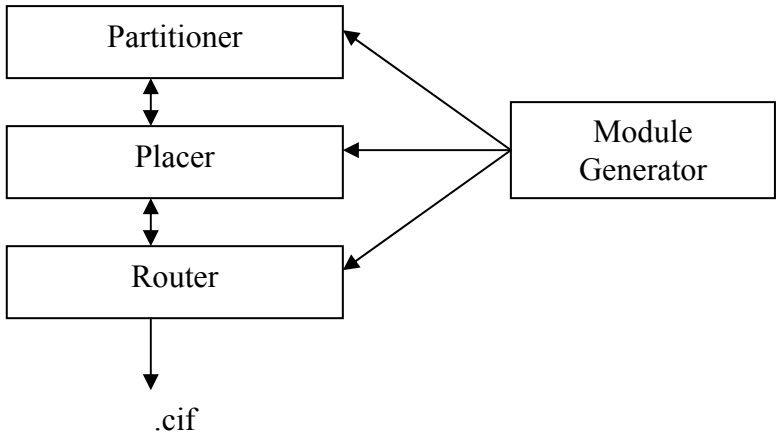


Figure 1.3. Flow of analog layout automation

1. module generator : transistors, capacitors and resistors are generated here
2. partitioner : modules are partitioned according to performance specifications. This stage is used to ease the job of the following stage.
3. placer: modules are placed using performance data from partitioner and layout adviser. Placer uses an optimization algorithm to place the modules.
4. router: placed modules are electrically connected at this step. There are many constraints to be satisfied such as coupling, parasitics, performance criteria.

ALG is a layout level synthesis tool designed at Boğaziçi University. ALGv2 uses the design flow shown in Figure 1.3. It is capable of generating layout automatically as well as admitting control of the user.

The work presented here (ALGv3) is an enhanced version of ALGv2. ALGv3 adds some features such as performance oriented layout generation, which will be discussed on following sections.

1.2. Background – Block(Macrocell) Layout Implementations

A classification of analog layout automation tools [2] is shown below:

1.2.1. Procedural Tools

These tools support only specific circuit topologies. Manually designed topologies are modified by passing arguments. Sizes of elements can be changed; also positions of the elements are changed accordingly. This style of implementation gives reasonable but limited results. Maintainability of procedural tools is very hard.

1.2.2. Template Based Tools

These tools capture layouts from manually designed circuits. This method enables the use of knowledge, but is also limited in its capabilities.

1.2.3. Rule Based Tools

Rule based systems can produce reasonable layouts in short time. All rules are given by designer, so layout depends not on the tool, but on the given rule set. Quality of the design is rule set dependent.

1.2.4 Algorithmic Tools

The approaches discussed above depicted strongly rely on human effort. Another option is the algorithmic approach. Partitioning, placement, routing and even module generation uses algorithms to minimize a cost function. Minimization of the cost function also optimizes the system, where cost function is composed of penalties that violate design constraints. The tool makes the distinction between bad and good solutions via a cost function; it also changes variables, such as orientation, position and distance. There are several algorithmic approaches divided in two groups; heuristic and performance oriented. The resulting layout depends on the algorithm and also on the topology at the circuit.

1.2.4.1 Heuristic Based Algorithmic Tools

Heuristic algorithms add some extra constraints to improve the quality of layout. These constraints are symmetry, matching, signal decoupling and electrical constraints. Heuristics are useful in reduction of complexity of the tool but they do not guarantee the best result.

1.2.4.2 Algorithmic Tools

Rule set and cost function in a heuristic algorithm are not directly related to performance of the circuit designed. Including performance information in the cost function or making the cost function performance dependent makes the algorithm performance oriented.

Approaches described above are methods of layout automation. That doesn't imply that a layout automation tool uses only one of these approaches. A tool may employ a combination of these approaches. There are many automation tools in literature, some of them will be reviewed below.

1.3. Previous Automation Tools

ALAS! [11] is a rule based tool that can generate common centroid and interdigitized modules, which are the most important feature of ALAS!. Passive components are also generated. ALAS! also has a simple routing and placing tool.

Two most important features of DTA [22] is matched device generation and reduction of noise coupling. Simulated annealing and genetic algorithms are used as placement algorithms. Device generator of DTA is capable of matching and stacking the transistors. Placer uses minimum wire and area constraints, while router minimizes noise coupling. DTA is a heuristic tool.

Aladin [18] is a powerful heuristic tool for layout generation. Module generator of Aladin can generate complex structures in order to improve matching. Aladin uses a description language, MOGLAN, which enhances the capability of module generator; it can be extended to support other technologies. Genetic placement approach with simulated annealing control is used as placement algorithm. Cost function is composed of area and net length terms. Channel intersection graph model is used for routing.

KOAN/ANAGRAM II [13] works in heuristic manner. These tools have procedural device generators, dynamic merging and abutment of devices while placing, over-the-device routing symmetric routing and crosstalk avoiding capabilities. KOAN is a placer tool, which employs simulated annealing. KOAN uses flat, non slicing annealing model. Three types of constraints are maintained during placement: symmetry, matching and topological constraints. Cost function includes overlap, area, aspect ratio, net length, proximity and merging parameter. ANAGRAM II is the router tool. Line-expansion routing technique is used. Routing is optimized using another cost function which includes wire length, direction, cross-talk and rip-up parameters. Features of router are over-the-device routing, symmetric routing, crosstalk avoiding and integrated rip-up/rerouting.

ALSYN [15] is a rule based tool; modules are recognized, placed and routed using user specified rules. Placement algorithm is based on min-cut approach and uses

Stockmeyer's algortihm for area optimization. Partial slicing, symmetry and clustering groups can be controlled with user control. Routing can be done with grid based maze routing or with gridless line expansion approach.

STAIC [25] generates rule based layouts for both CMOS and BiCMOS technologies. Two inputs are accepted; performance specifications and hierarchal circuit description.

Ballistic [24] is a high-level language used for analog layout description. Analog circuits are described hierarchally using building blocks such as transistors, differential pairs, current mirrors and capacitor arrays. Building blocks are fully parameterized and technology-independent.

SALIM [23] is both heuristic and rule based, it has two operation modes; automatic and manual. Automatic mode uses heuristic based algorithm, manual mode enables user to set the rules. The objective of the optimization algorithm is area efficiency, symmetry, reduction of parasitic values at some critical nodes, and avoiding parasitic coupling.

ILAC [17] and IDAC [16] are complete set of tools that can generate a geometrical layout from functional specifications. ILAC is the layout automation part of the system, it is a heuristic tool. Typical layout constraints; matching, symmetry, distance and coupling constraints are handled. ILAC uses block place and route approach, while blocks can be simple transistors or combinations of simple transistors (Common Centroid, Current Mirror). Produced blocks can have a number of different realizations; these are rectangular, snake, interdigitized, circular, concentric, S-shaped and waffle. ILAC supports three different ways of block generation: STUCCO, LISA and CIF file interface.

- STUCCO is a procedural block layout generator for commonly used blocks.
- LISA is an interactive layout tool with graphics. It is also technology independent.

Placement and routing is done by MOSAIC. Placement algorithm uses slicing structures with simulated annealing. Cost function of placer contains cell area, center-point distance between connected blocks and penalties of constraint violations.

Routing algorithm is best-first search based, it handles new sensitivities, planarity and channel congestions.

Final step is compaction and geometrical post processing, which is the job of PICTURE. PICTURE converts process independent output of MOSAIC to process-specific layers.

2. MODULE GENERATOR

2.1. Introduction

Module generators are the factories of layout automation tools. They produce element for layout with desired properties like matching and symmetry. A flexible and powerful module generator boosts capabilities of a layout automation tool. Generator has interaction with other parts of automation tool in every step; size, parasitic values and electrical properties of the module are supplied to the related step as shown in Figure 2.1.

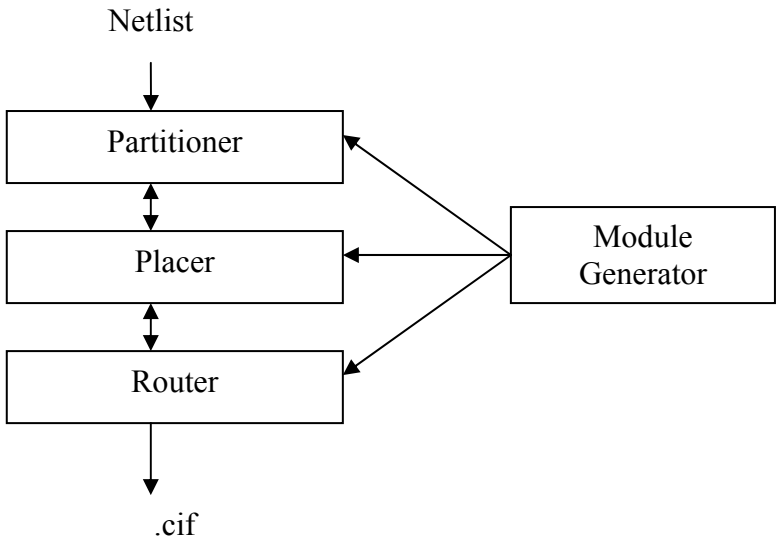


Figure 2.1. The layout automation flow

A module generator can be used as a separate tool as well as a component in an automation tool. The latter is more beneficial because interaction of module generator with other components such as partitioner and placer may drive the solution to a better point. Interactions are not one way, module generator and placer give feedback to each other in order to increase the quality of the layout. All steps in the automation tool have a connection with the module generator.

Module generators are used to generate basic building blocks of the circuit like transistors, resistors, and capacitors. Simple elements are easy to generate; however, sophisticated structures (interdigitized, common centroid) are hard to design and modify. Module generators can generate any kind and any amount of the desired element.

Common centroid (Quad), interdigitized and merged transistors are three complex structures that can be generated by the module generator, details will be described in the following pages.

2.2. Module Generator of ALG

Generated modules can be controlled either by user specified directives, or by performance criteria. Fold number, quad number, transistor width and length features are manually controllable, giving the designer freedom of producing any kind of layout.

One of the major features of the generator is its performance oriented module generation capability. The module is optimized according to a cost function to fit in the layout and to perform in the best way. The cost function includes mismatch, parasitics, and aspect ratio terms.

2.2.1. Mismatch

Achieving perfect values of resistors, capacitors and transistors is impossible in practice. Absolute values of these properties may have ±10% variation. In some critical cases, exact ratios are needed, these elements are generated by matching resistors, capacitors and transistors. Matching techniques can restrict variation down to ±0.1%. Interdigitized, and common centroid structures are two examples for matching techniques.

There are some models of mismatch in the literature [8][9], but it not realistic to model mismatch using an absolute model without the manufacturer's process variation parameters. Instead of a real model, a relative matching model is implemented here.

Common centroid is advantageous compared with interdigitized structure in terms of mismatch, because common centroid is point symmetric but interdigitized is only line symmetric. Devices are not generated only regarding matching and parasitic effects, aspect ratio is also taken in account.

2.2.2. Parasitics

Parasitics in this work are unwanted capacitors and resistors that degrade the performance of the circuit. Layout has a crucial role on performance of the circuit, even if circuit level design works fine, layout level design may not work properly. Parasitics are everywhere; they are in the channel, between the wires …. But not all of them are important; reducing only critical ones is enough. Designing layout with minimum critical parasitics makes the process performance oriented. Parasitics can be reduced in several ways:

- Stacking : abutting transistors reduces space and capacitance
- Generating compact layouts: wires introduce both capacitance and resistance, making layout more compact reduces wire length and parasitics also.
- Using fewer contacts: every via introduces a resistance, and decreases reliability.

2.2.3. Aspect Ratio (AR)

Aspect ratio is another parameter for device generation. Near unity layouts give higher yield and make layout more compact. Improving matching and parasitic effects also improves aspect ratio.

2.2.4. Cost Function

Combining all these effects in a cost function and weighting them with coefficients enables us to find the optimum solution. Coefficients are very important, they are selected to give the optimum result for a given application.

Before combining these three terms, they are normalized. Resistance is on the order of 10^{-3} , capacitance : 10^{-15} , aspect ratio : 10^{0} , mismatch : 10^{0}-10^{-6}. Capacitance and resistance are normalized with respect to their minimum values, mismatch is normalized with respect to it maximum value. Structure of the cost function is given below; details are discussed in following section.

$$\text{Cost Function} = \alpha_1 \,.\, C_{total} + \alpha_2 \,.\, \text{Mismatch} + \alpha_3 \,.\, \text{AR} \qquad (2.1)$$

2.3. Transistor Generation

Module generator is capable of generating simple structures (unfolded) transistors as well as complex structures (interdigitized, common centroid, merged). Supported module generation styles are:

- SimpleMOS
- FoldedMOS
- Merged
- Interdigitized – common source
- CommonCentroid(Quad) – common source

None of these structures can be generalized to be the best choice, size, area, matching and parasitic constraints for a transistor are used to select one. For example; a big transistor generated with simpleMos structure will have a very high aspect ratio, which is not acceptable, also matching between simple mos transistors is poor. Common centroid is the

best choice for matching purposes, but folding transistors too much increases capacitance. This structure is not convenient for small transistors.

2.3.1. Simple MOS

SimpleMOS is the simplest structure. Two diff contact array, a diffusion and a poly layer is used to construct simple mos.

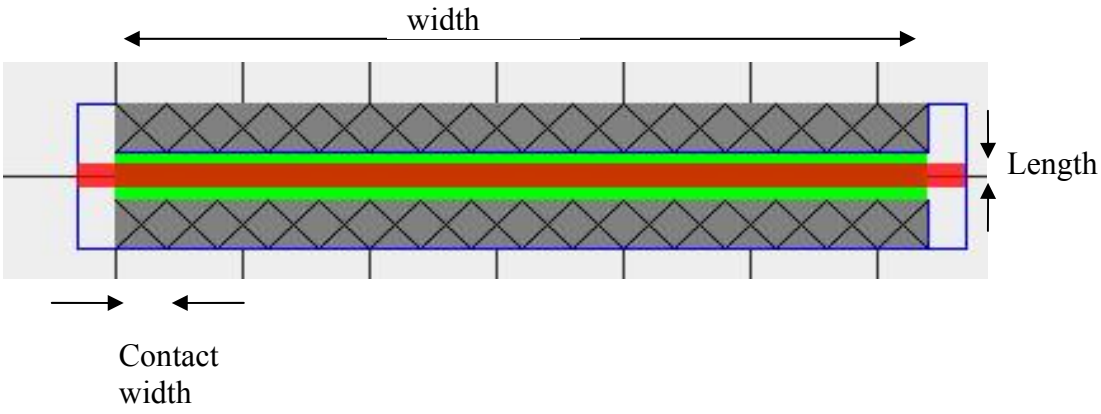


Figure 2.2. Simple MOS layout

Advantage of this type is its simplicity. It is convenient for small and unmatched transistors. Also no internal routing is needed. On the other hand it has some disadvantages; its perimeter to area ratio is high, so diffusion capacitance is high, matching is poor and aspect ratio is too high for big transistors.

Models used for simpleMOS is given below.

Capacitance Model:

$$C_{diff} = C_{diffArea} + C_{diffPerimeter} \quad (2.2)$$

Area is all diffusion area, perimeter is length of light gray line in Figure 2.3

.

$$C_{diffArea} = numberOfContacts \; . \; CapacitancePerContact \quad (2.3)$$

$$C_{diffPerimeter} = (numberOfContacts+2) \; . \; CapacitancePerContactSide \quad (2.4)$$

$$C_{drain} = C_{source} = C_{diff} \tag{2.5}$$

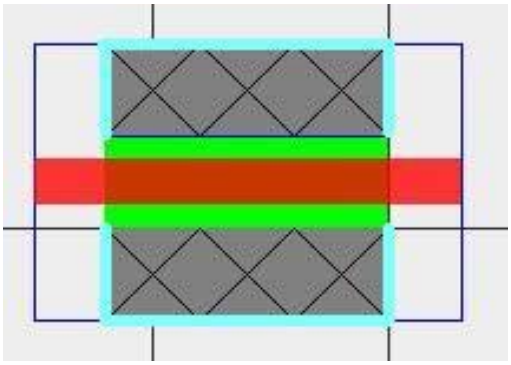


Figure 2.3. Active sides of simple MOS

Mismatch:

Since there is no matching with other transistors, matching model does not exist for this structure.

Aspect Ratio:

$$\text{width} = \text{min\_contact\_width}$$
$$\text{height} = 2 \,.\, \text{contactWidth} + 2 \,.\, \text{contact\_poly\_spacing} + \text{poly\_width(length)} \tag{2.6}$$

$$AR = \frac{\max(height, width)}{\min(height, width)} \tag{2.7}$$

Electrical Model:

Module generator also generates a layout, which shows electrical properties of transistor. This feature is designed for routing stage. Sides of the transistors are electrically active regions, they are routable. Note that only outside looking sides of contacts are free to route. Only contacts, poly, and metal are electrically active, also not all sides are active;

only routable sides are marked as such. This electrical layout provides detailed information to the router. When a contact is routed at one of its active sides, that side or a part of that side is marked as inactive. Sides of metal and poly layers are also marked active.

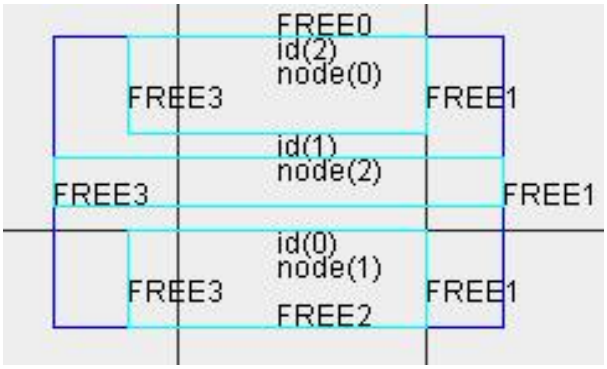


Figure 2.4. Electrical layer of simple MOS

FREE0 means top side is routable, FREE1 : right, FREE2 : bottom and FREE3 : left.

2.3.2. Folded MOS

Folding transistors decreases perimeter capacitance and gives the freedom to adjust aspect ratio. Adjustability of aspect ratio is very important for the placer. Among the structures described above, the only structure that is not adjustable in terms of aspect ratio is simpleMos.

FoldedMos is simply abutted or merged combination of simpleMos transistors. A transistor can be folded until width of the transistor decreases down to one contact width. Number of folds changes the capacitances of nodes; since perimeter capacitance is decreased for inner contacts. If numFold is even, the node that corresponds to outer contacts of the transistor has higher capacitance compared with the other one. Using this property, less capacitive node can be used as performance critical node. This structure is suitable for middle or large sized unmatched transistors.

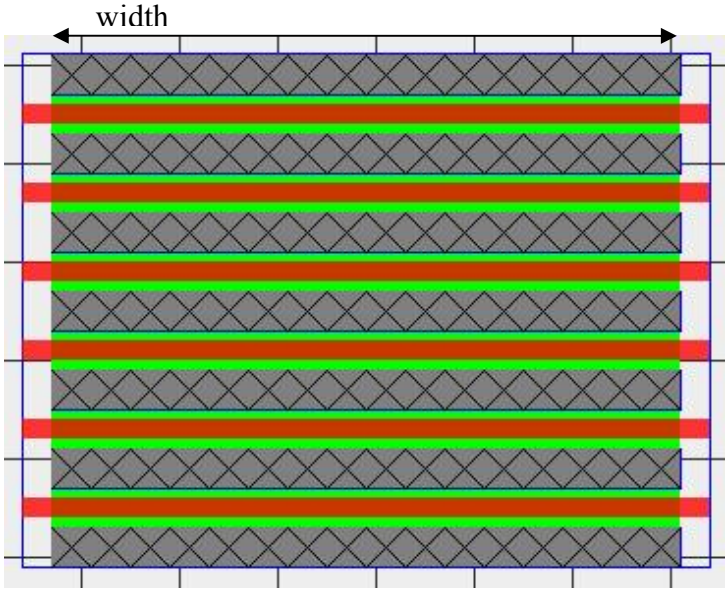


Figure 2.5. Folded MOS layout

Advantages of the foldedMOS are; aspect ratio is adjustable and capacitance decreases due to decrease in diffusion perimeter. Disadvantage is poor matching.

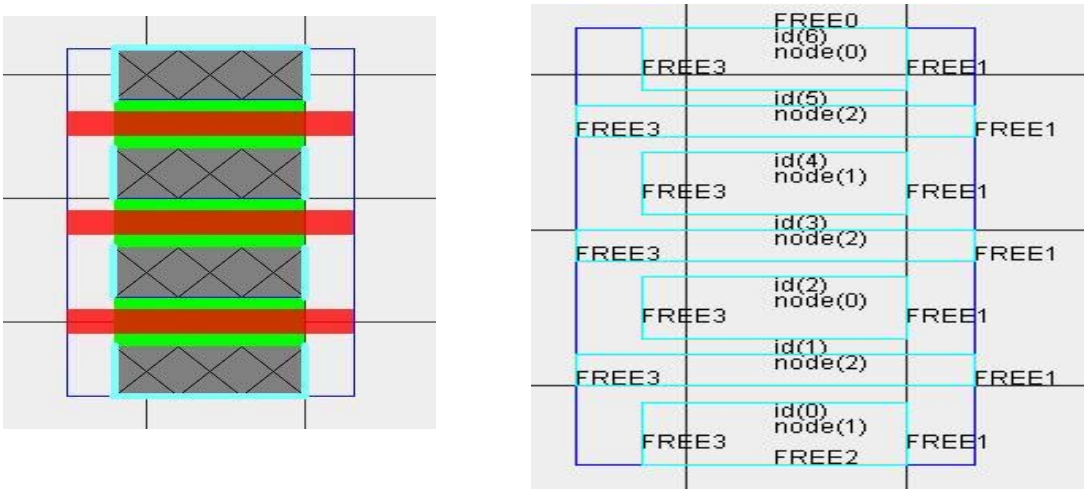
Models used for foldedMOS is given below.

Capacitive Model:

$$C_{diff} = C_{diffArea} + C_{diffPerimeter} \tag{2.8}$$

Capacitance for source and drain differs if folding number is even, one of the nodes appears on both ends of the transistor.

Area capacitance is same as the capacitance model of the simpleMOS, while perimeter capacitance is less. Only light gray sides that are shown in figure 2.6 have contribution to capacitance. Total perimeter is the sum of short sides and two (outermost) long sides. There is also wiring capacitance.



(a) (b)

Figure 2.6. Electrical view of a folded transistor (a) Active sides of folded MOS, (b) Electrical layer of folded MOS

Mismatch:

Mismatch is the same as simpleMos transistor. Mismatch decreases by reducing the distance between transistors and increasing area of the transistor [8][9]. Instead of "mismatch", "dependency of process variations" is more suitable here. Folding does not change area of the transistor, nothing is changed in terms of process variations.

Aspect Ratio:

$$\text{foldWidth} = \text{width} / \text{numFolds} \tag{2.9}$$

$$\text{foldHeight} = \text{numFolds} \cdot (\text{contactWidth} + \text{contact\_poly\_spacing} + \text{poly\_width}) + \text{contactWidth} \tag{2.10}$$

$$AR = \frac{\max(foldWidth, foldHeight)}{\min(foldWidth, foldHeight)} \tag{2.11}$$

2.3.3. Merged Transistors (Stacking)

Merged transistors are generated by abutting two transistors that share one of their drain or source nodes. Geometry sharing affects both layout area and parasitics. Abutted transistors occupy less space, so layout becomes more compact and routing between transistors is not needed any more. Most important outcome of device merging is reduction of capacitance which also boosts performance.

Merging devices not only reduces perimeter capacitance as in foldedMos structure, area capacitance and possible routing capacitance is also reduced. Merging devices has a very important role in performance optimization, because it is the most efficient and easy way of capacitance reduction.

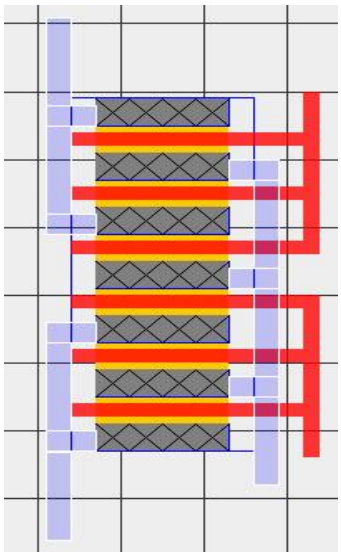


Figure 2.7. Merged MOS transistors

Advantages of merging are capacitance reduction and area optimization. Disadvantage is poor matching.

Models for merged transistor structure are given below.

Capacitive Model:

Capacitance calculation is the same as foldedMos structure calculation, the only difference is that one area capacitance, one long and two short side capacitances are reduced from related nodes.

Mismatch:

Mismatch is poor but better than two separate transistors.

Aspect Ratio:

$$\text{height} = \text{heightOfTransistor1} + \text{heightOfTransistor2} \tag{2.12}$$

$$\text{width} = \text{commonWidthOfTransistors} \tag{2.13}$$

$$AR = \frac{\max(width, height)}{\min(width, height)} \tag{2.14}$$

2.3.4. Interdigitized Transistors

This is the first mismatch reducing structure discussed so far, it is not as good as the following common centroid structure but better than previous ones in terms of matching. Interdigitization is to distribution of a transistor into another, similar to game cards. This technique relies on the fact that reducing distance between transistors reduces mismatch [8][9]. Distribution technique reduces the mismatch. However, there is a limit to distribution; a transistor cannot be divided to have less than one contact. Interdigitizing is similar to folding transistors and placing those folds between folds of other transistors.

Interdigitization is the best suitable for middle sized transistors, because small transistors do not need mismatch reduction, big transistors are better to generate using common centroid technique. Two or more transistors can be interdigitized, but only two transistor interdigitization is supported in module generator of ALGv3.

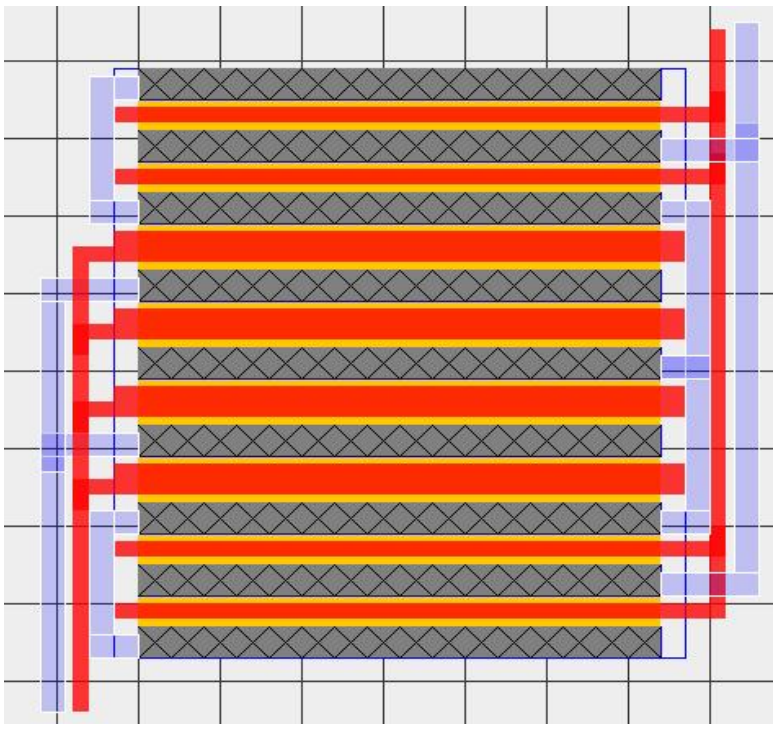


Figure 2.8. Interdigitized MOS transistors

2.3.4.1. Distribution algorithm

Two transistors, say A and B are folded so that every fold has the same width. If width of A is 50 and width of B is 50, A is folded in 5 and B in 5 for example. Representing these folds with corresponding transistors they become AAAAA and BBBBB. Conventional distribution technique is to distribute these transistors one after another: ABABABABAB. This is easy and reasonable but when transistor sizes are different, say 140 and 50 a conventional algorithm cannot interdigitize that easily.

Distribution technique implemented in the module generator of ALGv3 uses another algorithm which easily handles different sized transistors problem with less effort.

Flow is shown below:

1. put folds of each transistor in a list
2. sort the list so that transistors with maximum folds is the first
3. select next transistor in the list
4. ratio = numberOfFoldsOfTransistor / numberOfFoldsOfNextTransistorInTheList
5. if number of folds of transistor is 1 insert it in the middle and exit else move Integer[ratio].2 transistors in the middle of interdigitized transistors.
6. goto 3

Transistors that are folded in 4 will be distributed as such:

AAAA + BBBB → BABAABAB

4 folds and 5 folds will be:

BBBB + AAAAA → BABAAABAB

10 folds and 3 folds will be:

BBB + AAAAAAAAAA → AAABAABAABAAA

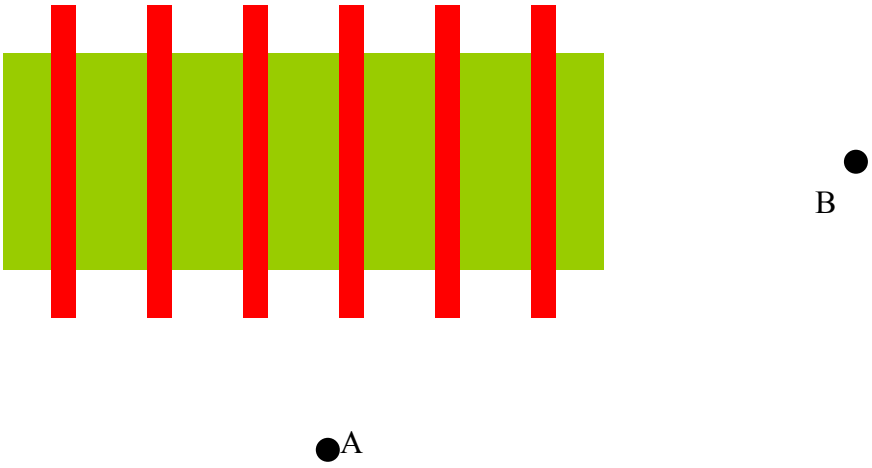


Figure 2.9. Process variation

As shown in Figure 2.9 if process variation is modeled to be a source of variation, which is the point of the symmetry, the interdigitized structure may have problems. When the source of variation is A, there is no problem. But if the source of variation is B, this structure is not as efficient any more.

Interdigitized structure is composed of many merged A and B transistors. Capacitance reduction advantage is inherited from merging. Advantages of interdigitized structure are; capacitance reduction, mismatch reduction (line symmetric) and area optimization.

Another advantage of more-than-one device structures is decrease in element number. Two merging transistors yields a single module, interdigitizing N transistors yields a single complex module. This is especially important because the complexity of design is NP complete and reduction of module numbers makes a huge impact on computation time.

Models used for this structure are given below.

Capacitive Model:

$$C_{drain1} = C_{drain2} = C_{diff} + C_{wire} \quad (2.15)$$

$$C_{source} = C_{diff} + C_{wire} \quad (2.16)$$

$$C_{total} = C_{drain1} + C_{drain2} + C_{source} \quad (2.17)$$

Since there is a lot of merging, a lot of capacitance is reduced. Source nodes as well as drain nodes are merged. Interdigitizing devices reduces diffusion capacitance but at the same time increases wiring capacitance. There is tradeoff between mismatch and parasitics. Too many folds may have an unwanted effect on the performance of the circuit.

Mismatch:

Mismatch model used in ALGv3 is relative. Matching increases proportional to square of folds count. More folds increases matching and decreases process variations.

$$Mismatch = 1/numFolds^2 \quad (2.18)$$

Aspect Ratio:

Height of the structure is proportional to number of folds and width is inversely proportional to number of folds. Aspect ratio can be adjusted changing number of folds.

2.3.5. Common Centroid(Quad)

Quad is the most complex structure; it is used for big transistors and/or mismatch critical transistors. This technique is the best way of reducing mismatch. Common centroid transistors are distributed while their center of gravity remains common. It is a point symmetric structure, position of source of variation is not important, mismatch is handled effectively. As shown in Figure 2.9, whether source of variations is at A or B is not important.

Conventional implementation of quad is shown in Figure 2.10. Transistors A and B are cross coupled, this kind of coupling requires two metal layers, which is not desirable. There is also unused space between transistors that wastes area.

Common centroid implementation in ALGv3 is different, already generated interdigitized structure is used for simplicity. This way layout is made simpler (single routing layer) and compact, unused area is minimized. Interdigitized rows are stacked vertically to form a quad as shown in Figure 2.10.

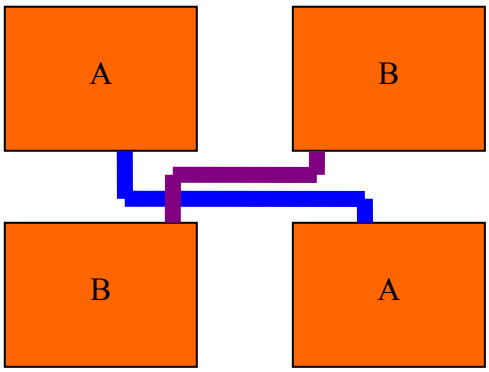


Figure 2.10. Routing of conventional common centroid transistors

Table 2.1. Map of transistors in a common centroid structure

A	B	A	B	A
A	B	A	B	A
A	B	A	B	A

Quad is a very difficult to draw by hand and almost an unmodifiable structure . This is not the case for module generators, any number of quads with any parameters can be generated immediately. Structures depicted above use or inherit properties of previous structures. This way, models do not differ, they just become combinations of previous models.

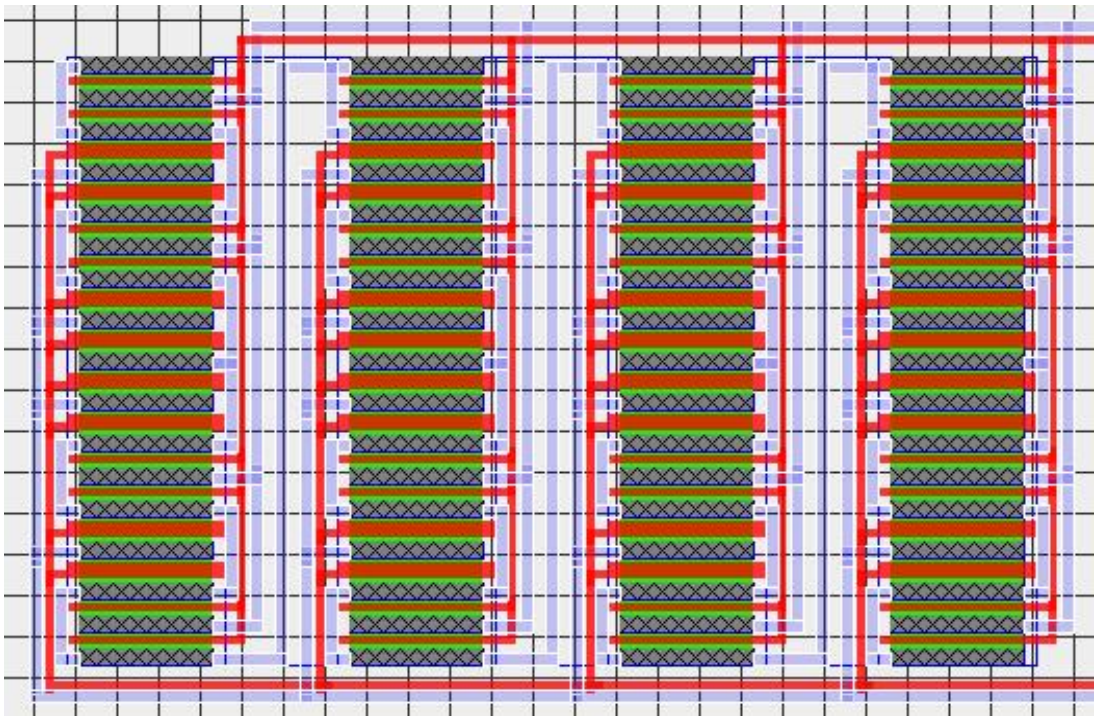


Figure 2.11. Layout of common centroid

Models used for this structure are shown below.

Capacitive Model:

Quad is composed of interdigitized structures, only difference is extra routing between these modules. Capacitance at each node can be found using interdigitized transistor's capacitive model adding wiring capacitance to it.

$$C_{drain1} = C_{drain2} = \text{numQuad} \,.\, C_{drain1Interdigitized} + C_{wiringQuad} \quad (2.19)$$

$$C_{source} = C_{sourceInterdigitized} + C_{wiringQuad} \quad (2.20)$$

Mismatch:

Mismatch is inversely proportional to square of numQuads and numFolds.

$$Mismatch = \frac{1}{(numQuads \cdot numFolds)^2} \tag{2.21}$$

Aspect Ratio:

height is proportional to numQuads

width is proportional to numFolds

Aspect ratio can be adjusted by changing numQuads and numFolds parameters. Increasing numFolds and numQuads increases matching but it also increases capacitance due to wiring giving rise to performance degradation.

2.4. Performance Oriented Module Generation

Modules can be generated with combinations of their parameters; number of folds parameter is used for folded transistors, number of quads and number of folds parameters are used for common centroid transistors. User can control all parameters of generated modules, but rule set for maximum performance cannot be determined easily. Trial and error method will work for placement but this is not enough; user has no way to know the node parasitics of the generated modules. Module generator extracts parasitics from modules and uses this data to generate a performance optimized module. Performance oriented generation is applicable to devices that have alternative realizations with same widths and lengths. Folded, interdigitized and common centroid structures are generated using a performance oriented approach.

Performance oriented module generation is based on testing all possible realizations of devices and selecting best of these. For example if a folded mos has width of 40λ and contact width of 4λ, possible realizations are 1, 2, 5 and 10 folds.

These transistors have same width and length but their aspect ratio and capacitance values are different. Best of these alternatives can be selected using cost function that combines aspect ratio and capacitance information.

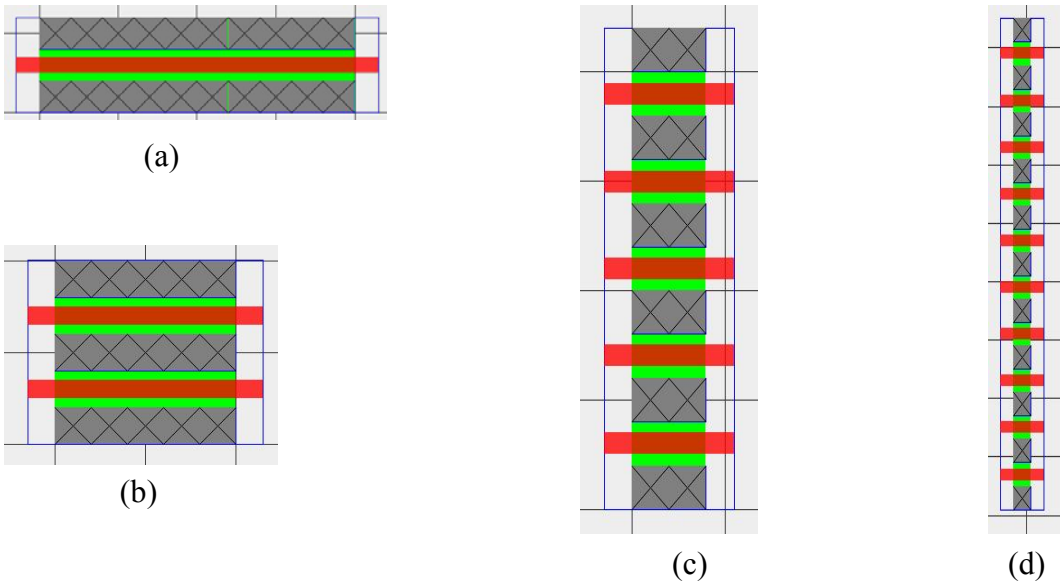


Figure 2.12. Folded transistors with different folds. (a) folds = 1, (b) folds = 2, (c) folds = 5, (d) folds = 10

2.5. Examples

Example 1: Common Centroid

width1 = 32 , width2 = 32 , length1 = 2 , length2 = 4

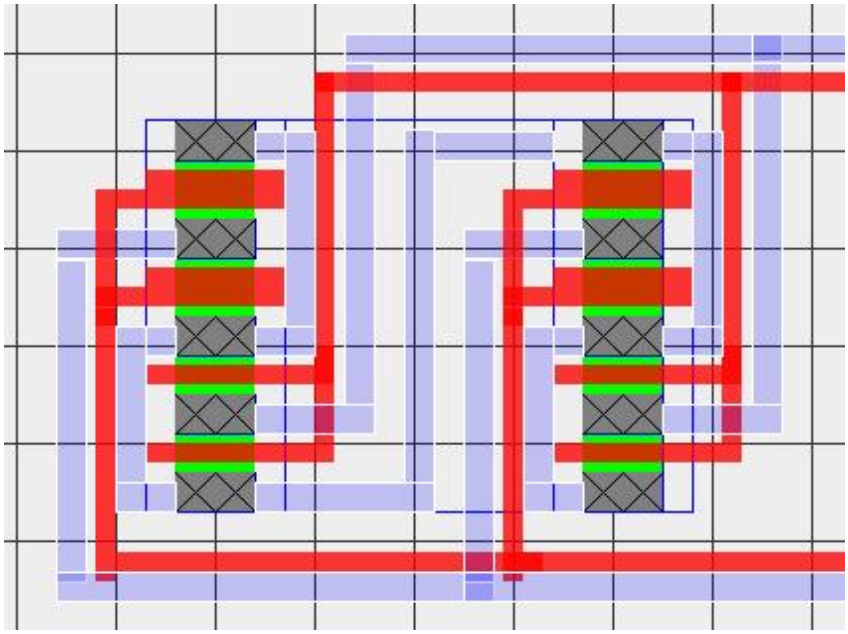
Coefficients in cost function are selected as follows:

$$\alpha_c = 1 \text{ , } \alpha_M = 4e\text{-}7 \text{ , } \alpha_{AR} = 5$$

Outcome of the module generator is shown below: best choice is numQuad=2, numFold=1

gcd: 8	numFold= 4 cost:187,92	numQuad=1...	Parasitics: Ctotal= 2,92	mismatch=0.031	AR=37,00
gcd: 8	numFold= 2 cost:12,55	numQuad=2...	Parasitics: Ctotal= 3,28	mismatch=0.0078	AR=1,85
gcd: 8	numFold= 1 cost:18,38	numQuad=4...	Parasitics: Ctotal= 4,00	mismatch=0.0020	AR=2,88
gcd: 16	numFold= 2 cost:49,15	numQuad=1...	Parasitics: Ctotal= 1,64	mismatch=0.13	AR=9,50
gcd: 16	numFold= 1 cost:8,13	numQuad=2...	Parasitics: Ctotal= 2,00	mismatch=0.031	AR=1,23
gcd: 32	numFold= 1 cost:13,53	numQuad=1...	Parasitics: Ctotal= 1,00	mismatch=0.50	AR=2,50

(a)



(b)

Figure 2.13. Module generation example 1. (a) Outcomes of cost function in module generator, (b) Layout

Example 2: Common Centroid generation

width1 = 256 , width2 = 256 , length1 = 2 , length2 = 4

Coefficients in cost function are selected as follows:

$\alpha_C = 1$, α_M=4e-7 , $\alpha_{AR} = 5$

minimum values are:

@ccNum=4, numFolds=4

@ccNum=8, numFolds=2

Table 2.2. Cost function table for example 2

Cost Function		Common centroid folds (ccNum)					
		1	2	4	8	16	32
Interdigitized Folds (numFolds)	1	1e6	231	5811	1466	373	98
	2	412	198	92	21	18	
	4	69	34	17	17		
	8	32	33	53			
	16	94	944				
	32	1930					

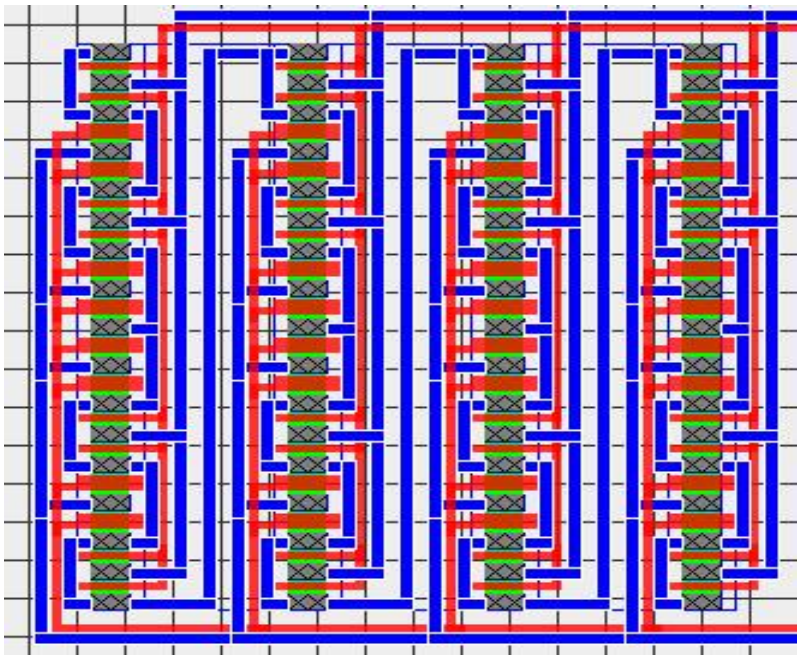


Figure 2.14. Module generation example 2

Example 3:

$width_1$=2000 , $width_2$=2000 , L_1=2, L_2=4

α_C =1 , α_M =4e-7 , α_{AR} = 5

results: ccNum = 5, numFolds = 10 , cost = 23.64

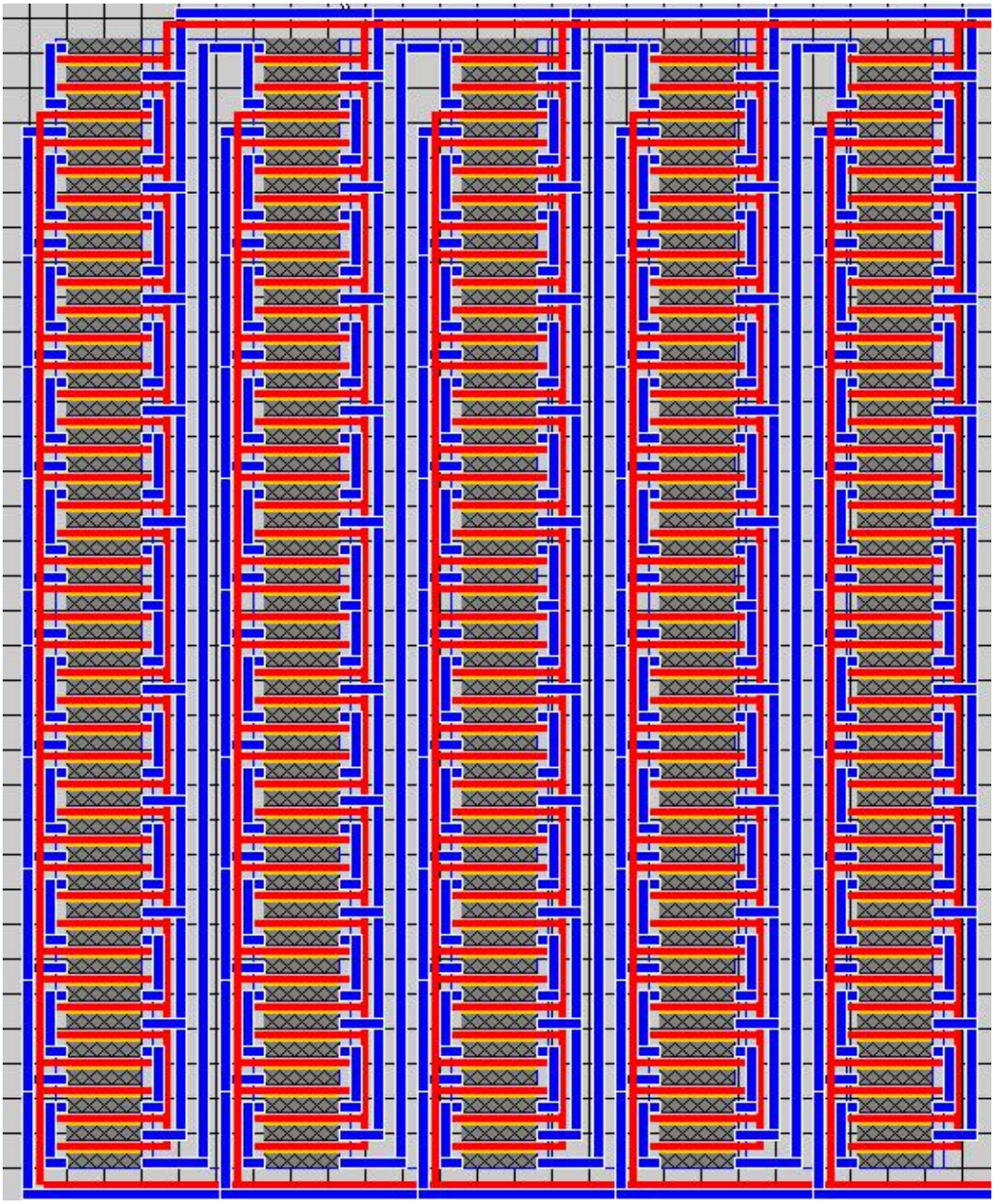


Figure 2.15. Module generation example 3

Example 4:

$width_1$=2000 , $width_2$=2000 , L_1=2, L_2=4

α_C =1 , α_M =1e-6 , α_{AR} = 5

results: ccNum = 10, numFolds = 5 , cost = 37.30. Comparing example 3 and example 4, all parameters are the same except matching coefficient (α_M); importance of matching is increased from α_M =4e-7 to α_M =1e-6. Layout changes according to cost function, reflecting the change in matching coefficient. Transistors are folded more in order reach a better matching.

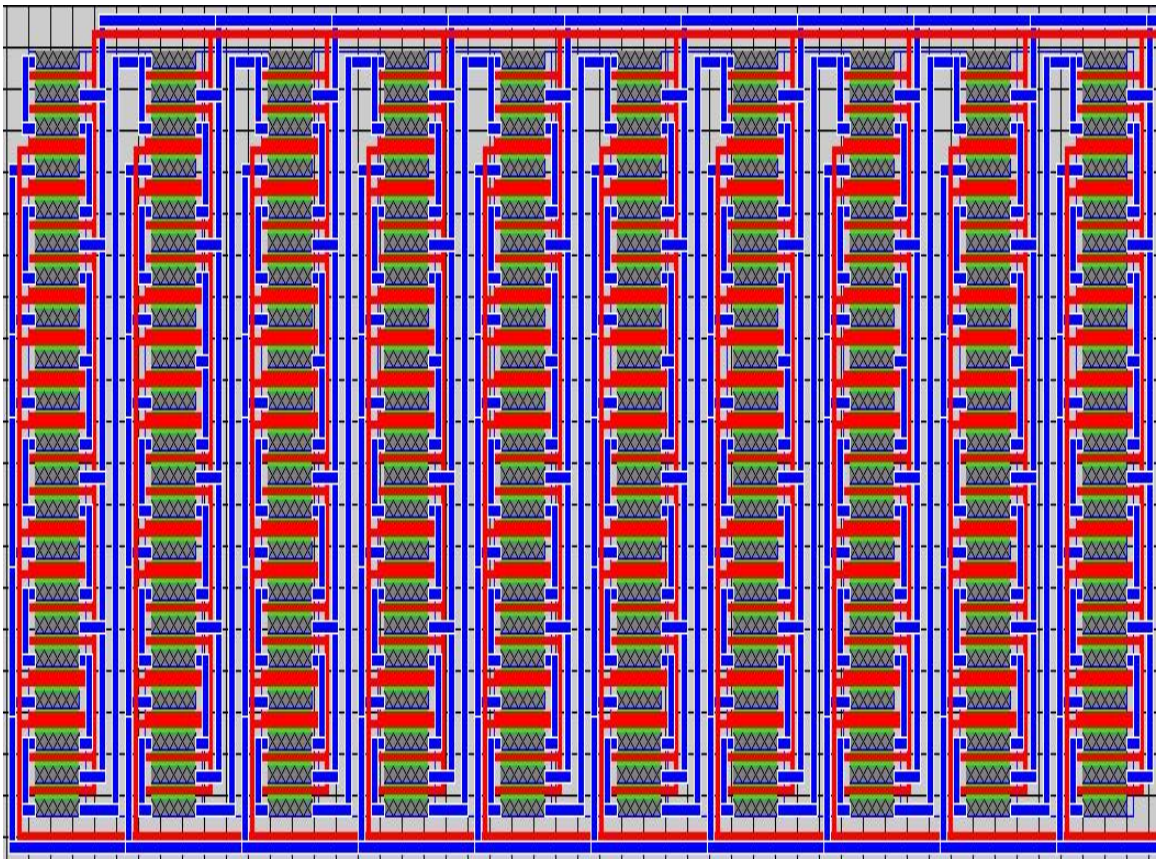


Figure 2.16. Module generation example 4

3. PARTITIONING

Partitioning is the first step in layout automation. The netlist provides devices and their connections to the partitioner where this data is processed and clues about the layout are extracted. Thus, purpose of the partitioner is to extract rules and clues about circuits and provide them to the placer. Partitioning can be regarded as a processing step for the placer.

Partitioning can be done hierarchically; devices in the circuit are divided into groups (partitions), which can also be grouped to form new bigger groups. The major aim of grouping is to reduce element count, because algorithms employed in analog design are usually NP complete, hence decrease in element count boosts the performance of the tool. Grouping devices can be performed in two ways; knowledge based and optimization based. The knowledge based approach is based on recognizing well known structures such as current mirrors and differential pairs. The optimization based approach, on the other hand, is based on grouping in such a way that connections between groups is minimized.

Partitioning stage is not necessarily used by all layout automation programs. In tools such as [13] automation can start with the placer directly. In our opinion, good layout can be achieved only by exploiting all information and clues about the circuit. Therefore, the partitioner gives valuable information to the placer which not only eases the burden on the placer, but also increases the quality of the layout.

3.1. Problem Specification

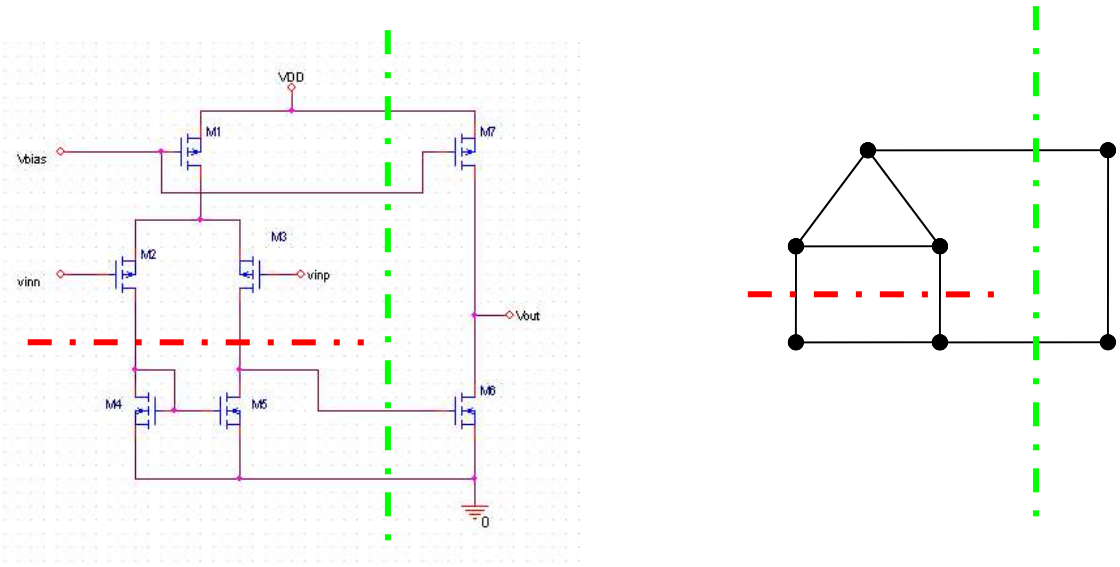


Figure 3.1. Partitioning of a differential amp.

A circuit can be represented by a graph, where devices correspond to vertices and connections correspond to edges. The partitioning problem is to group these vertices in such a way that the number of edges between these two groups is minimized. Since edges represent wires, minimization of wires between distant modules boosts performance. Edges can be weighted in order to make a distinction between important and unimportant wires, so that critical path length is minimized. The only criterion used in optimization is not to minimize wire crossing, further criteria can be employed for better results. Using area information of devices in partitioning, devices are grouped in such a way that their areas are equal or close.

3.2. Partitioning Algorithms

Some popular partitioning algorithms are reviewed below. They are extensively used by digital automation tools except the last one. Last algorithm is designed for this automation tool.

3.2.1. Kernighan Lin Algorithm and Its Successors

Kernighan Lin(KL) [26] algorithm starts partitioning by dividing vertices in two equal groups. A gain function is defined such that edges that are in the group have positive and edges that cross the boundary have negative effect on the function. A vertex pair is selected, one from each group so that exchanging these vertices will increase gain function using select-the-best method. Algorithm is repeated until no more gain is achievable.

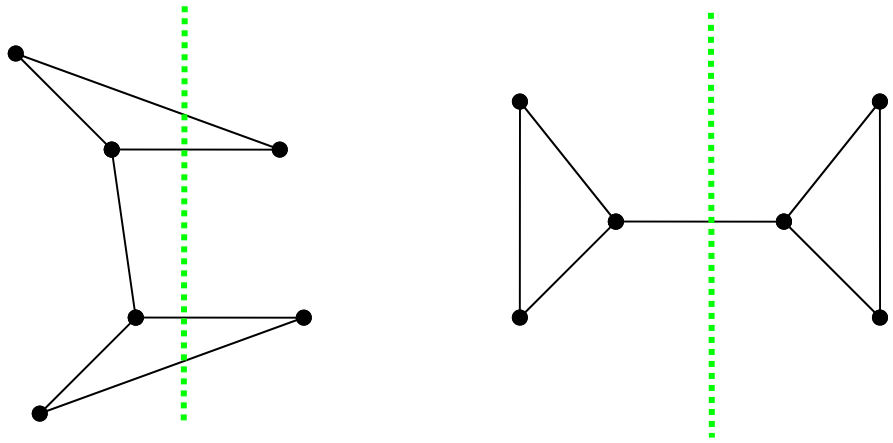


Figure 3.2. Kernighan Lin partitioning

The outcome of the Kernighan Lin algorithm depends on initial selection of the groups. Select-the-best method, which is used for selection of the best vertex pair to exchange, is a greedy algorithm that does not give best results, it usually gets stuck to a local minimum. A better algorithm like simulated annealing can overcome this problem but an even bigger problem remains; group sizes are fixed and not changeable.

An improved version of KL algorithm is Fiduccia-Mattheyses [27] algorithm.This is an improved version of KL algorithm. Single vertex is moved between groups in each iteration, so that group sizes are changed during partitioning. Algorithm is also optimized to work faster.

Goldberg and Burnstein [28] algorithm is another improved version of KL. It relies on the fact that the KL algorithm is efficient if ratio of number of edges to number of

vertices is higher than 5. In typical designs this ratio is 2. This algorithm aims to increase this ratio by matching vertices so that vertex count is decreased. After the partitioning process, matched vertices are separated.

Simulated annealing, simulated evolution and genetic algorithms are other alternatives for partitioning algorithms. These algorithms are not deterministic; they will give different results each time and the best result is not guaranteed to be one of these solutions. The solution is dependent on selected parameters. All of these algorithms work according to a cost function, determination of cost function is another problem, cost function does not always yield the true cost due to poor modeling.

It should be noted that algorithms used for partitioning are developed for digital design; vertices that are processed in digital design may be thousands or tens of thousands of gates. It is not surprising that the algorithms are not deterministic; element count strongly restricts the ability of deterministic algorithms.

The situation is different for analog design. There are not thousands or tens of thousands of gates; on the contrary there are only tens of elements at most. This feature opens the way of deterministic enumeration algorithms that find best solution by exploring all the solution space. Cost function or objective function is also very important here because selection is performed using this cost function, poor cost functions will result in meaningless grouping of elements.

3.2.2. Enumeration Algorithms

Analog circuits have much fewer elements than digital circuits, so partitioning algorithms can be allowed to be more time consuming if they guarantee a better result. If element count is below a limit (in practice, a few tens of elements), enumeration algorithms are best choices, they are guaranteed to find the best result. As concluded from the above discussion, there is a huge difference between algorithms employed in digital based partitioning and this one, the main difference being that enumeration guarantees to find the best result if there is one.

Enumeration algorithm is very advantageous in analog partitioning; however, this approach is not utilized in the literature to the best of our knowledge. Algorithms used for partitioning and placing are usually simulated annealing and genetic algorithms which are borrowed from digital design. Since analog design is different from digital design, it needs more specialized and effective algorithms, this being one of them.

It should be noted that although the enumeration algorithm finds the best solution, the solution is cost function dependent; a poor cost function will not be able to yield the best result. A simple cost function used in partitioning involves minimization of wires between partitions and equalization of areas of partitions. To be more specific; if elements are partitioned in two groups these groups should have similar areas and minimum number of wires connected between the groups.

3.3. Partitioning in This Tool

Partitioning algorithm implemented in this tool is enumeration due to its advantages. Steps of partitioning process are given below.

3.3.1. Partitioning of Circuit Using Cutset Matrix

Cutset is a set of edges that disconnects a graph in two or more parts. Since partitioning is simply “dividing circuits by two” method, using cutsets for partitioning is natural. Finding cutsets of a graph is a well known method and can be applied to the partitioning problem. There are three steps to find base vectors of cutset space.

1. Find spanning tree
2. Find circuit base matrix
3. Convert circuit matrix to cutset matrix

3.3.2. Finding Spanning Tree

Spanning tree of a graph is found using spanning tree algorithms [5][6]. Spanning tree is a non-cyclic edge set that includes all vertices in the graph. There may be many spanning trees in a graph, one of them will be sufficient for the remaining steps.

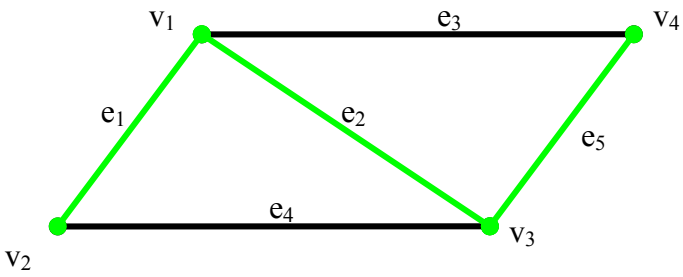


Figure 3.3. Spanning tree

For the graph shown above, { e_1, e_2, e_3 } is a spanning tree, { e_1, e_2, e_5 } and {e_4, e_5, e_3} are also spanning trees of the graph. One of these spanning trees will be used for the next step.

3.3.3. Finding Circuit List

The circuit basis matrix of a graph need not be unique, each spanning tree yields a circuit base matrix. Removing spanning tree from the graph is enough to find chords of the graph. Chords are edges that are not included in the spanning tree. Let spanning tree be { e_1, e_2, e_5 } for the figure, chords are { e_3, e_4 }. These chords are used to find circuits in the graph. Circuits are built so that each circuit contains only one chord. Number of circuits is equal to number of chords. Circuits for the figure are { e_2, e_3, e_5 } and { e_1, e_2, e_4 }.

Table 3.1. Circuits of a graph

		e_1	e_2	e_3	e_4	e_5
C =	C_1	1	1	0	1	0
	C_2	0	1	1	0	1

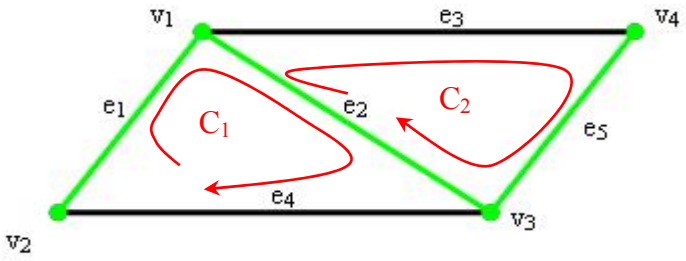


Figure 3.4. Circuits of a graph

3.3.4. Circuit List to Cutset List Conversion

Cutset basis matrix is found using circuits basis matrix. Circuit base matrix is shown in Table 3.1.

Chord columns are removed. $\{e_3, e_4\}$

Table 3.2. Modified circuit base matrix 1

		e_1	e_2	e_5
C_0 =	C_1	1	1	0
	C_2	0	1	1

Identity matrix is added to the bottom of C_0 , transpose of the matrix is cutset matrix of the graph.

Table 3.3. Modified circuit base matrix 2

		e_1	e_2	e_5
C_0' =	C_1	1	1	0
	C_2	0	1	1
		1	0	0
		0	1	0
		0	0	1

$$K = (C_0')^T \tag{3.1}$$

Table 3.4. Cutset base matrix

		e_4	e_3	e_1	e_2	e_5
K	K_1	1	0	1	0	0
	K_2	1	1	0	1	0
	K_3	0	1	0	0	1

Note that removed columns(chords) are labeled first, then the others follow. Using this base matrix, all cutset space can be generated. Each cutset entry (K_x) is multiplied with a constant, “0” or “1” and all these rows are added in modulus 2 to give a new cutset row.

$$K_X = a_1K_1 + a_2K_2 + ... + a_nK_n \tag{3.2}$$

If number of base rows is n, 2^n-1 rows are generated. Complexity of the algorithm grows exponentially with base vector count.

All these cutset rows partition the circuit in two or more partitions. Desired partitioning outcome is two partitions, all rows are applied to the circuit and tested to check whether the circuit is divided in two or more parts. Cutset rows that divide the circuit into two are added to cutset list. Selective addition to list limits usage of memory, list size is much less than generated cutset list.

A second selection mechanism is applied to the reduced cutset list, the aim of this step is to select partitions that have minimum cross connections and ratio of areas of partitions to be close to unity. Cost function is defined for selection purpose, it is composed of two terms: first is number of connections or sum of weights of the connections, second is ratio of areas of two partitions. What remains from a huge cutset list is a single row, this remaining cutset row divides the circuit in to two optimum partitions.

"Paritioning in two" procedure is applied recursively until nothing remains to partition, the new circuit is now composed of partitions which are also composed of sub partitions. The only exception is leaf partitions, which are composed of single devices.

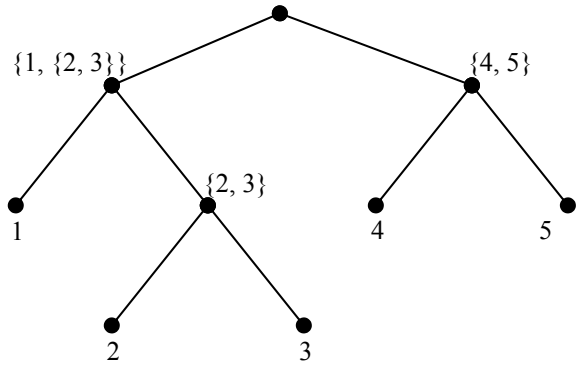


Figure 3.5. Partitioning tree

This algorithm works well, possibility of going out of memory is not that crucial; adjusting selection mechanism will solve the problem. Unfortunately, just partitioning is not enough for easy placement in analog circuits, another important issue is relative positions of these partitioned blocks.

3.4. Shortcomings of Partitioning for Analog Circuits

Enumeration-Partitioning algorithm discussed so far is sufficient for digital circuits, because they only need to be grouped. The same is not true for analog circuits; relative positions of partitions in analog circuits are very important. A simple initial placement step should also be included for better results. For example, when a group of digital gates are grouped into two, relative positions of these gates are not important, placing partitions horizontally or vertically does not alter the performance.

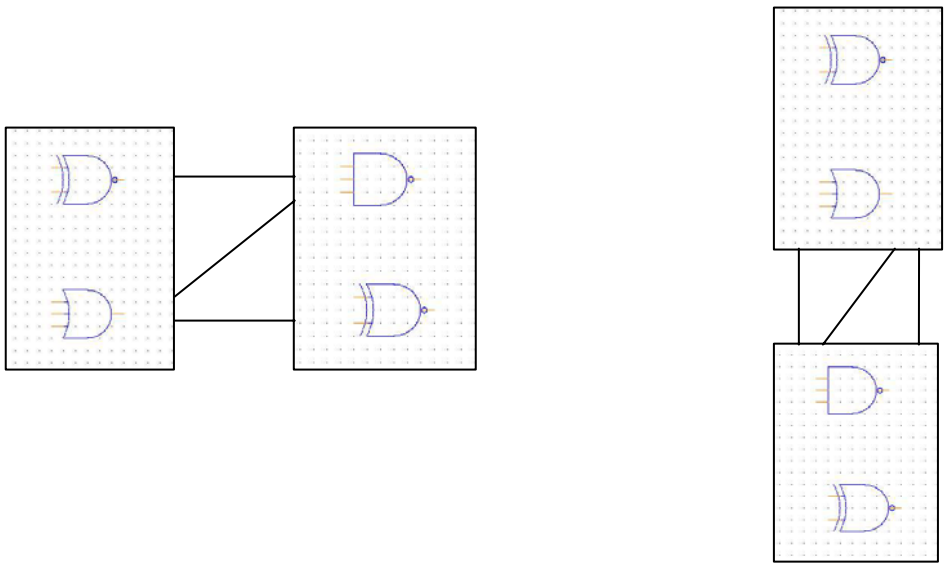


Figure 3.6. Horizontal and vertical placement of digital circuits

In the analog partitioning example shown in Figure 3.7, circuits are partitioned into two; circuit A and circuit B. Since there are electrical constraints, these partitions should not be placed vertically. So, in addition to the partitioning problem, there is an initial placement problem also. Details of partitioning (horizontal or vertical) should be determined. Even this is not enough, if the circuit will be partitioned vertically, next question is “which partition is at the top, which one is at the bottom.”.

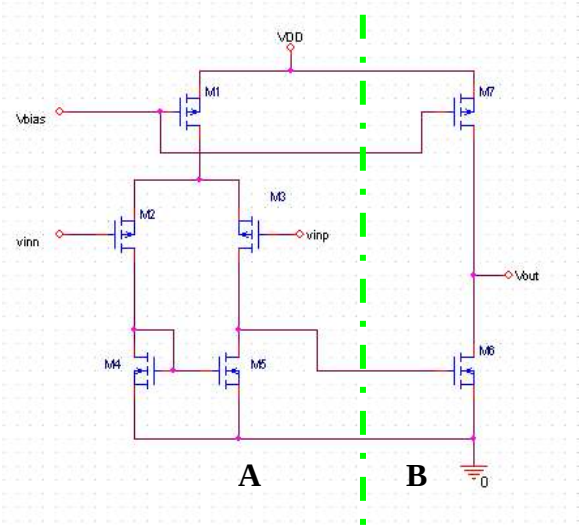


Figure 3.7. Horizontal partitioning of simple diff. amp. circuit

Using another enumeration algorithm to test all placement possibilities is not feasible, there are eight placing possibilities for each partition and the new enumeration algorithm will be even harder the previous one. Another way should be used instead of enumeration. A heuristic algorithm is implemented to solve this problem; it uses information of potential levels of devices.

3.4.1. Determination on Potential Levels of Transistors

The spice file parser processes the netlist and sends information to graph generator where graph of the circuits is generated. This is the usual procedure which is used in the partitioner. In this step however, graph generator works in a different manner; graph is generated but only current conducting wires are connected in the graph. Drain to drain and drain to source connections are used, gate connections are not used. As shown on Figure 3.8, edges connect Vdd and Gnd rail to rail.

Potential level determination algorithm starts assigning levels from the bottom, the number associated to n-transistors is increased at each step while going up. When all

vertices are assigned a number, process goes from top to the bottom assigning decreasing numbers in each level only to p-type devices.

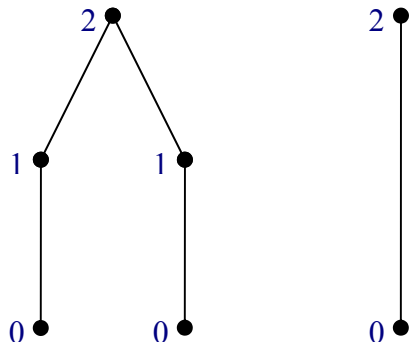


Figure 3.8. Potential levels of simple diff. amp.

3.4.2. Partitioning Of The Circuit Using Potential Level Information

Using this new information, partitions are placed with less effort. Coming back to the question about placing partitions horizontally or vertically, solution is as follows. If the average value of potential levels of a partition is higher than the other, partitions are placed vertically, if not they are placed horizontally. Actually there are three questions in the relative partition placement problem;

1. Should placement be performed horizontally or vertically?
2. If horizontal, which partition is on the left, which is on the right?
3. If vertical, which partition is on top, which is at the bottom?

Potential levels method not only solves the first problem but it also solves the third problem, details of vertical placement are available. The only remaining question is: If placement is horizontal, how can we determine which partition is at left and which is on the right.

There are many heuristic approaches to solve the problem, they bring some improvement but none of them is sufficient to reach the goal. One of the best solutions is to place horizontal partitions randomly and try to find their correct position.

Using new information about relative placement of modules, the partitioning tree is modified to include placement data. Vertical placed partitions are given directions north and south, horizontal placed partitions are given west and east. New tree contains both partitioning and relative placement data. In the figure shown below, nodes of the tree are labeled according to their relative positions.

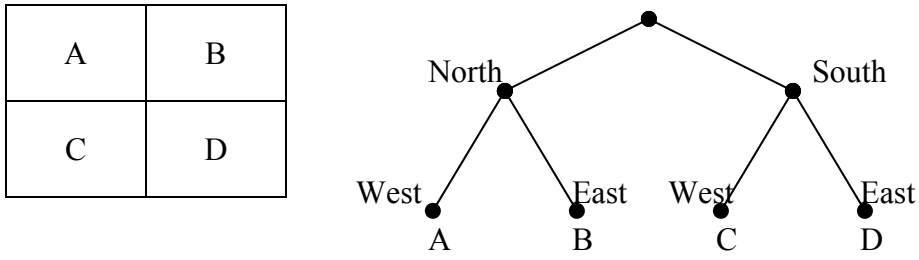


Figure 3.9. Partitioning with directions

Initial placement of modules in a horizontally aligned partition is an unsolved problem. Order of transistors from left to right is not determined until now. Partition exchanging algorithm is implemented to solve this problem. Local search is performed to find horizontal neighbor pairs that are also vertical neighbors of other horizontal neighbor pair. Figure 3.9 contains neighbors A and B that are vertical neighbors of C and D. If this is not a good placement, exchanging one of the pairs is enough to solve the problem. A goal function is defined between these partitions; vertical connections to vertical neighbors increases the gain (A-C, B-D) where vertical non-neighbor connections decreases the gain(A-D, C-B). If exchange of a horizontal pair increases the gain, they are exchanged giving a better placed layout. Algorithm is convenient for local changes, only neighbors are exchanged. Actually non neighbor partitions are also exchanged, an intermediate partition between {A,B} and {C,D} will not affect the algorithm. However exchange will

not occur if {A,B} and {C,D} partitions are placed horizontally (all partitions are horizontally placed).

3.5. Partitioning Examples

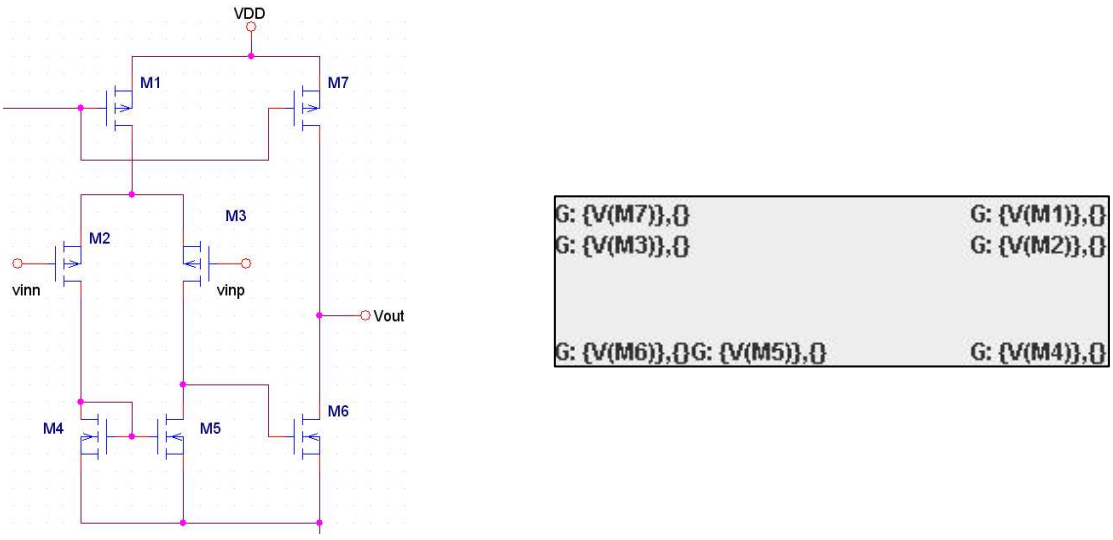
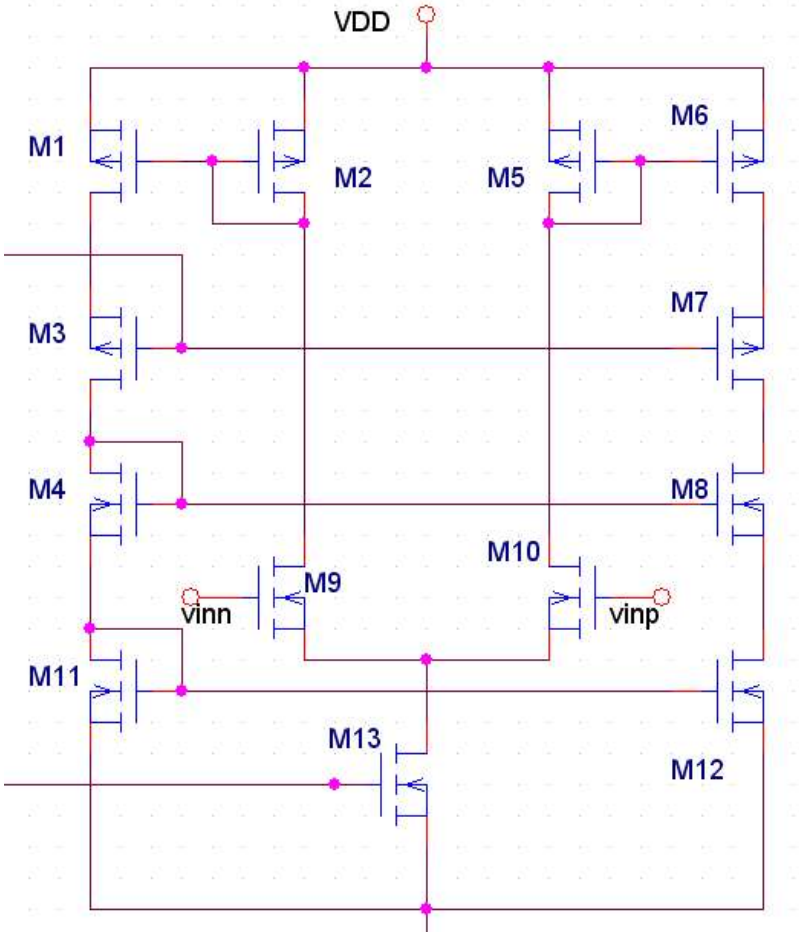


Figure 3.10. Partitioning of simple diff. amp.



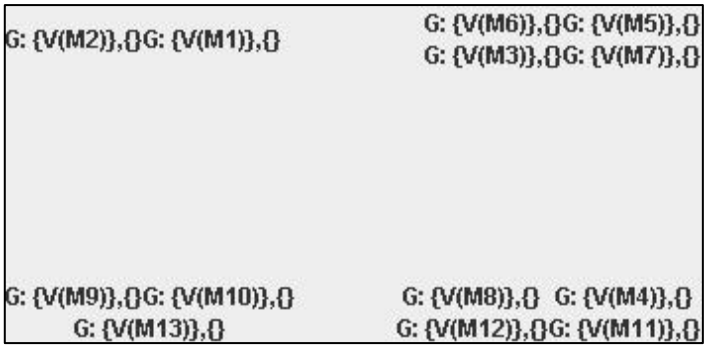


Figure 3.11. Partitioning of diff. cascode amp.

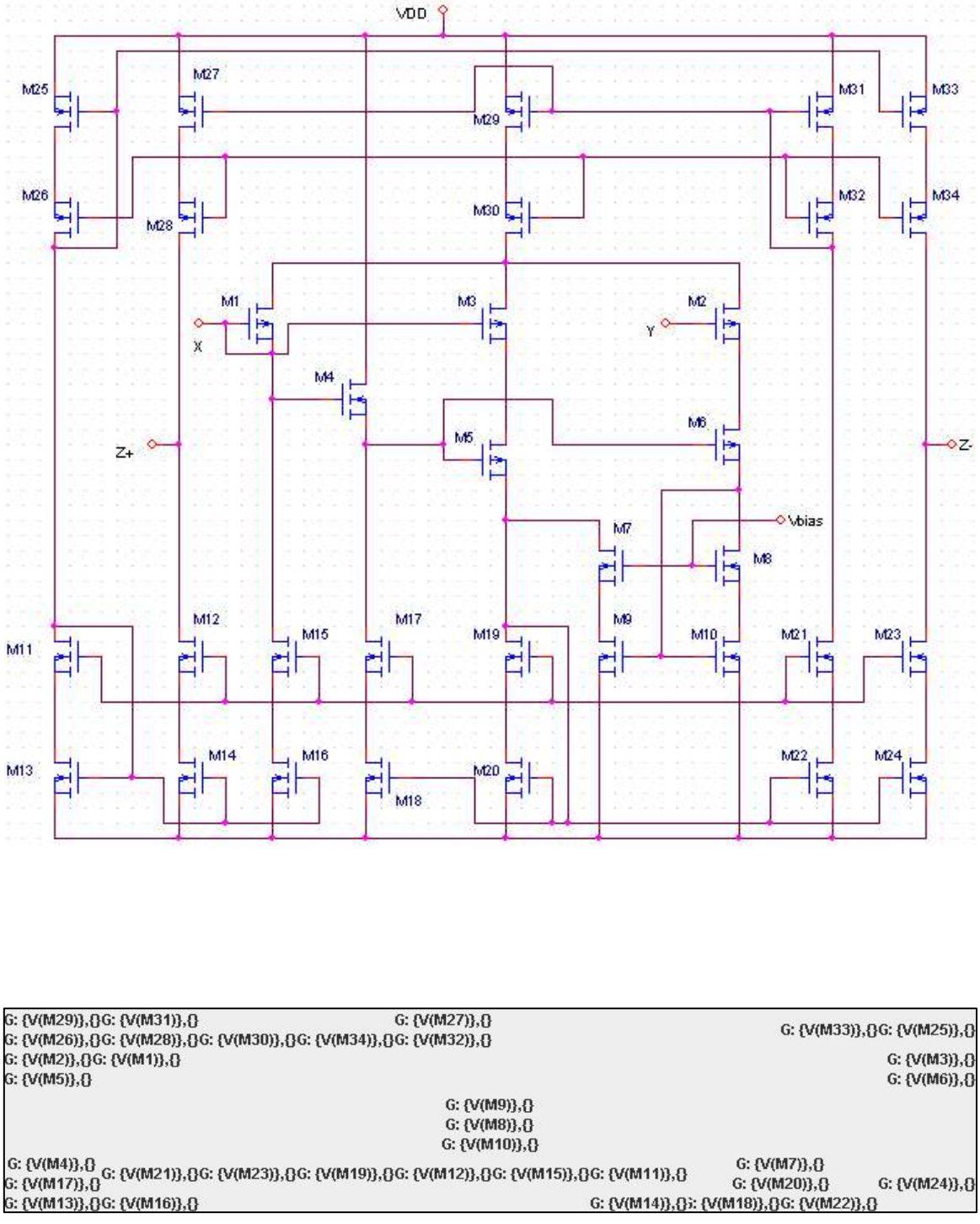


Figure 3.12. Partitioning of CCI low voltage

Partitions with only connections and potential levels information do not give reasonable results. As can be seen especially in the last example, transistors in current mirrors are not grouped together, instead they look as if they are distributed randomly.

Well known structures (current mirrors, diff amps) should be kept together as a module. Keeping transistors in a module both decreases the complexity of the problem and increases matching. A topology matcher tool was implemented for this purpose. This tool is explained in the following section.

3.6. Topology Matcher

Topology matcher extracts well known structures such as current mirror, cascode current mirror and differential pairs. Matching well known structures reduces the number of elements in the circuit. Since circuits are composed of these structures, element count reduces to an easily managable value. There are many well known structures in the literature, the most important ones are implemented in topology matcher. The list is given below.

- Simple Current Mirror
- Cascode Current Mirror
- Low Voltage Current Mirror
- Differential Pair
- Differential Cascode Pair

Instead of implementing these structures one by one, the hierarchy given below is used. Implementing each structure is a time consuming and hard to maintain job, a simple hierarchy is used which shows the relations and properties of the structures. Root node is the most general one, more specialized structures move away from the root. Each sub node inherits properties of their super node, common gate - common source node inherits properties of common gate node for example. This way, design effort is minimized and simplified. Complex structures are composed of simple ones; n stage cascode current mirror for example is composed of n cascoded simple cascode current mirrors.

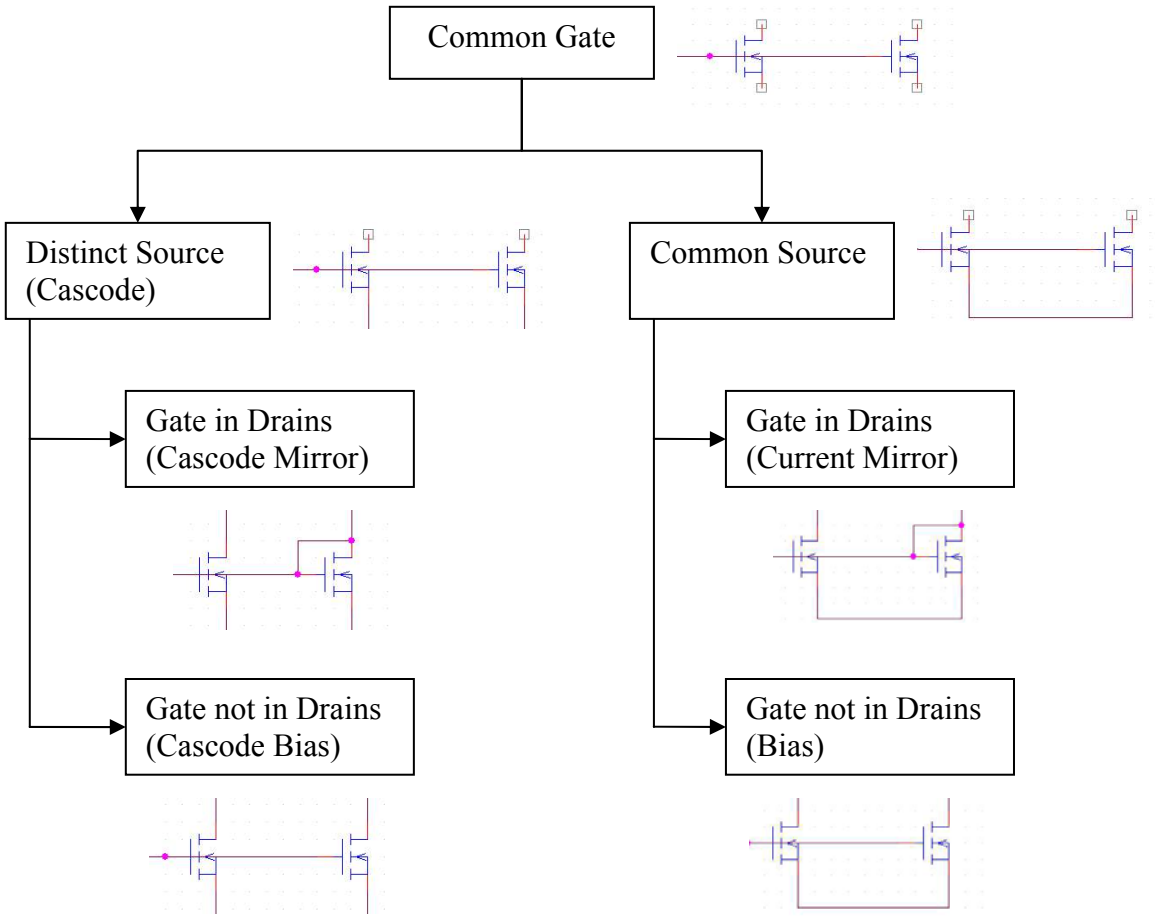


Figure 3.13. Common gate hierarchy

Common gate is the most general type in mirrors; all current mirrors are common gate structures. If sources are distinct for each transistor (cascode structure) it can be stacked vertically to form cascoded structures. Type of the cascade for mirrors is determined according to nodes drains and gate; if gate node is connected to one of the drain nodes structure is cascade current mirror, if gate is not connected to drains, structure is cascade bias.

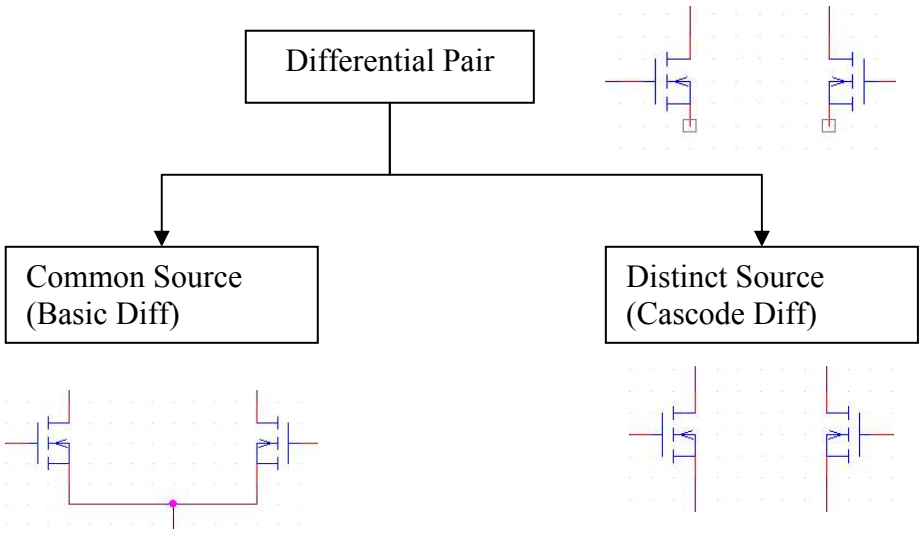


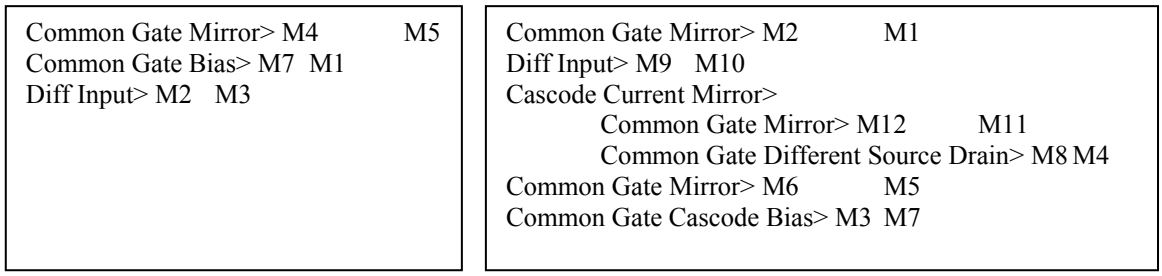
Figure 3.14. Differential pair hierarchy

Complex structures can be generated using simple ones in the hierarchy. For example, three stage current mirrors can be built using a simple mirror and two cascode bias structures. Cascode differential pair can be built by cascoding cascode mirror, cascode diff and a cascode mirror respectively. This composition technique greatly simplifies the algorithm.

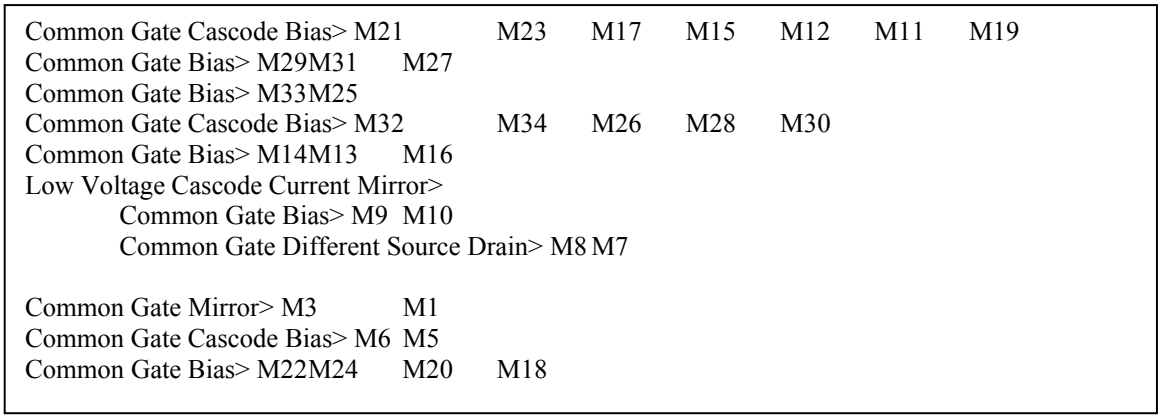
There are two hierarchies; first one is for current mirrors, second one is for differential pairs. Structures with same bulk, same size and different drains are differential pairs. If their sources are also common, they are called simple differential inputs, if not cascode differential input.

Topology matcher matches structures in the circuit starting from the most specialized ones such as cascode current mirror, basic diff. Each matched structure is removed from the circuit and matcher is reapplied until nothing remains to be matched.

3.6.1. Topology Matcher Examples



(a) (b)



(c)

Figure 3.15. Outputs of topology matcher. (a) Simple diff. amp., (b) Differential cascode amp., (c) Low voltage CCII

4. PLACEMENT

Placement is the most important part of a layout automation tool. Information extracted at partitioning step is used to place blocks consisting of devices. Quality of placement affects the routing step; high quality placement eases the job of the router.

Modules generated by the module generator are not assigned position information after the generation. Adjustment of positions of the modules is the job of the placer. The position of the floating modules should be set so that technology rules are not violated. There are many constraints to satisfy; first group is technology constraints, second group is performance criteria.

Generated modules are represented as blocks in the placer; they can be moved, rotated or grouped. This is similar to a rectangle compaction problem; rectangles should be placed so that none of them overlap and total used area is minimized. Difference between placer and rectangle packer is additional constraints; some rectangles should be kept together where some of them should be apart. Orientation of rectangles is also a constraint.

There are two types of placers in ALGv3, first one is rule based placer, it uses a control file that controls the placement of the layout, and second one is optimization based placer. Force directed placement algorithms are implemented in the automatic placer. User has full control on the layout in rule based mode, not only on placement; partitioning and module generation is also controlled. Details of rule based placement are explained in the next section. Automatic mode uses some heuristics as potential levels information to place modules vertically and force directed placer to find optimum positions of modules.

4.1. Rule Based Placement

User commands are read from a control file which is supplied in addition to the netlist. Grouping and orientation information is used in placer. Devices are grouped together and then assigned direction.

```
.part    p1 M1 M2
         p2 M3 M4 M5
         p3 p1 p2
.place   p1 horizontal
         p2 horizontal
         p3 vertical
```

Lines given above are a part of the control file. Transistors M1 are M2 are grouped in p1, M3 M4 and M5 are grouped in p2. Groups p1 and p2 are included in p3. A hierarchy is generated, where p3 is the root of the tree and transistors are leaves. The last three lines shows how orientation of the partitions are assigned; p1 and p2 are packed horizontally, then these horizontally packed partitions are vertically packed to form p3. Layout example of these lines is shown below.

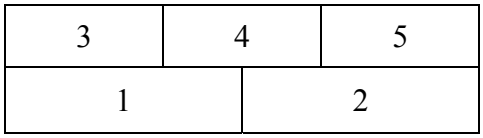


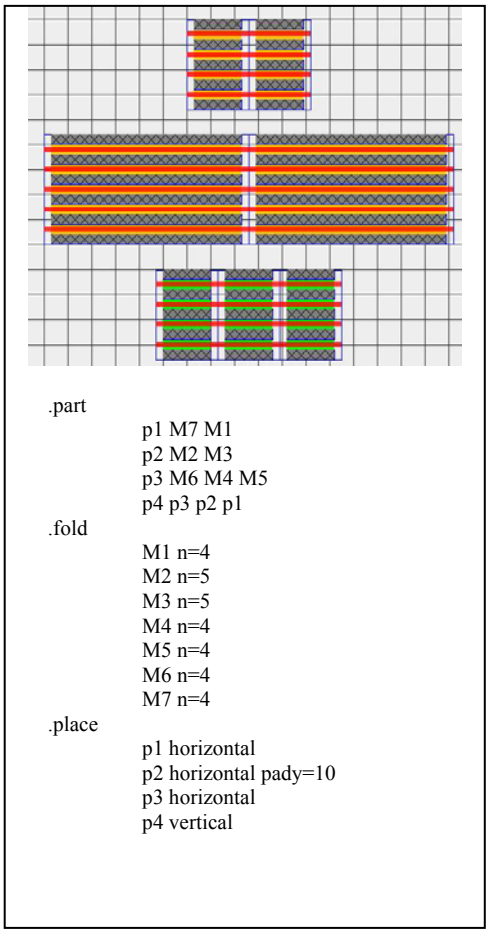
Figure 4.1. Layout of control lines.

This way the entire layout is generated. Control structure is simple and easy, it can be easily modified. Changing order of transistors will arrange modules in the layout to reflect the changes, changing orientation of p3 for example will result in a long horizontal layout.

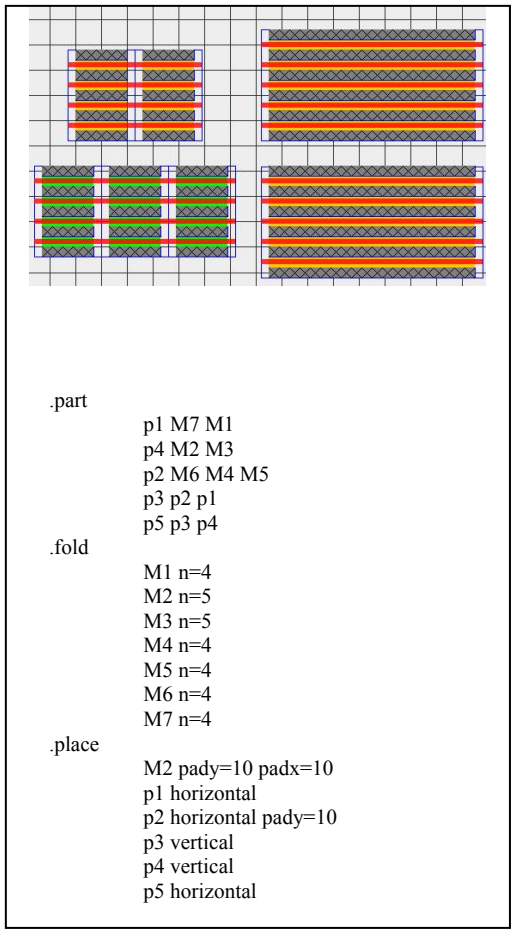
Advantage of rule based placement is that it works very fast, computation is not necessary. All rules are supplied by user and everything is controllable.

Rule based algorithms have some disadvantages; quality of the layout is highly dependent on control commands supplied by user who may not know the best configuration for the optimum result.

Some outputs of rule based placement are shown in Figures 4.2, 4.3 and 4.4.

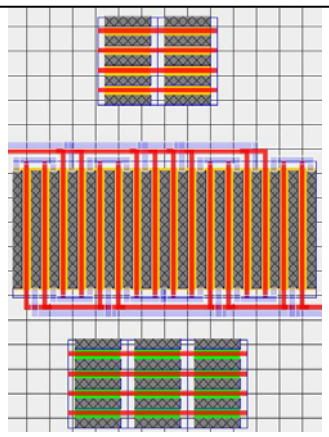


```
.part
        p1 M7 M1
        p2 M2 M3
        p3 M6 M4 M5
        p4 p3 p2 p1
.fold
        M1 n=4
        M2 n=5
        M3 n=5
        M4 n=4
        M5 n=4
        M6 n=4
        M7 n=4
.place
        p1 horizontal
        p2 horizontal pady=10
        p3 horizontal
        p4 vertical
```



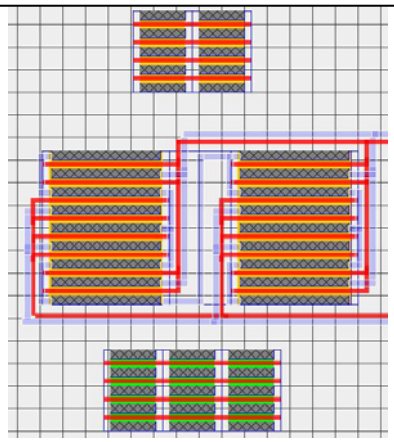
```
.part
        p1 M7 M1
        p4 M2 M3
        p2 M6 M4 M5
        p3 p2 p1
        p5 p3 p4
.fold
        M1 n=4
        M2 n=5
        M3 n=5
        M4 n=4
        M5 n=4
        M6 n=4
        M7 n=4
.place
        M2 pady=10 padx=10
        p1 horizontal
        p2 horizontal pady=10
        p3 vertical
        p4 vertical
        p5 horizontal
```

Figure 4.2. Different layouts of two differential amplifiers.



```
.part
        p1 M7 M1
        p2 M2 M3
        p3 M6 M4 M5
        p4 p3 p2 p1
.inter p2 n=4
.fold
        M1 n=4
        M2 n=5
        M3 n=5
        M4 n=4
        M5 n=4
        M6 n=4
        M7 n=4
.place
        p1 horizontal
        p2 vertical pady=20
        p3 horizontal
        p4 vertical
```

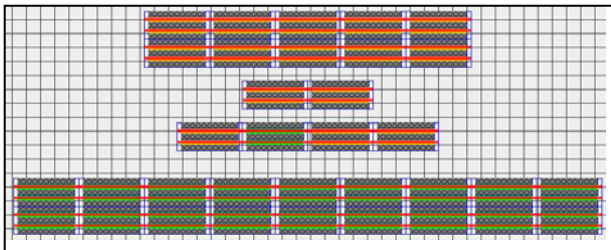
(a)



```
.part
        p1 M7 M1
        p2 M2 M3
        p3 M6 M4 M5
        p4 p3 p2 p1
.quad p2 q=2 n=2
.fold
        M1 n=4
        M2 n=5
        M3 n=5
        M4 n=4
        M5 n=4
        M6 n=4
        M7 n=4
.place
        p1 horizontal
        p2 vertical pady=20
        p3 horizontal
        p4 vertical
```

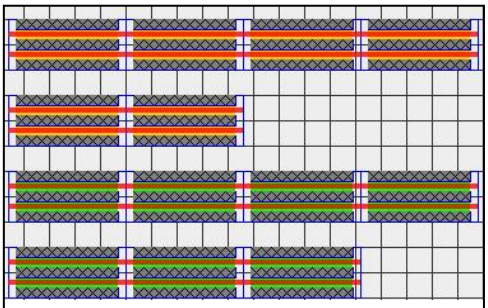
(b)

Figure 4.3. Interdigitized and Quad diff. amps. (a) Interdigitized, (b) Quad.



```
.part
        p1 M25 M26
        p2 M27 M28
        p3 M29 M30
        p4 M31 M32
        p5 M33 M34
        p6 p1 p2 p3 p4
p5 // top row horizontal
        p7 M1 M4 M3
M2 // horizontal
        p8 M5 M6 //
horizontal
        p9 p7 p8 //
vertical
        p10 M13 M11
        p11 M14 M12
        p12 M16 M15
        p13 M18 M17
        p14 M20 M19
        p15 M9 M7
        p16 M10 M8
        p17 M22 M21
        p18 M24 M23
        p19 p10 p11
p12 p13 p14 p15 p16 p17
p18 // bottom row
horizontal
        p20 p19 p9 p6
.fold
M1 n=2
M2 n=2
M3 n=2
M4 n=2
M5 n=2
M6 n=2
M7 n=2
M8 n=2
M9 n=2
M10 n=2
M11 n=2
M12 n=2
M13 n=2
M14 n=2
M15 n=2
M16 n=2
M17 n=2
M18 n=2
M19 n=2
M20 n=2
M21 n=2
M22 n=2
M23 n=2
M24 n=2
M25 n=2
M26 n=2
M27 n=2
M28 n=2
M29 n=2
M30 n=2
M31 n=2
M32 n=2
M33 n=2
M34 n=2
.place
p1 vertical
p2 vertical
p3 vertical
p4 vertical
p5 vertical
p6 horizontal pady=10
p7 horizontal pady=10
p8 horizontal
p9 vertical
p10 vertical
p11 vertical
p12 vertical
p13 vertical
p14 vertical
p15 vertical
p16 vertical
p17 vertical
p18 vertical
p19 horizontal pady=10
p20 vertical
```

(a)



```
.part
        p1 M1 M2 // h
        p2 M5 M6 // h
        p3 p1 p2 // h top row
        p4 M3 M7
        p5 M4 M8 p7
        p6 M11 M12 M13
        p7 M9 M10
        p8 p6 p5 p4 p3
.fold
M1 n=2
M2 n=2
M5 n=2
M6 n=2
M3 n=2
M7 n=2
M4 n=2
M8 n=2
M11 n=2
M12 n=2
M9 n=2
M10 n=2
M13 n=2
.place
        p8 vertical allignleft
        p5 pady=10
        p3 pady=10
```

(b)

Figure 4.4. Rule base placed circuits. (a) CCII low voltage, (b) Diff. cascode

4.2. Automatic Placement – Optimization Based Placement

Second option of the placer is optimization based placement. Positions of the blocks are not determined by user commands, instead they are automatically calculated. Optimization based placement explores the solution space using no human effort. Performance oriented criteria are also considered, where this is not possible in the rule based placement.

There are infinitely many combinations of generating a layout, optimization based algorithm tries to find best of these combinations. Exploring the whole of solution space is not feasible; some heuristics are used to confine it. These heuristics are explained in each algorithm.

Three of the most widely used optimization based algorithms are explained. Most popular algorithms are simulated annealing and simulated evolution. Both of them are probabilistic. The third algorithm explained here, which is not as popular as the previous one is force directed placement. Force directed placement is a simple physical simulator, it is not probabilistic but it is highly initial condition dependent.

4.2.1. Simulated Annealing

This algorithm is based on manipulating a system with certain operators and then calculating a cost function to see if the cost is reduced or not. If cost is reduced, manipulation is accepted, if cost is increasing, manipulation is accepted with a temperature dependent probability function. Probability function is controlled via a temperature parameter where it is altered (reduced usually) according to a temperature schedule. Cost of the system reduces to a minimal point, this is the solution of the algorithm.

The algorithm is able to perform two tasks: manipulate the system and calculate the cost of the system. A temperature based probabilistic control mechanism combines these two into a simulated annealer.

Simulated annealing is a probabilistic algorithm and can be applied to NP-Complete problems, which are not solvable using deterministic algorithms. If configured correctly, it converges quickly to the optimum solution. Simulated annealing is controlled via initial temperature parameter and temperature schedule. These parameters affect the solution, the parameter set may change from problem to problem in order to get the best result. Temperature is not the only problem; another limitation is quality of the cost function. A good model of the system is necessary in order to get an effective cost function.

4.2.2. Simulated Evolution

This is another popular algorithm; it is similar to simulated annealing algorithm. A cost function is employed as in all optimization based algorithms and a manipulation system, which is inspired from genetics is used. Difference of genetic algorithm from simulated annealing is that they do not have manipulation operators like translate or rotate. Instead all variables in the system (positions of the blocks in our problem) are connected together to form a string, it is actually called genome. Initially a random set of genomes are generated and the cross-over is applied with pairs or mutated mutually. Cross-over and mutation enables algorithm to explore the solution space. Each solution is tested using a cost function. If the solution is weak compared to other solutions, it is removed from the list. Outcome of the algorithm may depend on its parameters; mutation and crossover rate.

4.2.3. Force Directed Placement (FDP) [19][20]

This algorithm is not as popular as the previous ones. A spring system is employed in this algorithm. Each block is connected with springs, where modules are represented by blocks, and wires are represented by springs. Devices that are connected with wires are connected with springs that try to close the distance between modules, on the other hand devices that are not connected with wires are connected with repelling springs that try to increase the distance. Final positions of the blocks are determined with a few Newton-Raphson iterations.

4.3. Placement Algorithm Implemented in This Tool

Force between two blocks are calculated using Hook's Law, F = -k.Δx. Where k is the spring constant and Δx is displacement of the spring.

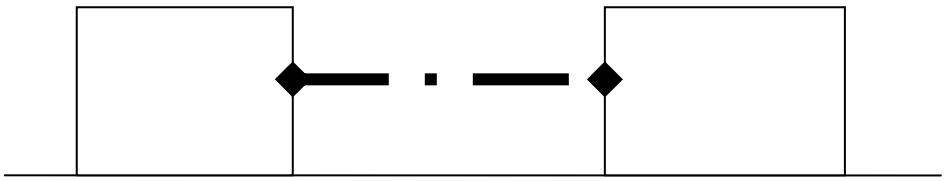


Figure 4.5. Two body Hook's Law

Model used for n-body system is the generalized version of the two body system. Force function is divided into two; first formula is used for connected blocks, force between these blocks is attractive, second formula is for unconnected blocks, force is repulsive between the blocks.

$$F_{ij} = \begin{cases} -K_{ij}\left(\dfrac{\Delta x_{ij}}{\Delta y_{ij}}\right), \textit{blocks are connected} \\ \dfrac{R_{ij}}{\Delta S_{ij}}\left(\dfrac{\Delta x_{ij}}{\Delta y_{ij}}\right), \textit{otherwise} \end{cases} \tag{3.3}$$

$$K_{ij} = 0, \textit{if blocks i and j are not connected} \tag{3.4}$$

$$R_{ij} = \begin{cases} \sum_{i}^{n}\sum_{j}^{n}\dfrac{K_{ij}}{n}, \textit{if blocks i and j are not connected} \\ 0 \quad , \quad \textit{if connected} \end{cases} \tag{3.5}$$

$$\Delta S_{ij} = \left|\Delta x_{ij}\right| + \left|\Delta y_{ij}\right| \tag{3.6}$$

Using the model given above, forces are calculated between each block and they are summed for each block. For example, force acting on the kth block is given by F_k below.

$$F_k = \sum_i^n F_{ik} \tag{3.7}$$

The next step is to calculate the derivatives of these forces, they will be used in next section in order to calculate the amount of movement.

$$F_{ij}' = \begin{cases} -K_{ij} & ,blocks\ are\ connected \\ \dfrac{R_{ij}}{\Delta S_{ij}^{\,2}}\left(\dfrac{\Delta x_{ij}}{\Delta y_{ij}}\right), & otherwise \end{cases} \tag{3.8}$$

All these results are combined in Newton-Raphson method to calculate the amount of movement. The term "1/2" is used for ½ movements for reciprocal cells.

$$x_{i+1} = x_i - \frac{1}{2}\cdot\frac{F_x}{F_x'} \tag{3.9}$$

$$y_{i+1} = y_i - \frac{1}{2}\cdot\frac{F_y}{F_y'} \tag{3.10}$$

Next position is calculated using current positions and force acting on the block. Each iteration gets the blocks closer until no net force remains. Steps are given below:

1. $F_j = \sum_i -K_{ij}\left(\dfrac{\Delta x_{ij}}{\Delta y_{ij}}\right) + \dfrac{R_{ij}}{\Delta S_{ij}}\left(\dfrac{\Delta x_{ij}}{\Delta y_{ij}}\right)$
2. $F_j' = \sum_j -K_{ij} + \dfrac{R_{ij}}{\Delta S_{ij}^{\,2}}\left(\dfrac{\Delta x_{ij}}{\Delta y_{ij}}\right)$
3. $x_{i+1} = x_i - \dfrac{1}{2}\cdot\dfrac{F_x}{F_x'}, \quad y_{i+1} = y_i - \dfrac{1}{2}\cdot\dfrac{F_y}{F_y'}$
4. If total movement is below a threshold goto 5 else goto 1

5. END

4.3.1. Avoiding Overlap

Since distance between blocks is calculated with respect to their center position, if algorithm is applied without an overlap avoiding mechanism, centers of connected blocks will converge to a common point. An overlap avoiding mechanism is implemented to overcome this unwanted situation. Movement calculation discussed is previous section is not modified in order to fix the problem, instead it is followed by a movement manager which takes care of overlapping.

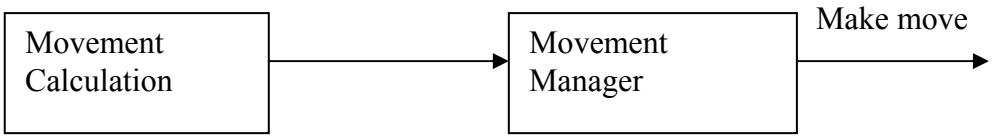


Figure 4.6. Overlap avoiding mechanism

4.3.2. Examples of Force Directed Placement Applied to Blocks

First example is successively connected blocks. Block are aligned on a line in a few iterations. Connections are as follows: (1, 2), (2, 3), (3, 4), (4, 5), (5, 6).

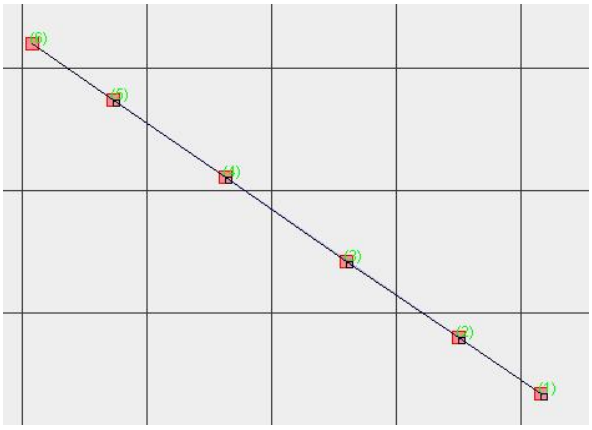


Figure 4.7. Linearly placed blocks

Second example is shown in Figure 4.8. Blocks are connected in a circular manner: (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 1).

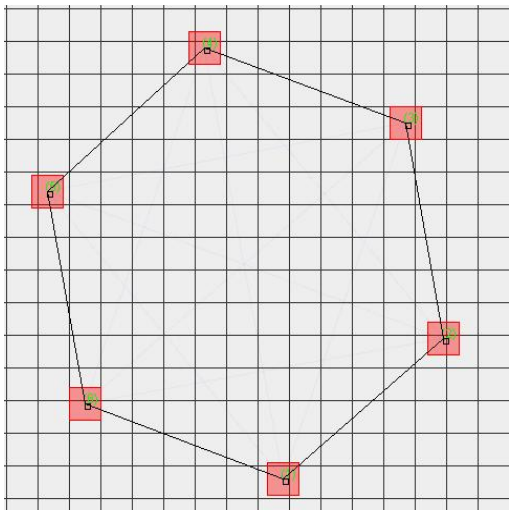


Figure 4.8. Hexagonal placed blocks

In the third example, (4, 6) connection is added to the previous example. This extra connection reduced the distance between 4^{th} and 6^{th} blocks.

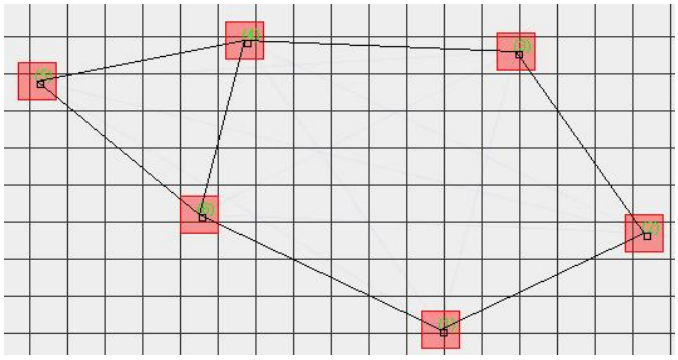
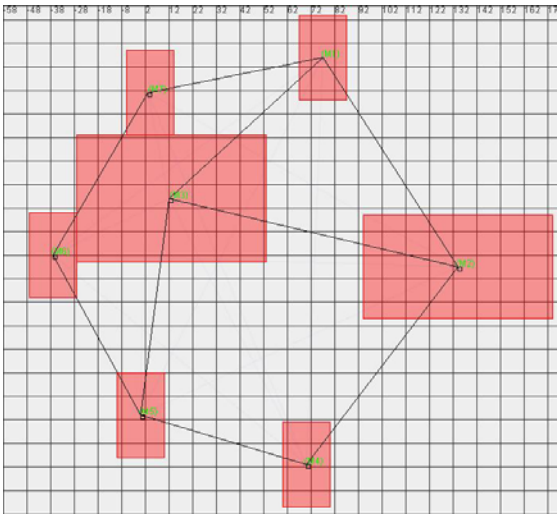


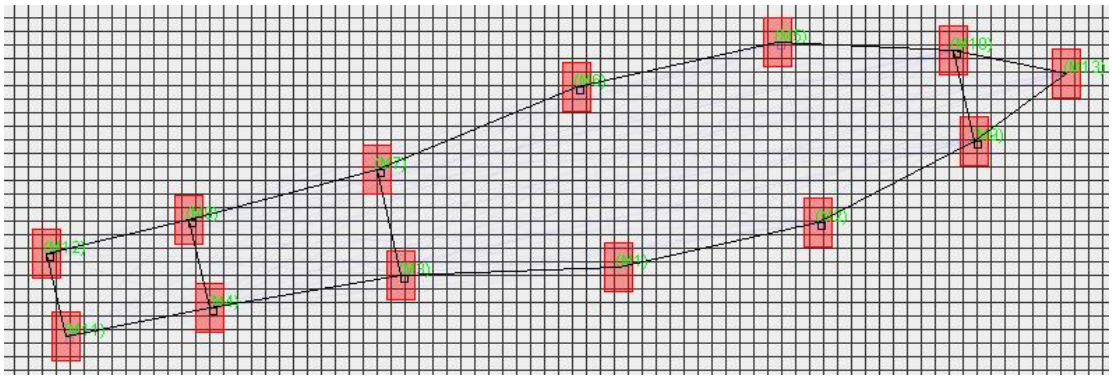
Figure 4.9. Modified hexagonal placed blocks

4.3.3. FDP Applied to Placement

Applying FDP to placement problem by just using transistors as blocks and connections as strings yields unacceptable results. Results are far from being satisfactory, direct application of FDP is given in Figure 4.10. Places of transistors are satisfactory for the differential amplifier. However placement of transistors with the same method is not acceptable for the differential cascode amplifier circuit. There is an interesting point in layout of cascode circuit, although it is very hard to find symmetries in the circuit, FDP algorithm automatically shows this symmetry.



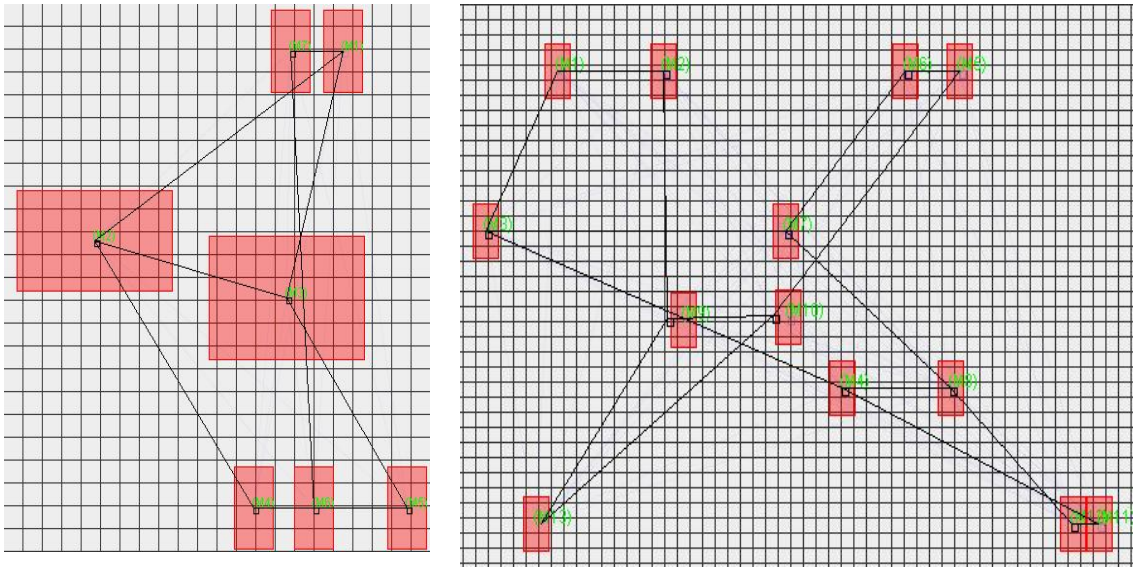
(a)



(b)

Figure 4.10. Circuits placed using FDP. (a) Simple diff. amp., (b) Diff. cascode amp.

An improvement to the algorithm is necessary, the first improvement tried was to add some constraints. Transistors that should be on the same level did not turn out to be on the same level. Gate to gate connected transistors are constrained to be on the same level. Transistors connected to the supplies are also constrained to be on the same levels. This way diff cascode circuit shown above wouldn't be that long and transistors wouldn't seem to be scattered. Adding constraints improved the layout of diff. amp.. However, for a more complex circuit; diff cascode, results are not that good. Outputs of Improved algorithms are depicted in Figure 4.11.

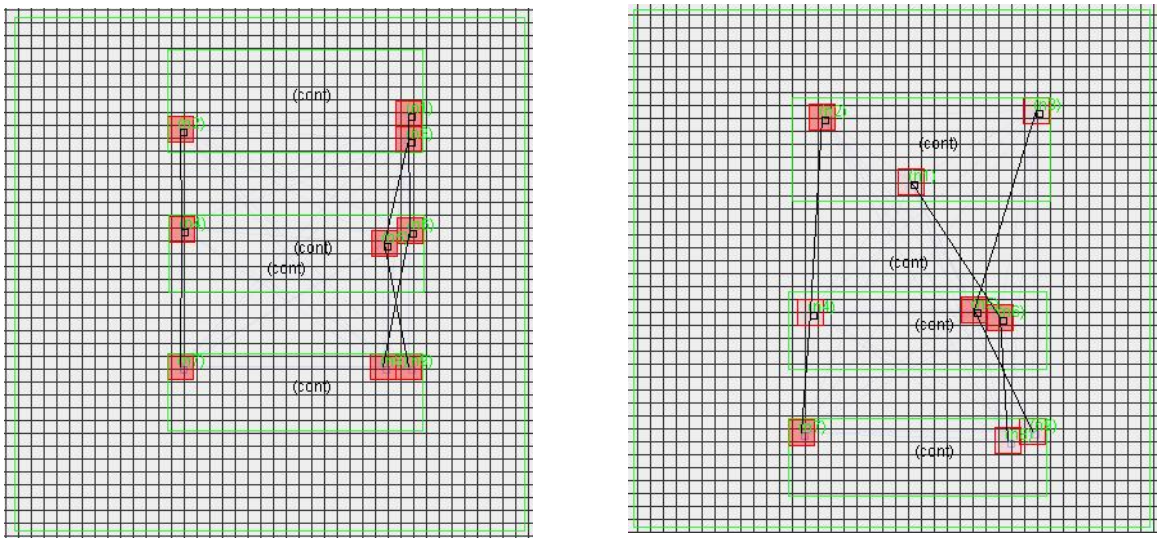


(a) (b)

Figure 4.11. Results of modified FDP algorithm. (a) Simple diff. amp., (b) Diff cascode amp.

Differential Amp. circuit shown above is satisfactory, reducing repel constant will make the layout compact. Only thing that should be corrected is the middle transistor (M6) at the bottom, it should be on the leftmost side of the bottom row. Diff cascode circuit is not satisfactory, reducing repel constant will compact the layout but there are more important problems. Layout is not as long as the unconstrained layout but, there is no symmetry any more, moreover the positions of some transistor pairs should be exchanged. The constraint that keeps common gate transistors on the same level does not allow them to exchange their places, an additional problem is created with this constraint; ordering of transistors from left to right. Another problem that constraint introduction causes is; force directed placer is not stable any more, blocks tend to oscillate or go out of scope.

Instead of keeping common gate transistors in the same level, a new improvement is made in order to overcome shortcomings of previous improvement. Transistors are grouped according to partitioning data obtained from partitioning step, these groups are placed in container blocks that are not actually blocks but confine the blocks to a limited area. This way, transistors that should be close wouldn't go apart. Results are shown below.



(a) (b)

Figure 4.12. Grouped placement. (a) Using no transparent blocks, (b) Using transparent blocks

Initial sizes of container blocks are set to be wide enough to allow the blocks to exchange their places. This is not enough; keeping container wide does not guarantee a good result. Next try was to make some of the blocks transparent letting them go through solid blocks so that ordering problem would be solved. This method introduces a new question; which blocks should be transparent so that final placement would be the best? If transparency is over used, transparent blocks may overlap. A result of transparent placement example is shown in the figure above. Some of the blocks overlap with other blocks and it is hard to separate them.

Improvements tried for placement so far introduced their own specific problems. None of the improvements could alleviate the problem; they could just transform the placement problem to another problem where solution is still missing. A new method is necessary in order to get a good layout in a reasonable time. An enumeration based placer is implemented as a temporary solution, it gives reasonable solutions in non-polynomial time.

4.3.4. Order Problem and Enumeration

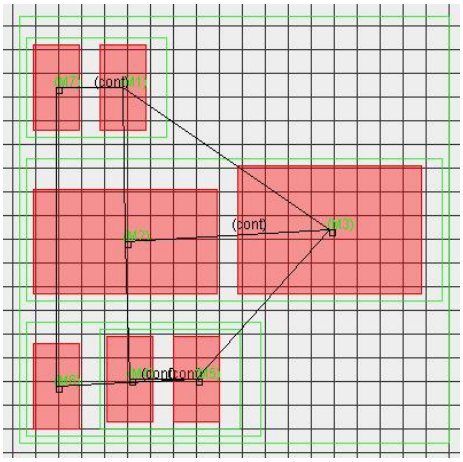
As discussed before, introducing constraints to blocks reduces their freedom to move around and ability to find their places. A result of this situation is disordered transistors. Correct order of these transistors should be found. Correct order of transistors is found by an inefficient but solution guaranteeing method; all combinations are tested and best one is selected.

Transistors are grouped as discussed before, their order is found by testing all combinations of their orders. Cost of each situation is tested using FDP, sum of forces between the transistors gives the cost. Best solution is the one that there is minimum total force between the blocks.

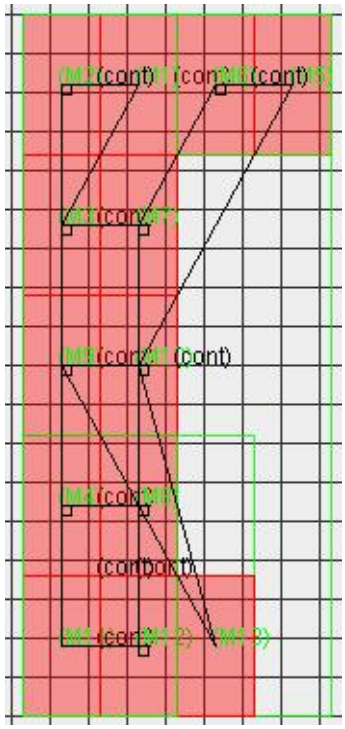
Table 4.1. Horizontal ordering combination counts

<table>
<tr><td colspan="3"></td><td>Number of combinations</td></tr>
<tr><td>M1</td><td>M2</td><td>M3</td><td>3!</td></tr>
<tr><td colspan="2">M4</td><td>M5</td><td>2!</td></tr>
<tr><td>M6</td><td>M7</td><td>M8</td><td>3!</td></tr>
<tr><td colspan="3">Total</td><td>3! . 2! . 3!</td></tr>
</table>

Table 4.1 shows a layout and corresponding combinations of each row in the layout. Since potential levels information is used to solve the vertical ordering problem, only the horizontal ordering problem remains. Number of combinations in a partition depends on the size of the group, big groups increases computational time to an unacceptable value. No iteration is done, blocks are placed to their initial positions according to their ordering and just total force is calculated. Although no iteration is done, algorithm is highly inefficient, because the simulation engine that is initialized once is not initialized for every combination. These initializations may be tens of thousand may be more. Enumeration algorithm is enough for small circuits, but highly inefficient for big circuits. Outputs of enumeration placement method are shown in the figure 4.13. Layouts look much better.



(a)



(b)

Figure 4.13. Outcomes of Enumeration algorithm. (a) Simple diff. amp., (b) Diff. cascode. amp.

5. ROUTING

Routing is the last step of this automation tool, generated and placed modules are electrically connected to complete the layout. Wiring has a great influence on performance of the circuit; increasing length of a wire increases its resistance and its capacitance. If wiring is not controlled properly, circuit will not work as desired. Performance of the router also depends on quality of placement.

Main job of the router is to connect the designated nodes. Routing is done in such a way that capacitance and resistance of the wiring is minimized. Moreover minimization of parasitics is not a local problem; overall parasitics of the wires should be considered, critical wires should be connected with the least possible parasitics. Another job of router is to minimize crosstalk effect between sensitive wires.

There are many graph models for routing and also many algorithms to route using those models. Performance of these models and algorithms depends on the application. Details about models and algorithms are given below.

5.1. Graph Models

Grid graph model is the simplest model, all layout is divided into squares with minimum side lengths (λ). Vertices that are occupied by placed modules are marked as not-free, other vertices are marked as free. Connection points of modules are start or end points of wires; wiring starts from a starting point and it is routed passing through the free vertices until the end point is reached.

This model gives best results but it is very time and memory consuming. It is not applicable to big layouts without some performance modifications. Even for a small layout, large amount of memory is allocated. For example a 1000x1000 sized layout requires; 10^6 vertices to be allocated in the memory. Increasing layout size increases the complexity

proportional to area of the layout. Timing performance of the router is even worse, increasing layout size not only increases solution space to explore but it also increases constraints due to placed modules and routed wires. Additional constraints increase iteration count.

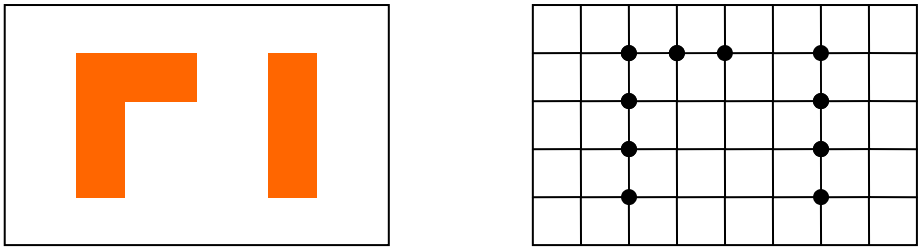


Figure 5.1. Grid graph model

Checker graph model is a more general model than the previous one. Layout is divided in rectangles according to modules. Side length of these rectangles are longer than the side lengths of squares used in previous model. This model is more efficient in terms of time and memory.

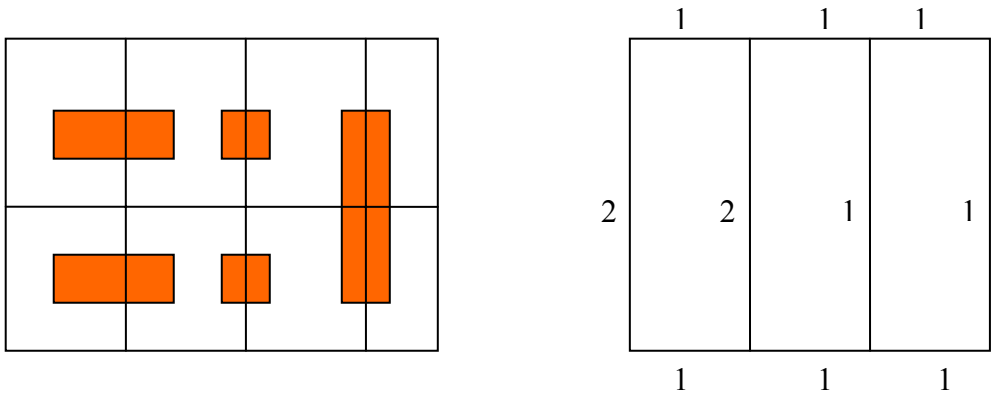


Figure 5.2. Checker graph model

Channel intersection model is the most general model. Channels are generated between the modules. Routing is performed using these channels as routing railways. Complexity of layout does not depend on layout size; it depends on module count.

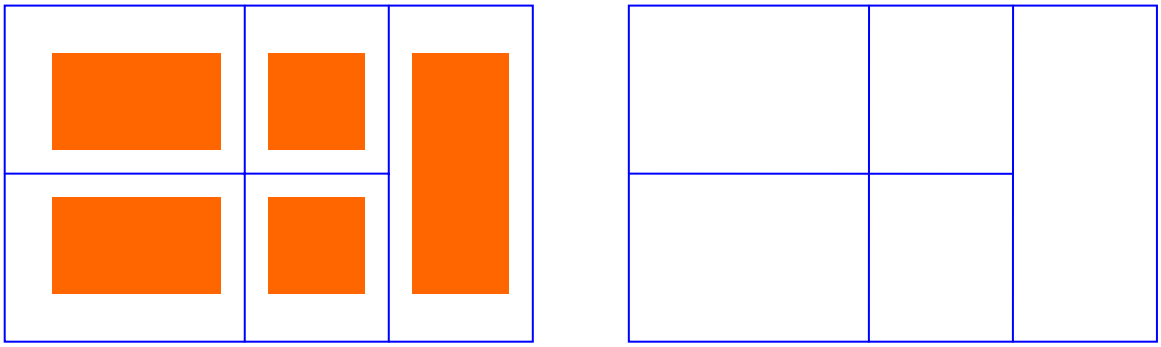


Figure 5.3. Channel intersection graph model

Intersection of the channels are vertices of the graph. A channel intersection graph is shown above. Pins of the modules are projected on the edges of the graph. Problem is simplified to find the minimum and least overlapping path between the pin vertices.

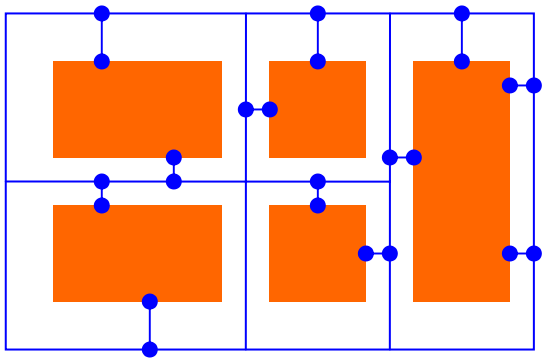


Figure 5.4. Projecting pin to the channel

Channel intersection model not only reduces memory usage, it also reduces iteration count due to reduced constraints. Iteration is required only for overlapping wires in order to find minimum overlapping wiring.

5.2. Algorithms

Maze routing is the most popular algorithm category for routing. Maze routing algorithms use grid-graph model. The first maze router algorithm in the literature is Lee's algorithm. It is simple but inefficient, thus leading to the development of many algorithms to overcome its shortcomings [29].

Although there are many maze router algorithms, they share common features. They use the grid-graph model and they label the grid with numbers starting from source vertex increasing the labels every step moving away from the source. Shortest path is found between two terminals, if there exists one. This algorithm is simple, but not the best; it has some shortcomings. Long routing times and large memory requirement makes it applicable to small sized circuits only

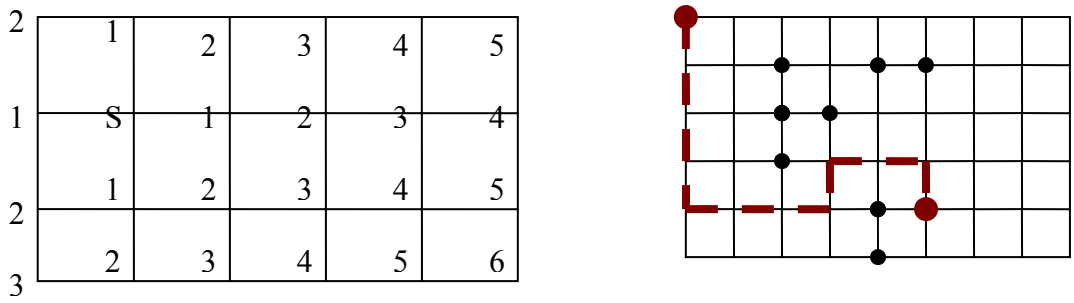


Figure 5.5. Lee's Maze Routing algorithm

Line-probe algorithm [30][31] uses lines (successive edges) to find the path, reducing memory usage and computation time. Lines are generated from the source and from the target. New lines are generated from escape points of these lines until lines originating from the source and the target intersect. The path found by line-probe algorithm may not be the shortest path.

There are two more routing algorithms; shortest path based algorithm [4] and Steiner tree base algorithm.

Shortest path algorithm is based on Dikjstra's shortest path algorithm [32]. Given the graph, shortest path between two terminals is found. Time complexity of the algorithm is proportional to square of vertex count.

Wiring is not point to point usually, there are more that two vertices that should be wired together most of the time. Using point to point connection algorithms is not convenient for multiple vertex connection. Steiner tree based algorithm [4] solves this problem. Multiple vertex terminals are connected efficiently using this algorithm.

5.3. Routing Algorithm Implemented in This Tool

In this work, channel intersection model is used, due to its simplicity. Instead of a single algorithm, a combination of routing algorithms are used, implemented routing algorithm is explained in the following section. Routing problem is divided into two; projection of pins of the module to the channel intersection graph and routing of these nodes using the routing algorithm.

Figure 5.4 is an illustration for the first step; projection of pins to the channel. Nearest point to a pin is selected on the channel to be the projection of the pin. Electrical layer of the modules are used for determination of positions of the pins. Electrical layers of modules were discussed in module generation section. If routable side's length is longer than a minimum contact width, equally spaced pins are projected onto the channel in order to find the best place to route. First step simplifies the problem, second step routes these projected pins.

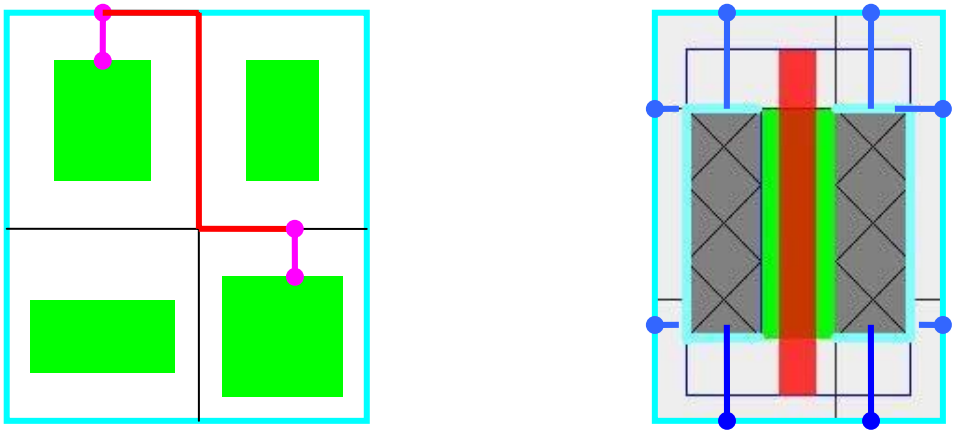


Figure 5.6. Routing the modules. (a) A point to point route, (b) Projection of pins of a simple MOS to the channel

5.3.1. Point to Point Routing

Elementary step of routing is point to point routing. Source and target vertices are connected using the simplest algorithm; direction of target is determined with respect to source, more distant direction is selected to move. Moving direction is changed when target becomes more distant in the other direction. Point to point connection is shown in Figure 5.7. Since there is no obstacle, algorithm is not iterative for point to point connections. Process works very fast, memory consumption is minimal.

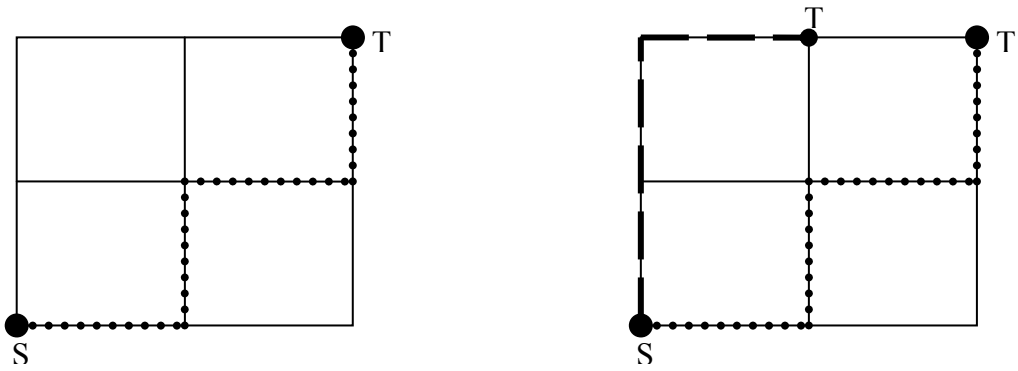


Figure 5.7. Two node point to point routing.

5.3.2. Multiple Node Point to Point Routing

Multiple wire routing leads to overlap problem. Nodes should be connected in such a way that wires that belong to different nodes should not overlap if possible. For example, Figure 5.7 shows how the second connection is done. The second connection (dashed) does not intersect first connection (dotted). The third wire cannot connect source to target without overlapping. Thus, it will be placed with minimum overlap

5.3.3. Single Node Tree Routing

The problem in routing is to make connections which are usually not point to point connections. There may be more than two points to be connected. Instead of Steiner Tree [4] based algorithm, which is NP complete, a modified version of point to point routing algorithm is used.

The problem is illustrated in Figure 5.8. S, T_1 and T_2 are to be connected to each other. First S and T_1 are connected and then T_2 is connected to (S, T_1) using the shortest path. Connection order of vertices is random; two of the vertices are connected first, then vertices are popped from the vertex list and routed to the previously routed vertices. First step is to connect S to T_1 using point to point connection in this example. Then, vertices on the route are tested to find the closest one to the next target, T_2. In this case, 2 and T_1 are nearest vertices to T_2. "2" is selected to route T_2, resulting figure is Figure 5.8.

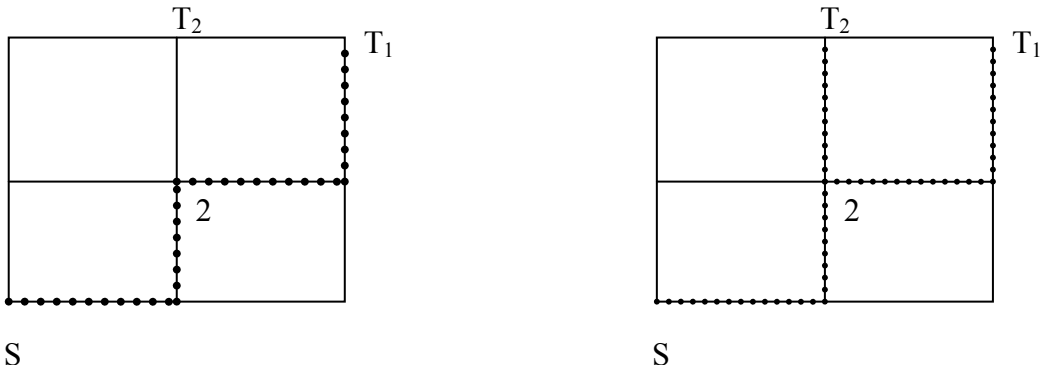


Figure 5.8. Single node three vertices routing

5.3.4. Multiple Node Tree Routing

Last step is to route all nodes using single node tree routing algorithm for each node separately. Each time overlap amount of new route is checked with previously routed paths, minimum overlapping route is selected. Overlap amounts are stored for each vertex in order to arrange appropriate wiring distance between the modules.

6. CONCLUSION AND FUTURE WORK

6.1. Conclusion

In this work, an analog automation tool (ALGv3) is presented. This tool consists of four steps; module generation, partitioning, placement and routing.

Module generator is capable of generating transistors. Not only simple transistors but also complex structures such as merged, interdigitized and common centroid transistors are also generated. Performance criteria are also considered by generator; matching, parasitic and aspect ratio values are evaluated in a cost function in order to find the optimum generation parameters for transistors.

First method used in partitioning is to divide the transistors in two groups, which is extensively used in digital automation. Using only this method does not give satisfactory results especially for big circuits. Information extracted in this method is not utilized in this automation tool due to complexity of combining two partitioning techniques employed in this tool. Using this valuable information, probably in next version of ALG, partitioning will be even powerful.

Second method used in partitioning is to group well known structures such as current mirrors and differential inputs in order to minimize complexity of partitioning and placement.

Placement has two options: automatic and manual. Force directed placement is implemented for automatic placement. Using only FDP did not yield satisfactory results; it was modified to give better results. Information extracted in partitioning and potential levels information is utilized. Unfortunately even this is not enough; enumeration is done in order to find the correct placement order of transistors, which makes the automatic placement process inefficient. It works with small circuits, though it is not convenient for

big circuits. Second option of placement is manual placement. Manual placement is done according to directives of the user.

Routing stage uses channel intersection model and a simple maze router algorithm. Due to model and implemented algorithm, routing works very efficiently. A simple point to point routing algorithm is used iteratively to route all nodes with minimum overlap. Routing stage does not route the modules actually, it is done in data structures, but it is not implemented for doing real routing using layers due to lack of time. Routing step is not difficult when isolated from other steps, but it cannot be isolated. It requires interaction with module generator and palacer. This interaction mechanism makes routing step difficult to design.

6.2. Future Work

Existing features of ALGv3 were described until now. In this section, shortcomings or features that will be implemented in next version will be discussed.

6.2.1. Module Generation

Technology supported in this tool is 0.25μm. Other technologies will be supported in next version. Minimum contacts for a connection are generated in this version, current rate controlled contact number feature will be implemented. Modules can be generated using two groups of parameters; first group is parameters that are described up to here (number of folds, number of quads,…), second group is parameters that are fed back from placer such as height, width and aspect ratio. An interaction loop between module generator and placer may increase the quality of the layout.

There should be a strong interaction between module generator and router. Electrical layer and active regions model used in module generation gives sufficient information to the router, this information will be updated after each routing step of router.

Interaction between these two is not just for routing, it can be used for parasitic minimization also.

6.2.2. Partitioning

Partitioner uses two algorithms as explained before; partitioning algorithm borrowed from digital automation and topology matching algorithm. Both of them give valuable information but information extracted using first algorithm is not utilized due to complexity of combining the outcomes of two different algorithms. Exploiting outputs of the first algorithm will increase the quality of the generated layout.

6.2.3. Placement

Manual placement section is almost complete, but improvements should be made for automatic placer. Enumeration method used in placement step makes placement inefficient for big circuits. This enumeration algorithm should be enhanced to work faster or a completely different method can be implemented such as simulated annealing or genetic algorithm.

Placing modules according to their potential levels information decreases the complexity of the problem, but it also diminishes the solution space. Solution space that is omitted should also be considered.

6.2.4. Routing

Routing is not complete yet, it will be implemented with a strong interaction between module generator.

Routing will be divided into two groups; local routing and global routing. Local routing is for individual modules and global routing is for inter module routing. This way computation effort and memory usage will be minimized.

APPENDIX A: USER DIRECTIVES

Providing a directives file, user can control the steps of this automation tool. This feature is implemented in order to give user more control on automation. User directives feature is actually for manual placement option, but it is enhanced to manipulate module generator also.

There are three main sections in directives file; partitioning, structure specific derectives and placement sections.

A.1. Partitioning

Partitioning section is used to put the transistors in groups of hierarchy. Declaration of partition section starts with ".part". Each partition is written on a different line. Declaration line, starting with ".part", can be used also to define a partition. An example is shown below.

```
.part   p1      M1      M2
        p2      M3      M4
        p3      p1      p2
```

First item on each line is the name of the group being defined, while rest of the items are elements of this group. Three lines that are shown above generates a hierarchy, where p3 is the root of the hierarchy. Hierarchy is shown in Figure A.1. All the transistors in the circuit are grouped in this manner. Lines are parsed to generate new partitions until a new section is defined.

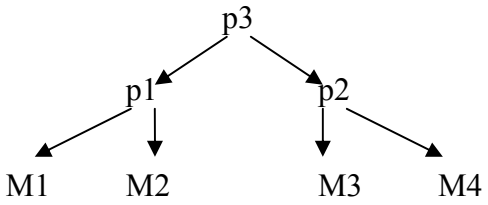


Figure A.1. Partitioning tree

A.2. Structure Specific Commands

These commands are merge, folds, inter and quad. These commands are used to generate complex structures and to set their properties.

A.2.1. Folding

This command sets number of folds of individual transistors. An example is given below.

```
.folds  M1      n=2
        M2      n=5
        M3      n=1
```

Lines given above folds the transistors M1, M2 and M3 into 2, 5 and single folds respectively.

A.2.2. Merge

Transistors to merge are set using ".merge" command. First of all, transistors to merge are grouped in partitioning section. This group, for example "pMerge" is given as a command in merge section as shown below.

```
.merge          pMerge
```

A.2.3. Interdigitize

".inter" is used to generate interdigitized structures. A group name and properties of inter structure, such as number of interdigitization is supplied in the "inter." section as shown below.

```
.part   p1      M1      M2
.inter  p1      n=2
```

A.2.4. Quad

"quad" is used to generate common centroid structures. A group name and parameters of this quad are supplied in the quad section. An example is shown below.

```
.part   pQ      M1      M2
.quad   pQ      n=2     q=3
```

A.3. Placement Commands

Last Section is ".place" section. Orientations of transistors, groups of transistors and spacing between them are set in this section. Declaration of this section starts with ".place". Placement information of different transistors or partitions is given in separate lines. Orientation of transistors or a partition can be either horizontal or vertical. One of these orientations are appended to the line of interest.

```
.place  M1      horizontal
        p2      vertical
```

Lines shown above places M1 horizontally and p2 vertically. Default orientation is horizontal. Hierarchally grouped transistors form complex layouts using these simple directives. An example is given below.

```
.part   p1      M1      M2
        p2      M3      M4
        p3      p3      p1 M5 p2

.place  p3      vertical
```

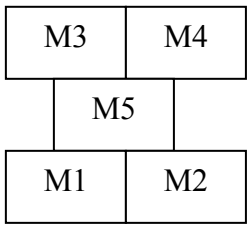


Figure A.2. Orientation example

Spacing between the modules is arranged using “padx”, ”padxleft”, ”padxright”, ”pady”, ”padytop”, ”patybottom” dirctives. These directives are appended to the line that defines placement properties of the desired transistor or partition.

Parameter “padxleft” sets the spacing of a module with its left side module. “padxright” is similar to “padxleft”, but it sets the right spacing of the module. “padx” parameter sets both “padxright” and “padxleft” to the same value. Other parameters works in the same manner, but they are related to “y” axis instead of “x” axis.

Alignment is used to fix alignment of partitions that are oriented in one direction but free on the other direction. For example a vertically placed partition can be aligned to left, center or to the right. User can select the alignment that fits most with the rest of the layout. Commands “alignleft”, “aligncenter”, “alignright”, “aligntop” or “alignbottom” are appended to the line that belongs to desired partition.

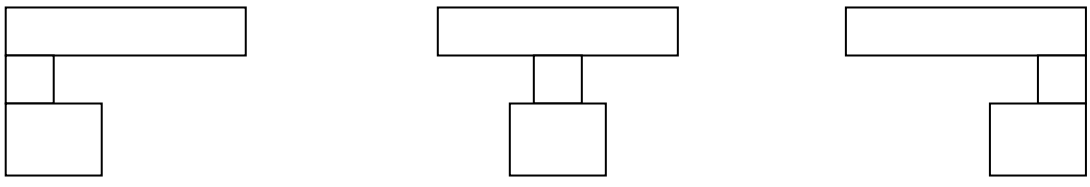


Figure A.3. Alignment options for vertical orientation. (a) Aligned to left, (b) Aligned to center, (c) Aligned to right

Figures shown above are for vertical orientation, same is applicable to horizontal orientation. Modules in a horizontally oriented partition can be aligned to top, center or to the bottom. Default alignment is center.

A complete directives file of a design is given below.

```
.part
        p1 M7 M1
        p2 M2 M3
        p3 M6 M4 M5
        p4 p3 p2 p1

.inter p2 n=4

.fold
        M1 n=4
        M2 n=5
        M3 n=5
        M4 n=4
        M5 n=4
        M6 n=4
        M7 n=4

.place
        p1 horizontal
        p2 vertical pady=20
        p3 horizontal
        p4 vertical
```

Figure A.4. Directives file

REFERENCES

1. Balkır, Sina, Günhan Dündar and A. Selçuk Öğrenci, *Analog VLSI Design Automation*, CRC Press 2003.

2. Lampaert, K., G. Gielen and W. Sansen, *Analog Layout Generation for Performance and Manufacturability*, Kluwer Academic Publishers, 1999.

3. Cohn, J. M., D. J. Garrod, Rob A. Rutenbar and L. R. Carley, *Analog Device-Level Layout Automation*, Kluwer Academic Publishers, 1994.

4. Sherwani, N., *Algorithms for VLSI Physical Design Automation*, Kluwer Academic Publishers, 1995.

5. Chachra, V., P. M. Ghare and J. M. Moore, *Applications of Graph Theory Algorithms*, North Holland, 1979.

6. Seshu, S. and M. B. Reed, *Linear Graphs and Electrical Networks*, Addison Wesley, 1961.

7. Rabaey, J. M., A. Chandrakasan and B. Nikolic, *Digital Integrated Circuits*, Prentice Hall, 2003.

8. Lakshmikumar, K.R, R.A. Hadaway and M.A. Copeland, "Characterization and modelling of mismatch in MOS transistors for precision analog design" *IEEE Journal of Solid –State Circuits*, vol. SC-21, pp. 1057-1066, 1986.

9. Pelgrom, M.J.M., A.C.J. Duinmaijer and A.P.G Welbers, "Matching properties of MOS Transistors" *IEEE Journal of Solid-State Circuits*, vol.SC-19, pp.1433-1440, 1989.

10. Drennan, Patric G. and C. McAndrew, “Understanding MOSFET Mismatch for Analog Design”, *IEEE Journal of Solid-State Circuits*, vol 38, no:3, 2003.

11. Bruce, J.D., H.W. Li, M.J. Dallabetta and R.J. Baker “Analog Layout Using ALAS!”, *IEEE Journal of Solid-State Circuits*, vol.31,no.2, 1996.

12. Gatti, U., F. Maloberti and V. Liberali, “Full Stackes Layout of Analogue Cells”, *ISCAS*, pp.1123-1126, 1989.

13. Cohn, J., D.Garrod, R. Rutenbar, and L. Carley, “KOAN/ANAGRAM II: New tools for device-level analog placement and routing”, *IEEE Journal of Solid-State Circuits*, vol.26,no.3, 1991.

14. Naiknaware, Ravindranath, “Automated Hierarchial CMOS Analog Circuit Stack Generation with Intramodule Connectivity and Matching Considerations”, *IEEE Journal of Solid-State Circuits*, vol.34, no.3, 1999.

15. Meyer, V.,”ALASYN: Flexible Rule-Based Layout Synthesis for Analog IC’s”, *IEEE Journal of Solid-State Circuits*, vol. 28, no. 3 , march 1993.

16. Degrauwe, M. G. R., O. Nys, E. Dijkstra, J. Rijmenants, S. Bitz, B. L. S. G. Goffart, E. A. Vittoz, S. Cserveny, C. Meixenberger, G. Van Der Stappen and H. J. Oguey, “ IDAC : An Interactive Design Tool for Analog CMOS Circuits”, *IEEE Journal of Solid-State Circuits*, vol. 22, no. 6 , December 1987.

17. Rijmenants, J., J. B. Litsios, T. R. Schwarz and M. G. R. Degrauwe, “ILAC : An Automated Layout Tool for Analog CMOS Circuits”, *IEEE Journal of Solid-State Circuits*, vol. 24, no. 2, April 1989.

18. Zhang, L., U. Kleine, R. Raut and Y. Jiang, “Aladin: A Layout Synthesis Tool for Analog Integrated Circuits”, *Analog Integrated Circuits and Signal Processing*, 46, 215-230, 2006.

19. Quinn, N. R. and M. A. Breuer, “A Forced Directed Component Placement Procedure for Printed Circuit Boards”, *IEEE Transactions on Circuits and Systems*, vol. cas-26, no. 6, June 1979.

20. Aluppaei, S. and S. Katkoori, “Net-Based Force-Directed Macrocell Placement for Wirelength Optimization”, *IEEE Transactions on VLSI Systems*, vol. 10, no. 6, December 2002.

21. Van der Plas, G., G. Debyser, F. Leyn, K. Lampaert, J. Vandenbussche, G. G. E. Gielen, W. Sansen, P. Veselinovic and D. Leenaerts, “AMGIE - A Synthesis Enviroment for CMOS Analog Integrated Circuits”, *IEEE Transtactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 9, September 2001.

22. Lai, Y. T., Y. C. Jiang and C. C. Kao, “DTA: Layout Design Tool for CMOS Analog Circuits”, *IEEE Asia-PAsific Conference on Circuits and Systems*, pp. 537-540, December 2004.

23. Kayal, M., S. Piguet, M. Declercq and B. Hochet, “SALIM: A Layout Generation Tool for Analog ICs”, *IEEE Custom Integerated Circuits Conference*, pp. 751-754, 1988.

24. Owen, B. R., R. Duncan, S. Jantzi, C. Ouslis, S. Rezania and K. Martin, “Ballistic: An Analog Layout Language”, *IEEE Custom Integrated Circuits Conference*, pp. 351-354, 1995.

25. Harvey, J., M. I. Elmasry and B. Leung, “STAIC: A Synthesis Tool for CMOS and BiCMOS Analog Integrated Circuits”, *ISCAS*,2004

26. Kernighan, W. and S. Lin.,”An efficient heuristic procedure for partitioning graphs”, *Bell System Technical Journal*, 49:291-307, 1970.

27. Fiduccia, C. M. and R. M. Mattheyses, “A linear-time heuristic for improving network partitions”, *Proceeding of IEEE International Conference on Computer-Aided Design*, pp 414-417, 1986.

28. Goldberg, M. K. and M. Burnstein. “Heuristic improvement technique for bisection of VLSI networks”, *Proceeding of IEEE International Conference on Computer Design*, pp 122-125, 1983.

29. Lee, C. Y., ”An algorithm for path connections and its applications”, *IRE Transactions on Electronic Computers*, 1961.

30. Mikami, K. and K. Tabuchi, “A computer program for optimal routing of printed circuit connectors”, *IFIPS Proc*., H47:1475-1478, 1968.

31. Hightower, D. W., “A solution to the line routing problem on a continous plane”, *Proc. 6th Design Automation Workshop*, 1969.

32. Dijkstra, E. W., “A note on two problems in connexion with graphs”, *Numerische Mathematik*, 1:269-271, 1959.

33. Gajski, D. D. and R.H Kuhn, “New VLSI Tools”, *Guest Editors Introduction IEEE Computer*, 16(12):11-14, December 1983.