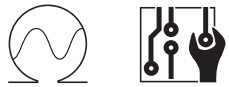


Technische Universität München
Department of Electrical Engineering and Information Technology
Institute for Electronic Design Automation

Implementation of Routing in Plantage

Bachelor Thesis

Benedikt Schmidt

Supervisor :	M.Sc. Pang-Yen Chou
Supervising Professor :	PD Dr.-Ing. Helmut Gräb
Topic issued :	02.05.2013
Date of submission :	00.10.2013

Benedikt Schmidt
Mitterweg 63
6020 Innsbruck
Austria

Abstract

The aim of this bachelor thesis is to implement the routing of analog circuits to the already existing placer Plantage. Ahead of the actual implementation the placer must be modified to fit the needs of a router. Afterwards the known routing algorithms for analog circuits will be compared and the best fitting solution selected and implemented. For a comparison, the resulting software will be applied to some examples and the results will be discussed.

Contents

1. Introduction

Currently a lot of new chips are designed and manufactured every single day. A lot of them are mainly digital, as our modern world becomes more and more digital as well. But some parts of these digital chips are still analog, even if it is not desired. The problem is that the surroundings of a chip will always be analog, so at least some circuit elements must also be viewed as analog. The analog world still lacks the tools to fully automatize the steps from an idea to the actual layout of a chip for manufacturing. Some tools for placing and some implementations for routing algorithms are available. During the last few years a completely new approach for the placement was developed, the hierarchical placer called *Plantage* (Strasser et al. 2008), and now it is time to implement the next step towards a fully automatized analog layout tool into this software: routing.

The actually last missing step will be the integration of the already developed automatic circuit analysis. This tool could automatically generate the constraints from the schematic and therefore increases the degree of automation. The user will certainly still have to modify these suggested constraints but the main work should already be done by the tool.

As we already have a fast and reliable placer, I have implemented the routing into this tool. The main results of this thesis are:

- a line expansion router, which creates, in most cases, design rule clean routes and considers symmetries
- based on the previous router an extended version, which tries to reduce the parasitics
- a software design, where the routing algorithm can be exchanged easily

2. Definitions

2.1. Module

One module in Plantage is just a rectangular shape. From that point of view, all circuit elements can be seen as modules, transistors, capacitors and so on. For the placement only width and height of the module are important. The actual dimensions are stored in the variant of the module. Every module can have various different variants, for example, with the same electrical behavior, but different numbers of gates (Figure 2.1).

Also a module typically has some pins, which are named and, additionally, have a net name as well, according to the net they belong to. One pin can have some contact areas, but must have at least one. Every contact area is only a rectangular shape, which has a height and width and is in a relative position to the module position.

2.2. Group

In Plantage there are two different types of groups. One type contains only groups and the other one only modules (Figure 2.2). At the moment it is not possible to mix modules and groups in one group. This is important because slightly different algorithms must be applied for every type of group.

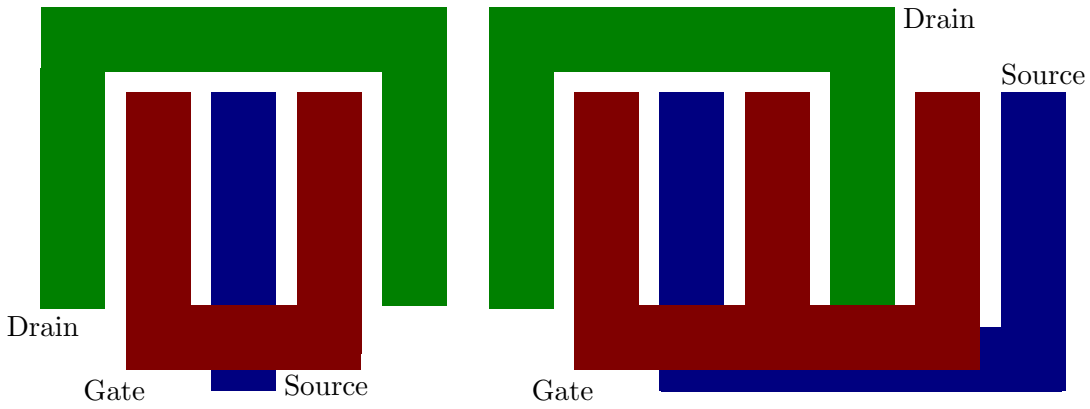
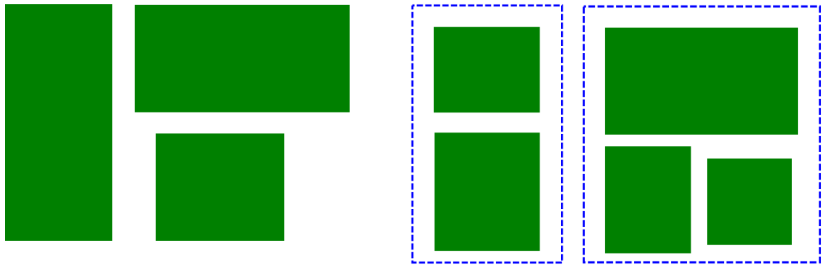


Figure 2.1.: the same MOSFET with two different numbers of gates



(a) group of single modules (b) group of two groups

Figure 2.2.: groups arrange modules and groups together

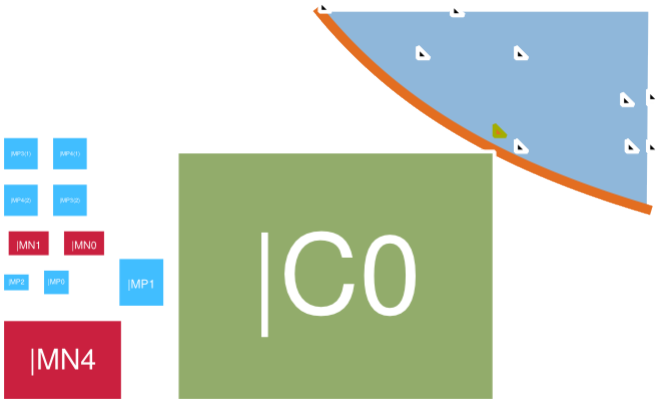


Figure 2.3.: shape function with possible placements for a miller amplifier

2.3. Shape Function

Actually a shape function is usually defined as the minimum height given as a function of the width. In this case Mr. Strasser used this term slightly different, in Plantage a shape function is basically a list of possible layouts for a given circuit. This list is called shape function (Figure 2.3) because the minimum possible area for these placements is a hyperbolic function and the placements located near this forms a *Pareto front*.

2.3.1. Shape

A shape is the layout for a certain couple of modules. On the one hand, it is used for the layout of the whole circuit but, on the other hand, also for the layout of circuit parts.

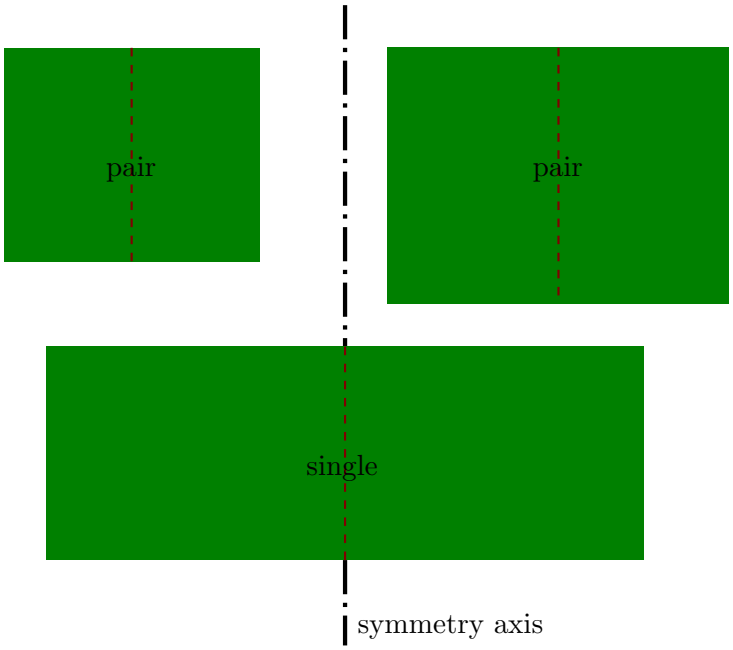


Figure 2.4.: horizontal symmetry constraint with one pair of modules and one single module

2.4. Constraints

In the design of analog circuits many constraints have to be met. They mainly consider that certain elements of the circuit should have similar electrical characteristics as far as possible. The most prominent example for such a pair of modules are the parts of a differential pair. Differences in these transistors result in a bigger offset of the resulting amplifier, which should be as low as possible.

2.4.1. Symmetry Constraint

Symmetry constraints define a vertical or horizontal symmetry axis for modules. They can contain two different types: single modules and pair modules. Single modules are placed self-symmetrically to the axis, pair modules are pairwise symmetrical. For the symmetry here, only the centers of gravity are considered, therefore pair modules can have different shapes. (Figure 2.4)

2.4.2. Alignment Constraint

An alignment constraint defines a relative position between modules, vertical or horizontal (Figure 2.5). In the terminology of Plantage you always have an A- and a B-module. In the case of a horizontal alignment the A-module is placed left to the B-module, in the other case

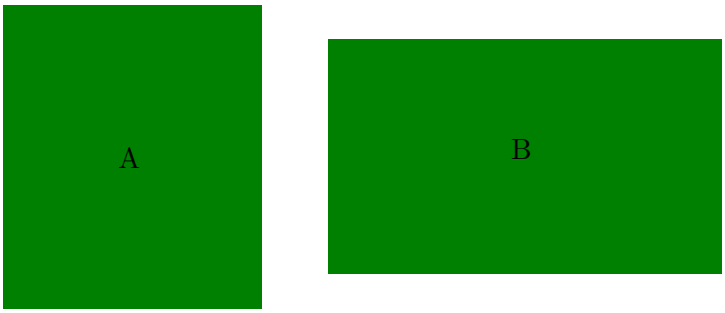


Figure 2.5.: horizontal alignment constraint

of a vertical alignment the A-module is placed below the B-module. Additionally, it is possible to define a minimum distance between the modules.

2.4.3. Group Symmetry Constraint

The group symmetry constraint is nearly the same as the symmetry constraint. The only difference is that it can be applied to groups instead of modules. Like the normal symmetry constraint we can have single groups, which are self-symmetrically placed, or pair groups, which are pairwise symmetrical to the symmetry axis.

2.4.4. Group Alignment Constraint

Once again the group alignment constraint is nearly the same as the alignment constraint where the only difference is that it can be applied to groups.

2.4.5. Same Variants Constraint

The same variants constraint is very important too, for example, especially for differential pairs. To get the same characteristics in one placement, as far as possible, these modules should have the same variants. They can still have various different variants, but in one certain placement for these modules the same variant is selected.

The implementation of this constraint is based on matching the names of the variants. Therefore it is still possible that the variants differ, for example the variant for one module is mirrored to the ones of the other part of a pair. This is very useful, as it gives the user a lot more flexibility.

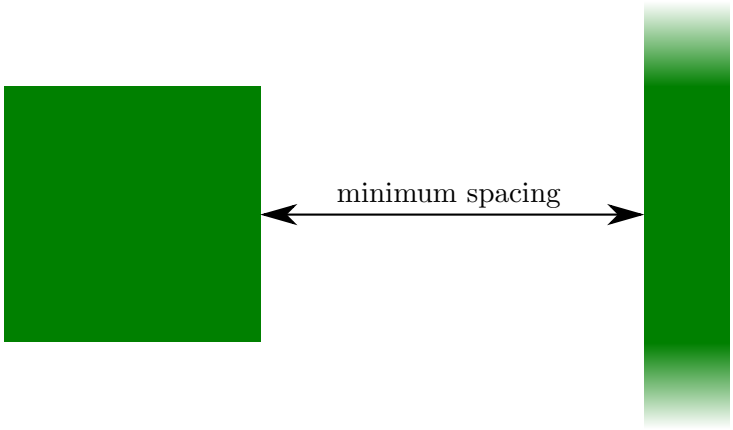


Figure 2.6.: minimum spacing rule

2.5. Technology Rules

Typically, the manufacturer of the chip provides a so-called *tech-file* for the circuit designer, which can be used to create the layout. This information includes, for example, the minimum distances between two modules. As the knowledge of the manufacturers is very important to them the technology-files are kept secret, and therefore no real standard for these files exists. The following descriptions are based on a technology file from Austria Microsystems (AMS), which was available for this project. I assume that other possible formats can be transformed through the ICFBInterface into something similar. The rules were already extracted through a skill script from Cadence but they were not stored or looped through to Plantage.

The technology rules can have four different types:

- minimum width
- minimum spacing
- minimum notch
- minimum enclosure

The first two rules, considering minimum width and spaces, have one value, the actual minimum width or distance, and the layer, which they should be applied on. The last two types also have a value but, additionally, two layers on which they must be considered. All the rules must be applied to the placement and the routes.

The minimum spacing rule (Figure 2.6) is quite easy to understand, as it only describes a minimal distance between two shapes on a certain layer. These rules may not be considered if deep trench isolated transistors are concerned, but this does not affect the routing. Deep trench

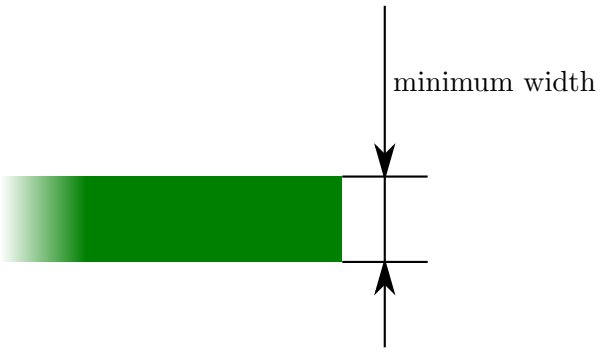


Figure 2.7.: minimum width rule

isolated transistors only result in a possible bigger, combined, forbidden area for routes.

Minimum widths are very central technology rules for the routing (Figure 2.7). These values are usually used as width for the routes. Only in special cases, if for example a high current must be transported, the designer may choose a bigger width. Nevertheless, the rules can be different for every layer.

The minimum enclosure rules describe minimum distances, which need to be met between shapes on different layers. In most cases a technology file contains only minimum enclosure rules, where one of the considered layers is a via-layer. A via-layer is an additional layer during manufacturing, in which the connections between the different layers are created. For the routing these rules result in the dimensions of the vias on the layers, which they connect.

For some Via-Definitions, for example in the Hit-Kit of AMS, some enclosure rules are missing, at least in the technology file, but for the design rule check these rules are still existing. In these cases, I have to assume that there is a certain default value for the minimum enclosure rules: 0.2μm to the layer below and 0.15μm to the layer above a via-layer. These values work at least for the technology definition of AMS, but they may differ in the definition of other manufacturers.

Every via-definition must have two minimum enclosure rules, one for every layer, which is connected through the via. Additionally, we need one minimum width rule for the via-layer, which describes the width of the actual connection. From these three rules we then can calculate the dimensions of the via, which typically differ for the two layers (Figure 2.8).

$$\text{via dimension} = 2 \cdot \text{minimum enclosure} + \text{minimum via width} \tag{2.1}$$

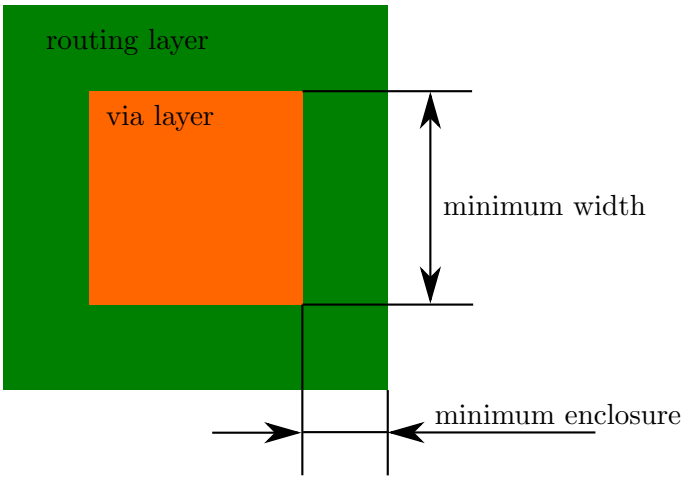


Figure 2.8.: minimum width rule

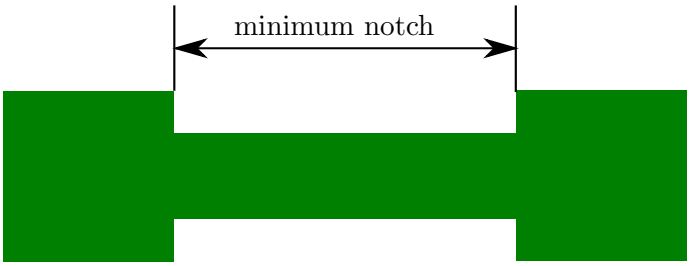


Figure 2.9.: minimum notch between two vias

Last but not least, there are several minimum notch rule definitions available in the technology definition of AMS. They are mostly used for the internal layout design of, for example, a transistor. During the routing they define, for instance, a minimum distance between two vias, connected by a route, as this construction forms a notch (Figure 2.9).

2.6. Electrical Rules

In the tech-file from the manufacturer there also are electrical rules implied. These rules describe the exact electrical behavior and can be used, for example, for a simulation of the whole circuit. In the routing algorithms, that will be described later on (Section 4.3) I will use these values to optimize the behavior of the circuit layout. Therefore, I want to describe how they can be interpreted.

The first important value is the sheet resistance. This value defines the resistance of a layer and I use it to calculate the resistance of the routes. I do not need an exact resistance, only a tool to measure and compare, which takes the different layers into account. Therefore, my personal definition for this algorithm of the sheet resistance will possibly be not the right one. But from my point of view it is useful in this case. For routes on one layer I just multiply the sheet resistance value with the length of the route, as I always use the minimum width for routes. A via does not really have a length, so I multiply the value with the area. Afterwards, these two results are summed up for a whole connection and this then represents the resistance of this connection.

The other two used values are the edge- and area-capacitance. The edge-capacitance illustrates the capacitor between two neighbor routes on the same layer, whereas the area-capacitance describes the capacitor between two routes on the same position but on different layers. The total capacitance (C_e and C_a) is calculated from the length where two routes are neighbors (l), the distance between them (d) in case of an edge-capacitance, the width they overlap, (w) in case of an area-capacitance, and the values from the technology file (C'_e and C'_a).

$$C_a = C'_a \cdot l \cdot w \tag{2.2}$$

$$C_e = C'_e \cdot \frac{l}{d} \tag{2.3}$$

All in all, the aforementioned definitions are most likely not the right ones but the results are multiplied with adjustable weights afterwards anyway. Therefore, the exact values are not a matter of interest; the results are only used as rough estimations and measurement.

3. Placement

The algorithm for the placement, a deterministic analog circuit placement algorithm using hierarchically bounded enumeration and enhanced shape functions (Strasser et al. 2008), was mainly developed and implemented by Martin Strasser in his doctoral thesis as well as by Michael Eick during his diploma thesis. The main idea behind it is that circuits typically have a certain hierarchy. For example, an operation amplifier (Figure 3.1) consists of a differential amplifier, a current mirror and an output amplifier. Usually it is a good idea to place the modules of the same group not too far away because they belong together in the functionality of the circuit. This is also the typical way a designer creates a layout out of a circuit manually. The designer starts with a certain functionality and places the modules for this function close to each other. If the modules are grouped together in the layout (Figure 3.2), they do not see that large differences in, for example, temperature of the substrate. This will result in a better performance of the circuit. At the moment, this information on the groups must be fed in by the user but the target for the future would be that even this step could be automatized by, for instance, a circuit analysis. Consequently, the information can be used to find the best placements for these groups, which are then combined on the next level in the hierarchy.

Additionally, it is possible to define certain constraints for modules and groups of modules:

- symmetry
- alignment
- group symmetry
- group alignment
- same variants

Some of these have to be considered by the routing, because, for example, symmetric modules typically should also have symmetric routes to avoid different resistances of the routes. Furthermore, if examined in more detail, symmetric routes should even have similar capacitive couplings with other routes.

The general structure of this application is divided into two parts:

- ICFBInterface
- Plantage

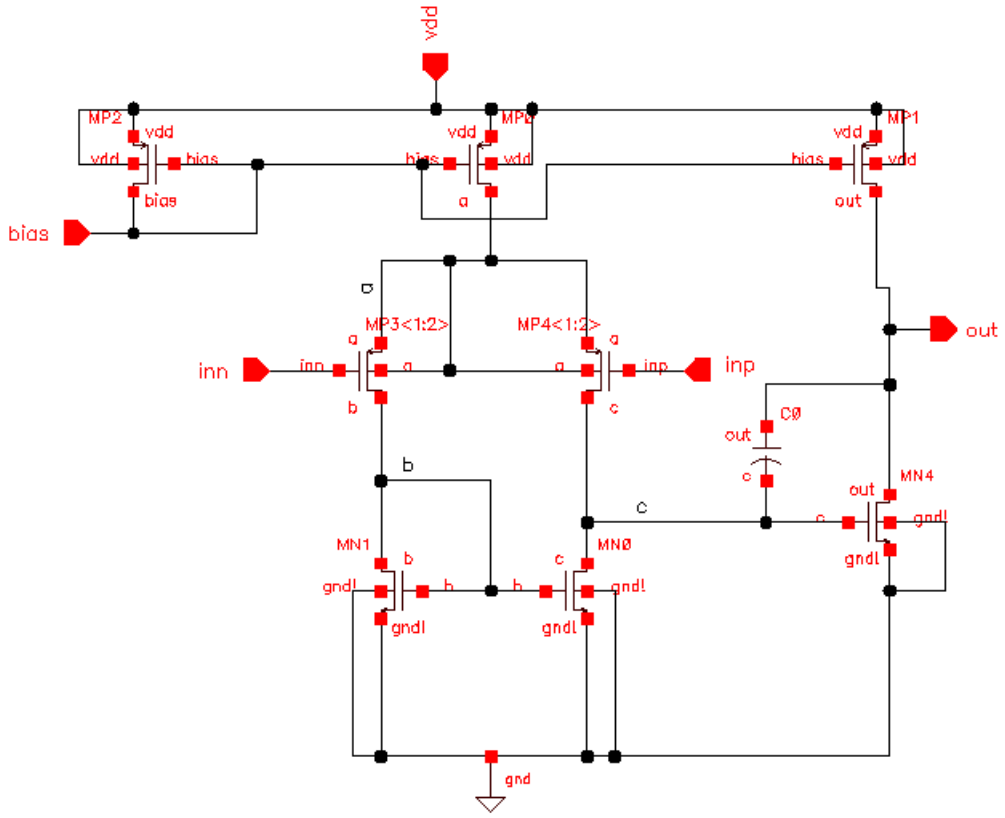


Figure 3.1.: schematic of a miller amplifier

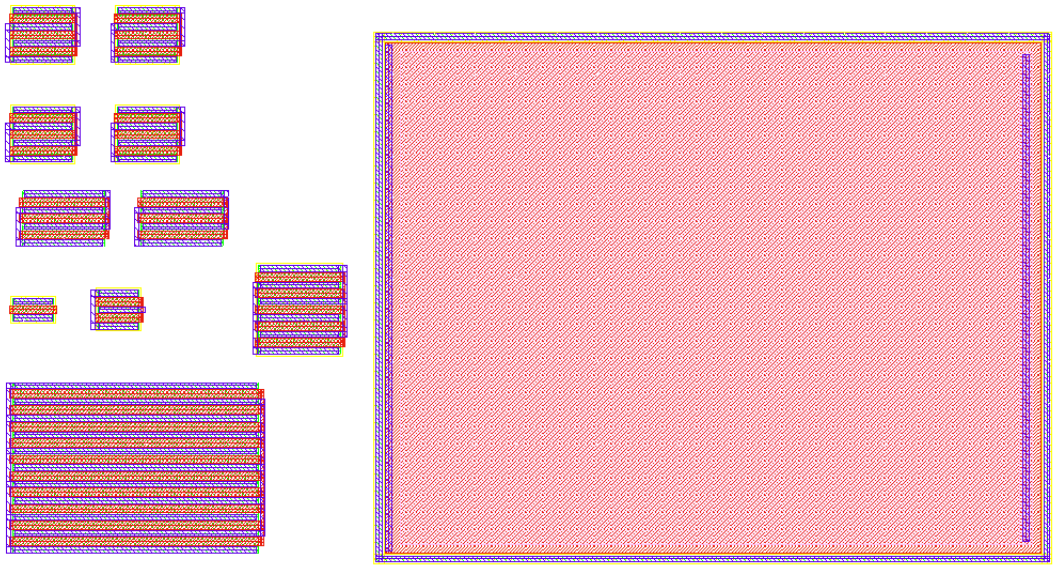


Figure 3.2.: possible layout for a miller amplifier

First, I will start with a short description of Plantage. It is a simple command line tool which accepts an XML-input-file and produces an XML-output-file. The input-file contains the basic information on the circuit, e.g. the modules with their various variants, group definitions, constraints and so on. The output-file of this tool contains a so-called *shape function* with different shapes. The shapes are the actual placements which contain the positions of the modules and the selected variants.

The ICFBInterface is, as its name already suggests, an interface between Cadence and Plantage. It receives, for example, the data of a circuit from Cadence. Based on this data, the users has to define groups and constraints, where the ICFBInterface supports them. This information is then sent to Plantage, which calculates several possible layouts. These layouts are returned in form of a shape function back to the ICFBInterface. There the results are displayed and the user can generate layouts in Cadence.

3.1. Algorithm

As already mentioned before, the placement is generated hierarchically. The starting point for the calculation of a placement is an enumeration sequence. This value is a very compact way to store the information of related positions for an accumulation of modules. It consists of a list of Boolean variables, usually stored as a string with the values E for East and N for North. The modules and selected variants for this enumeration are then stored in a B*-tree. On this data structure the actual calculation of the coordinates, considering the given constraints, is done. From the B*-tree several lines for a linear program are created. After this the linear program is solved by an external application, at the moment lp_solve (Eikland & Notebaert 08.09.2013). This tool solves the linear program and from the solution the actual coordinates for the modules can be calculated. This is usually done for some different enumerations. The list of possible placements, which is generated that way, is then filtered by the area. To reduce the computing time only a certain number of placements is then used on the next higher hierarchy level.

3.2. Software Architecture

In the following part only the software architecture of Plantage will be described, as this was the main part for the implementation of the routing. On the side of the GUI only some additional information needed to be displayed or manipulated by the user.

Plantage is a typical example for a grown code base. It contained a lot of different coding styles, had some half-implemented features and a lot of old and dead development branches, which were part of the production code. So my first task was to eradicate parts that were not in use anymore and unify the code base. The main problem was that the existing and working features should not be broken, what could have happened easily. For the beginning, I generated some test inputs, which covered the most important features. After every risky change I could generate the output from this data and compare it with the results before

each change. Through this practise I was able to slowly refactor the code and increase the encapsulation for some parts. This enabled me to write unit tests for these parts and reduce the risk of broken features even more. One by one, I applied this to nearly the complete code base. The result of these steps is a now well-tested and -documented code base, which can be changed and improved easily. Additionally, the basic structure of the architecture improved, or better said, an architecture was developed and applied to the code. Therefore, it became easier to understand what the different parts of the code actually do. This was also important for the implementation of a routing algorithm, as it also produced a single point, where the routing can be done.

4. Routing

Even nowadays the routing is often done manually for analog circuits. This is mainly caused by a lack of good tools and, consequently, the missing trust of engineers in them. The engineers consider a lot of different constraints during the routing, for example symmetries, net lengths and capacitive couplings. All these factors have a major impact on the performance of the circuit, as it is very important in the analog world to have an accurate signal level because a division into high or low is not enough.

Additionally, the designer adds guards and rings to the placement to protect certain devices from noise. This noise can come, for instance, from the clock net or other routes, which carry a signal with high frequency. But high frequencies are not the only disturbing effects: During the placement, the designer has to consider temperature gradients, caused by transistors with a high power consumption. To avoid problems with different temperatures in the substrate of, for example, the devices of a differential pair they can be placed symmetrically or can even have a common centroid. Around these grouped modules the designer then can create a guard ring to protect the devices even more. An example for a manually routed layout can be seen in (Figure 4.1).

So for an automatic routing algorithm a lot of different constraints must be considered. There are even more obstacles than just the devices and already created routes, for instance, the routes should pass over guard rings. To solve this problem in the following chapter two different approaches are presented: one simple sequentially line expansion router and one extended version of the previous one, which considers additional constraints. But first I will discuss the changes that were necessary for the implementation of the routing algorithms.

4.1. Changes for Routing in the existing Applications

4.1.1. Plantage

Missing Data

First of all, to actually route the placements some data was missing in Plantage. Although at some points the routing was already prepared, as, for instance, the net names were already available, important information was not given, including:

- actual dimensions of the modules

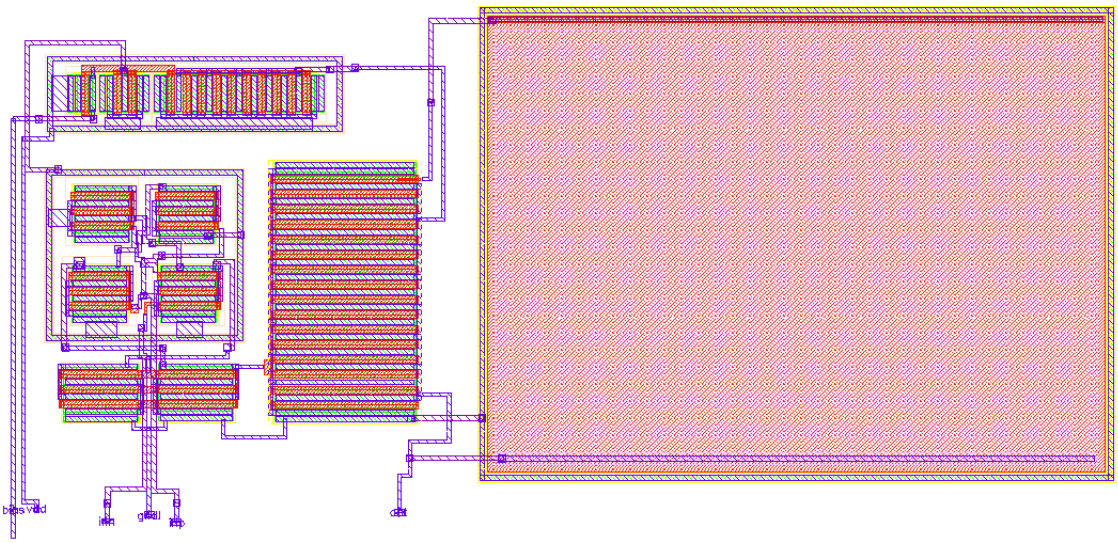


Figure 4.1.: manually routed layout of a miller amplifier

- parameter of the technology
- vias and routes
- guard rings

In awareness of the necessary routing for every module a bigger area was defined for the placement. The additional space between the modules could then be used to route the placement. Unfortunately, the information on the actual dimension of the modules was lost through this step. Therefore, I withdrew these changes and added the possibility to define a minimum distance between all modules.

The technology rules are passed through to Plantage by the ICFBInterface. At least for the technology definitions of Austria Microsystems it is not necessary to convert these rules, as the same format is chosen. For other manufacturers a conversion might be necessary.

Previously, the vias and routes were not considered at all. The main information which is needed to route are the dimension of the vias on the different layers and the width for the routes. Actually, there is no real width for a route, but a minimal width, which a route must have. Usually this width is used for the connections. Also, the dimensions for the via must be calculated (Equation 2.1), while the generated routes and placed vias needed to be added to the output of Plantage, so that they can be shown in the GUI.

Beforehand, every pin consisted only of one rectangle, which is obviously not enough. This basic information was used to minimize the estimated net length for the placements. For real routing more complex possibilities need to define pins, as for example, the gate can be connected from both sides. Moreover, it is not recommended to contact it over the diffusion

area. This will result in even more complex contact areas, as they could be split up in different parts. The definition of these areas is done by the user in the GUI and handed over to Plantage. The used format defines for every pin, which is part of a module, at least one contact area, but eventually a lot more. During the routing the actual contact area is then selected based on a certain heuristic.

4.1.2. ICFBInterface

The GUI-part of this application also needed some adaptions. This work was mainly done by Martin Keßler and contained the following parts:

- possibility to define contact areas
- displaying routes and vias
- possibility to change algorithm settings

4.2. Overview of Routing Algorithms

An interesting overview of the existing routing algorithms and the different approaches can be found in (Balasa et al. 2011, page 149 till 201). The more sophisticated routing algorithms described there are based on two basic algorithms:

- Maze Router
- Line Router

These algorithms are so-called general purpose router, they can be used for global and detailed routing. To find out which type of algorithm may be appropriate in my case the most important algorithms will be explained.

4.2.1. Basic Routing Algorithms

Maze Router

Another name for an algorithm, which is often used, is Lee's algorithm. But whatever the name is, it is based on a grid, where the obstacles and pins can be filled in. Basically, this is the entire information, which is necessary for the router. Afterwards, a wave of numbers is started from one pin and if this wave reaches the target a possible routing is found (Figure 4.2). The router guarantees the shortest possible route and will find a solution in all cases, if

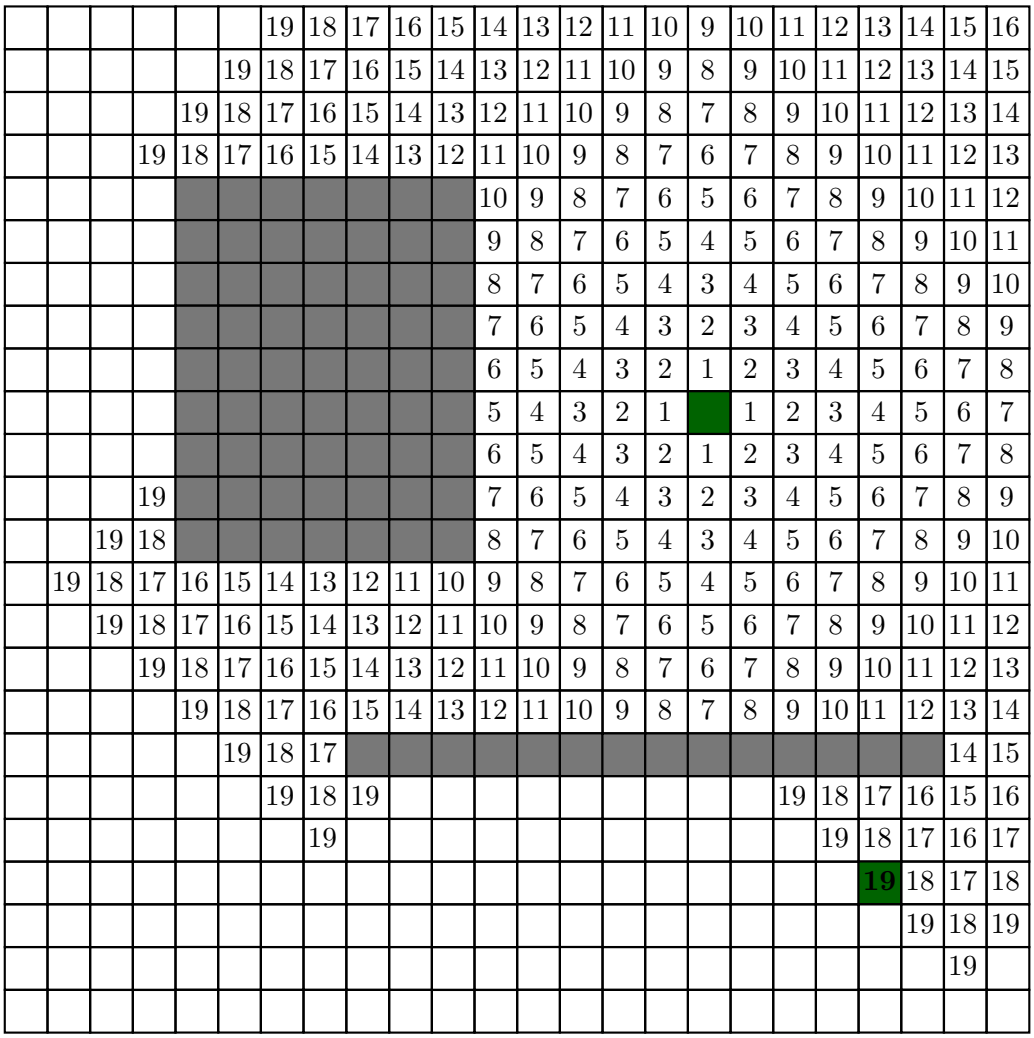


Figure 4.2.: a wavefront explores the area till it reaches the target

it is possible (Wang et al. 2009, p. 693). All in all, this is a very simple approach, but it has several drawbacks. First of all, it is quite slow and very memory-consuming (Wang et al. 2009, p. 693), as a big matrix might be necessary. This memory-consumption also heavily relies on the grid size, which raises the next question: How fine should the grid be? Those are problems which can be solved. So, for example, to attack the problem with the performance it is possible to transform the algorithm into an A*-Algorithm, which is actually just a modified maze router that prefers the shorter directions to the target. Also, for the other problems there are solutions which are mainly part of the more sophisticated routing algorithms.

Line Router

The main difference of a line router to a maze router is the usage of lines as part of a route instead of points. This improves the memory usage and usually also avoids unnecessary bends. At the same time this modification causes the loss of the guarantee to find the shortest path (Wang et al. 2009, p. 695). The other difference lies in the search algorithm itself, as the line router performs a depth-first search instead of a breadth-first search like the maze router. The basic idea behind this type of router is to try to reach the target, or at least one of its coordinates in the first step, through a line that is as long as possible. This line may hit an obstacle, which means that the direction must be changed. In this way the router always goes as far as possible into one direction and changes it until it reaches the target.

4.2.2. Steiner Routing

The algorithms above only considered two-pin nets, but very often nets have more than two pins. For these cases a possible solution would be to consider only two pins at a time and connect the others sequentially to the previous ones. This can result in a worse outcome than the optimum (Wang et al. 2009, p. 701). A better approach is to use a Steiner-tree (Balasa et al. 2011, page 153).

4.2.3. Sophisticated Routing Algorithms

As it is not the topic of this thesis to describe every possible routing algorithm in detail, as I only want to explain some possible approaches. They are built around several interesting ideas but all have at least one drawback, which made them not useful for the hierarchical placer, or they were too complex to be easily implemented. As the target of this thesis is mainly to create feasible and useful, but not perfect, layouts, I have not chosen one of these sophisticated algorithms.

- ROSA (Balasa et al. 2011, page 165)
- MIGHTY (Balasa et al. 2011, page 166)
- ILAC (Balasa et al. 2011, page 167)
- SALIM (Balasa et al. 2011, page 167)
- SLAM (Balasa et al. 2011, page 167)
- hierarchical global router (Balasa et al. 2011, page 168)
- plan-based layout algorithm (Balasa et al. 2011, page 168)

- ANAGRAM (Balasa et al. 2011, page 168)
- ANAGRAM II (Balasa et al. 2011, page 168)
- area router (Balasa et al. 2011, page 168)
- modified Lee's algorithm (Balasa et al. 2011, page 170)
- ROAD (Balasa et al. 2011, page 170)
- ANAGRAM III (Balasa et al. 2011, page 170)
- router with constraint generator (Balasa et al. 2011, page 172)

4.2.4. Routing algorithms integrated into the placer

It could be a good idea to try to integrate the routing into the placer. Possible approaches into this direction are KOAN (Balasa et al. 2011, page 178), RACHANA (Balasa et al. 2011, page 178), GELSA (Balasa et al. 2011, page 179), and several others. The major problem with these solutions is the big modification of the placer. Actually, it is useless to try to integrate the routing algorithm this way into a placer after it was developed, as it must be created for this case especially with the routing in mind, from the beginning on. As the placer in Plantage was not supposed to include the routing algorithm, I had to use a different approach.

4.2.5. Template Based Algorithms

Template based algorithms do not start from scratch, they already have a possible layout for a slightly different circuit or technology. Such a solution can be used in several cases and it gives the layout designer nearly full control over the process. In my case I have to assume that there is no such template available and the intention of the whole set of tools would be that the user has to provide as little information as possible. This would really mean *electronic design* ***automation*** and therefore should be the real target.

4.3. Implemented Routing Algorithms

The first decision I had to make was to decide which algorithms should be implemented. The more sophisticated solutions and all global and detailed approaches seemed a bit to complex to implement them properly in only a few weeks. So the real choice was to use something based on a maze or a line router. As the line routers seemed to be a good trade-off between complexity and performance I have chosen something based on them. In the end I have implemented three

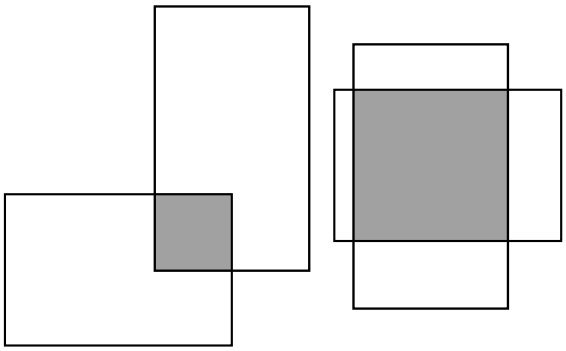


Figure 4.3.: overlapping area of two rectangles is again a rectangle

different routing algorithms: first of all, a typical line expansion router. This solution does not consider all constraints but it is able to create at least reasonable results. After that, I extended this algorithm in order to consider additional constraints. To improve the performance I also implemented a very special version of the line expansion router, which really only creates feasible but not very useful routings. This router can be used for the pre-routing, where the results are not stored anyway.

4.3.1. Line Expansion Router

Select a Target

The first thing which the algorithm has to do before it can calculate a route is to select the start and target contact area. There can be several possibilities, as every pin can have several contact areas. Later on it will be noticeable that already created routes can be candidates to start or end with a route. Basically, this task can be seen as the selection of the closest rectangles of two lists of possibilities. For this I want to talk about how the closest distance between two rectangles can be calculated.

For two rectangles there are several possible constellations. For the first possible position one has to check whether they overlap. If this is the case I can easily calculate the rectangle which is covered by both (Figure 4.3) and the distance is 0. As starting points for both rectangles I can select the center of gravity of the overlapping area.

If the two rectangles do not overlap it becomes more difficult. In this case I can at least say that one of the rectangles has its closest point at one of the edges. Actually, there can even be an endless count of solutions for this problem, but I am only interested in one of them. Because of this I select the solution that is the easiest to calculate, which means the selection of a corner for one rectangle. The closest corner from the first rectangle to the second one is one that has the closest euclidean distance to the center of gravity of the second one (Figure 4.4). As the closest point of the first rectangle is now known, I can start to select the closest

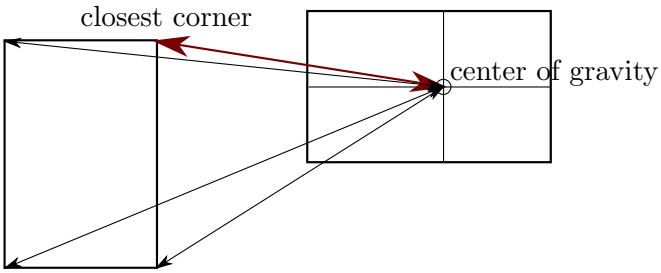


Figure 4.4.: the closest corner is that one, which has the smallest distance to the center of gravity of the other one

point on the second one. This point must be obviously on one of the edges again. Every edge is basically a line segment, where the closest point can be calculated easily. For this task I, firstly, need to expand the line segment into a whole line. Secondly, I can then turn this line 90 degrees (it does not matter in which direction as it is a line) and move this new line, so that it crosses the point, from which I want to know the closest point on the line segment. The crossing point of these two lines, if it is also inside the line segment, is the closest point. If this crossing point is outside the line segment, the closest point must be the start or end point of the line segment, which are actually the rectangle's corners.

The task of the selection is at this point not finished, because in special cases it is possible that I can now again find a closer point on the first rectangle to the second one. So I have to do the last step once again, for one last time, but with the point of the second rectangle as source.

A very similar procedure to this one is done for every step - the selection of a new target. This may be useful, as a different target area can be now closer than previously, mostly if the last step went into a bad direction. The task in this case is to find the closest rectangle to the current point and, afterwards, the closest point on this rectangle.

Consider Obstacles

After the algorithm has selected start and target contact areas the next step is to avoid routes across modules. For this purpose during all the following steps the modules are considered as obstacles. But during the first step usually the route has to get out of the module first. To avoid additional, accidentally added gates the algorithm has to choose the shortest way to get out of the module. To avoid problems with the design rules it is not possible to start right at the contact area to the next layer and use, for example, metal to get out of the module as fast as possible.

As the algorithm now has all the necessary information, the actual line expansion routing can

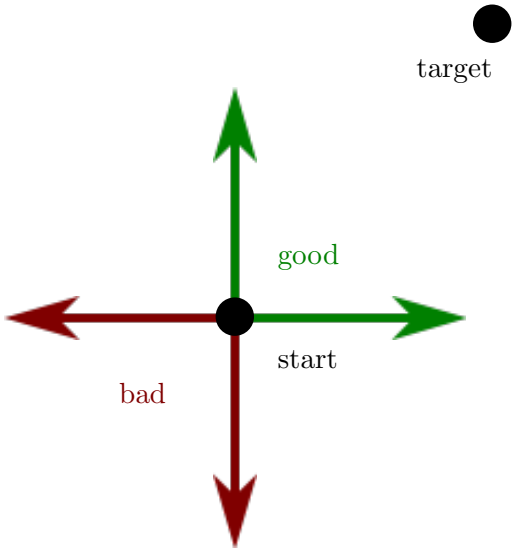


Figure 4.5.: good and bad directions

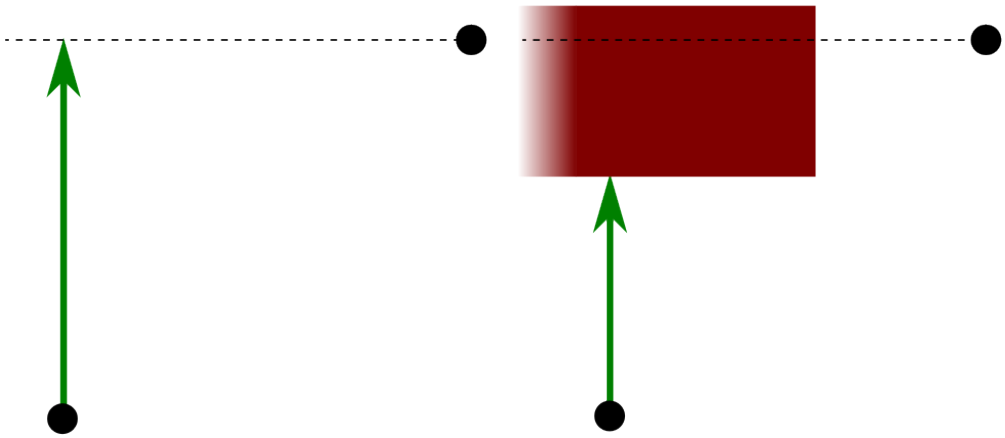
start. The main idea behind a line expansion router is that the routes are created sequentially. Every single step is based on a map of obstacles, which the route must not cross. Beginning from the starting point I always calculate the next possible point of a route and afterwards apply the same algorithm again to this partial problem. At the beginning of this recursive algorithm, as usual, I check the terminating condition whether I have reached the target point.

In every single step I have to make several decisions: First, I group the possible directions in good and bad directions (Figure 4.5). Good directions are those which reduce the distance to the target, whereas bad ones are all the others.

Additionally, it is pointless to go back the same direction from where the last step came from. Also, going the same direction as with last step is not reasonable. The reason therefore is that if this would be a good decision, the last step could have gone further. So I remove these two directions from the good directions. After this selection I can make one step further in every good direction, as far as possible. This means that the next step can go as far as no obstacle is in the way or the target (in this coordinate) is reached (Figure 4.6).

During the calculation of the steps towards the target I count those, which are feasible. If none of them is feasible I will have to select one of the bad directions for the next step. A major premise for this is also that the last step ended at an obstacle. The router now can go as far as necessary into this bad direction (Figure 4.7).

Sometimes it is also necessary to go above an obstacle, which is on a layer below or above it. This step is also one into a bad direction, because if it would be a good direction it would be done automatically. As it is a bad direction it should only go as far as necessary (Figure 4.8). The same direction of the obstacle may not be possible as on the current layer only vertical or



(a) no obstacle (b) with obstacle

Figure 4.6.: in a good direction the route can go as far as possible

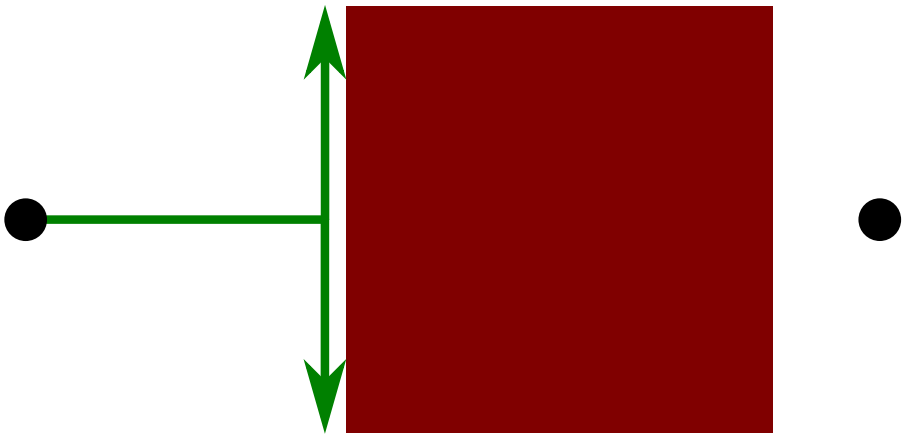


Figure 4.7.: a step into a bad direction should go only as far as necessary

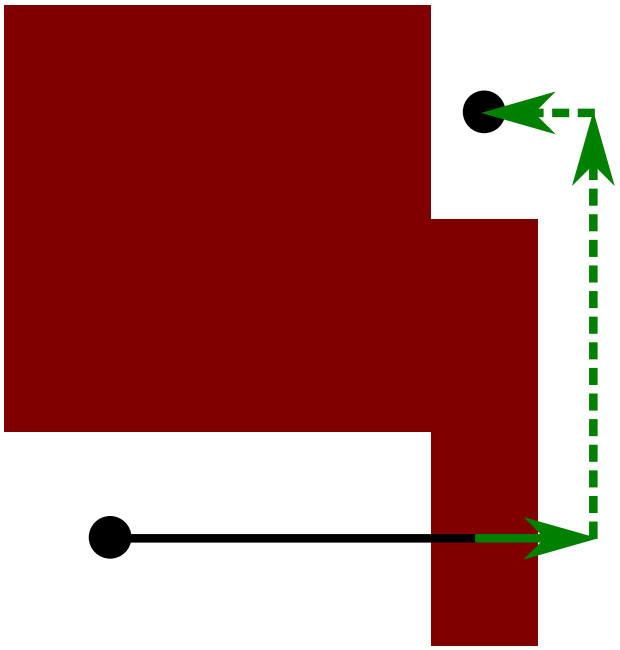


Figure 4.8.: a step over or below an obstacle

horizontal routes are allowed.

Last but no least, it is also possible to switch to the neighboured layers. For modules the obstacle cannot be crossed this way, as they are obstacles at every layer. But if for example the obstacle is just another route a layer switch may solve the problem. In such cases it is even reasonable to force a layer switch when it is currently not possible, because for instance another route is too close. In this case I try to make one small step, which is just big enough to fulfill all design rules (especially the notch rule) and switch the layer at this new point.

All these possibilities to find a way to the target must be combined in a clever way otherwise the performance would become unsufficient. Therefore it makes sense to, for instance, prefer the steps into good directions. I choose the bad directions only if the shortest possible connection with only steps into good directions at the current point was bigger than 10% of the Manhattan distance to the target. This ensures that routes that are too long can be avoided and still, results are created fast enough.

Avoid Collisions

To avoid collisions I take the preferred directions for every layer into account. This information is again stored in the technology file and describes if a layer should be used for horizontal or vertical routes. This reduces the possibility of collisions very efficiently and results in less complex layouts. Last but not least, it also improves the performance of the router significantly.

Symmetric Routes

In manually created layouts an usual way to route symmetric elements is to create them optically symmetric. This optical symmetry gives the designer an easy way to create symmetry. But the really important part about routing symmetries is to match the parasitics. The resistance of the route is certainly matched for the symmetric routes through an optical symmetry, but the capacitive couplings are obeyed in most cases. Therefore, I concentrated on the resistance. To also consider the capacitance it would have been necessary to create all routes at the same time. Only with such an approach I can evaluate the capacitive parasitics, otherwise I would have never known what influence routes that were created afterwards would have. As I have chosen to create the routes one by one I cannot consider the capacitance for the symmetries.

But before I go more into detail of the routing itself I want to further explain the situation. Symmetries are caused by symmetry constraints, which can have, as already described earlier (Section 2.4.1), single and pair modules. Single modules are quite useless for a symmetric routing. The only modules of interest are pair modules, no matter if they only must be connected together or also to another module.

In the routing I first extract all modules, which are symmetric and where both parts of the symmetry pair must be connected to the same net. Afterwards, I connect these pairs together and the only possible starting or ending point for other routes is then the middle of the route between the pair (Figure 4.9). In this case the middle does not have to be the same distance to both sides but the same resistance to both modules. Therefore, if the route uses different layers from the modules to the middle, it is possible that the middle in terms of resistance can be different to the middle in terms of distance.

Guard Rings

With guards (Figure 4.10) a designer can protect, for example, a certain signal from another very noisy one or protect whole module groups. The latter is often used for current mirrors, as they are very sensitive to noise. In physical representation a guard or guard ring is only a layer, for instance MET1, contacted to the substrate and connected to a certain Voltage level (often ground). For the routing this only results in an additional obstacle on this layer.

The major difference of guard rings to usual obstacles is that they are obstacles only for certain nets. If you look at it the other way round, a guard ring is not an obstacle for the nets inside. This is necessary so that other routes do not use the space inside a guard ring, but the pins inside can still be connected. For example, a MET1-guard will result in a normal obstacle for all nets on MET1. On all other routing layers there also will be obstacles, but only for nets which do not have pins inside the guard ring.

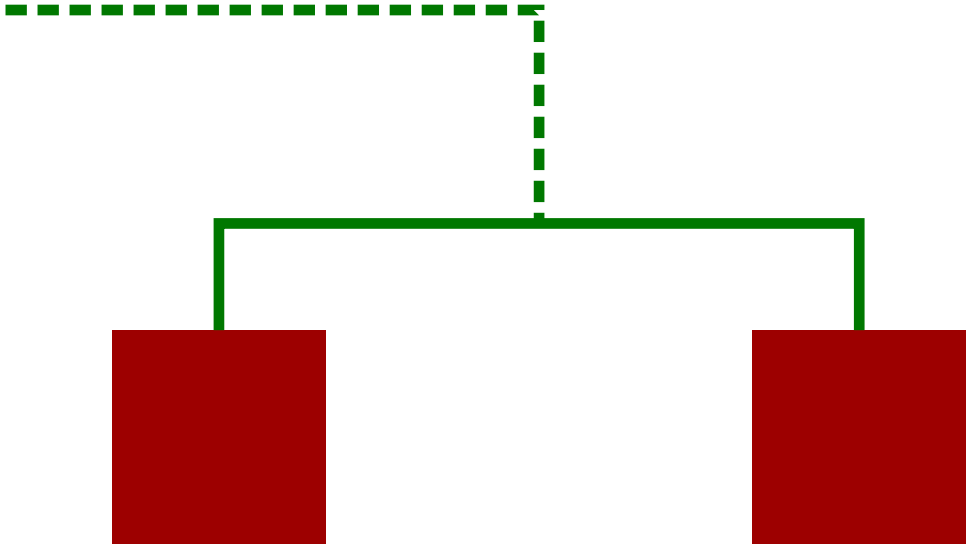


Figure 4.9.: a symmetric pair should be only connected in the middle in terms of resistance

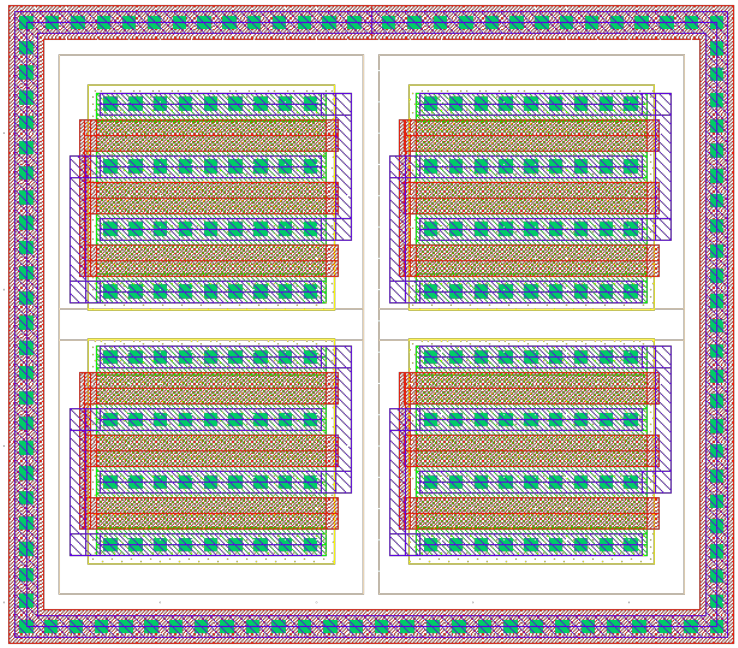


Figure 4.10.: the common centroid of a differential pair, protected from noise by a guard ring

Multi-Pin Nets

To solve the problem of multi-pin nets it would be possible to use a Steiner-tree (Section 4.2.2), but this approach would not be the easiest one to implement. That is why I have chosen a different solution. Basically, I connect the first two pins at the beginning, where the order of the pins is random (actually they are sorted alphabetically by their name, but this was not the intention and therefore does not matter). Afterwards, I transform the whole new connection into additional possible targets. From this now bigger list of possible targets I select the closest one and connect the next pin to it. The result of this step is again transformed into additional targets for the next pin, and so on.

Select a Route

Finally, if the starting point is the same as the ending point I have to calculate a feasible and complete route. I add the result to all the possible routes for this necessary connection. From these routes the first, quite simple algorithm chooses only the best one. As this algorithm calculates the routes sequentially it has no information on, for example, capacitive couplings. Therefore, the *best* possibility is chosen on the lowest resistance.

For this first algorithm a lot of different variants for every necessary connection is a lot of information, as it chooses only the shortest one in terms of resistance. But the results, which I generate here, can be reused for the extended version of the line expansion algorithm.

4.3.2. Line Expansion Feasibility Router

In this special case I had to implement the routing into the hierarchical placing algorithm. This creates some additional requirements for the routing algorithms. The most important difference to an usual placer is that the actual positions of a module can change. Because of this, the positions are not stored, only the selected variants and the enumeration sequence is used to specify a certain placement. As the positions of the routes would be relative to the module positions it is not useful to store the routing results for the next higher hierarchy level. Another reason for this is the possibility of other modules, which could overlap with routes after the calculation of the placement on the next higher level. It is still important to try to route the placement, as a placement may cause very long or even infeasible routes. Because of these special requirements it is possible to select a different router for the pre-routing step than for the final routing. Actually, it would be possible to select every router here, but it is only reasonable to select one which requires less computing time.

This is the point where the line expansion feasibility router comes into play. This algorithm is a modified version of the basic line expansion algorithm. The only difference is the amount of possible routes, which is created for every necessary connection. The basic version creates as many solutions as possible, whereas the modified version only creates one route. This will

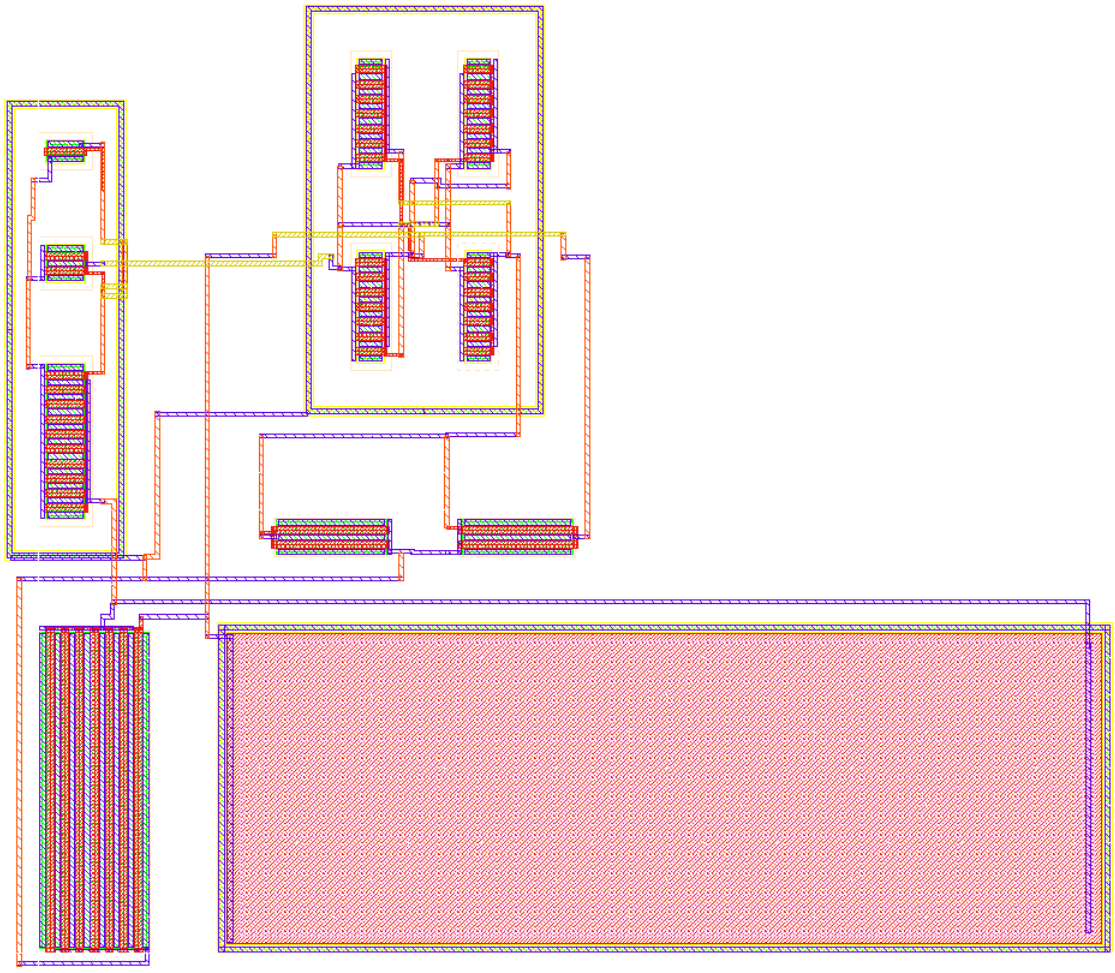


Figure 4.11.: layout of a miller amplifier routed with the basic line expansion route

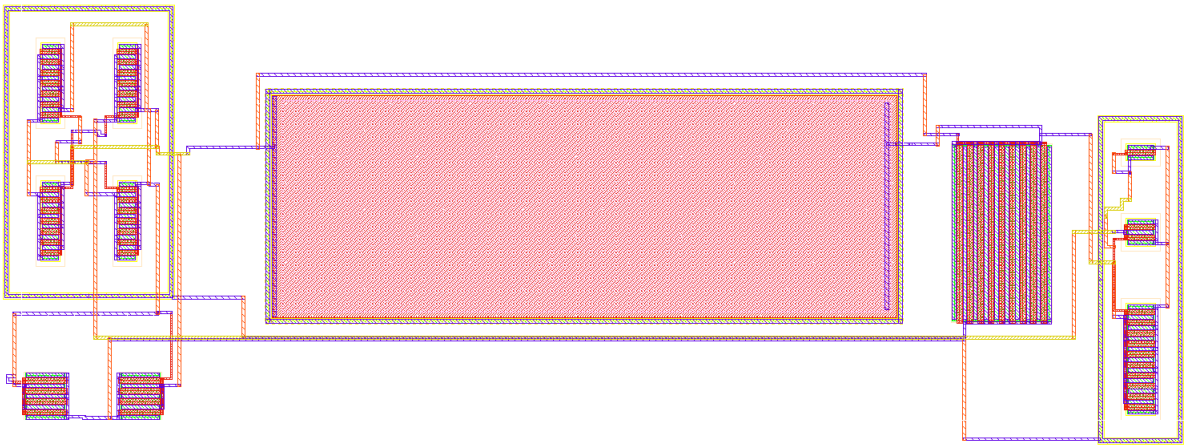


Figure 4.12.: layout of a miller amplifier routed with the basic line expansion route

result in usually quite bad layouts and because of this the router should not be used for the final step. But as pre-router the algorithm can be useful, as it is very fast and can select very efficiently if a placement is at least feasible to route for a line expansion router.

4.3.3. Extended Line Expansion Router

This extended version of the line expansion router is based on the simple version. So it is again a sequential approach with the same drawbacks and advantages as the previous versions. The main difference to it is how I select the routes. I actually create the routes in the same way as for the previous version. To improve the performance I make a pre-selection based on the resistance, which can be done easily, as the routes are already sorted by their resistance. For all the routes, which are left then, I calculate the total edge- and area-capacitance that would turn out if this route is added to the connections of the previously routed nets. I then combine these values together to the parasitic sum

$$p = w_R \cdot R + w_{C_a} \cdot C_a + w_{C_e} \cdot C_e \tag{4.1}$$

(where w_i are the weights and R, C_a and C_e are the parasitics as defined in (Section 2.6)) on which the actual selection is based.

5. Results

As only the basic line expansion router and the extended version produce useful results I will only explain results for these two. For the first version (Figure 5.2) it is quite easy to determine whether the result is reasonable, as it always chooses the shortest connection (or simply: the one with the smallest resistance). The two presented layouts have a relatively big minimal distance between all modules set, which causes a layout that is actually too big. This was necessary, as it is not yet supported by the GUI of the ICFBInterface to specify minimum distances between only some modules. Therefore, I had to choose a big minimum distance between all modules, although I only needed a bigger distance inside the common centroid.

For the second one, the extend line expansion router, it becomes a lot more difficult to determine manually if there are errors in the layout (Figure 5.2). This router decides which connection should be used also based on the capacitive couplings, therefore it must not always be the shortest one.

To proof whether the Plantage produces better results for circuits without common centroids, I have created two layouts, one with the basic line expansion router and one with the extended version. As it can be seen a far smaller distance between the modules is necessary. (Figure 5.4).

To also show the results of a layout with a symmetry I have prepared one simple example which contains only three modules (Figure 5.3). Two of them form a pair of a symmetry and the other one is a single module in this symmetry. The layout shows that the two connections between the symmetric modules are connected nearly symmetrically to the third module. The maximum allowed difference in resistance is 5%, which seems to be held by both connections. One of them is optical quite unsymmetrical connected, but it starts on both ends on POLY. Therefore, the resistance of the whole connection is dominated by those two small parts.

5.1. Conclusion

The results above show that the layouts created by Plantage are at least feasible. Additionally, they also consider several constraints for analog circuits. The results are certainly not perfect yet, but this wasn't the aim of this thesis. The real main point was to implement the routing and make it easy to replace the algorithm later, as it is not certain which solution will be the best for the hierarchical placer. This can now easily be done; it is only necessary to inherit from the class Router.

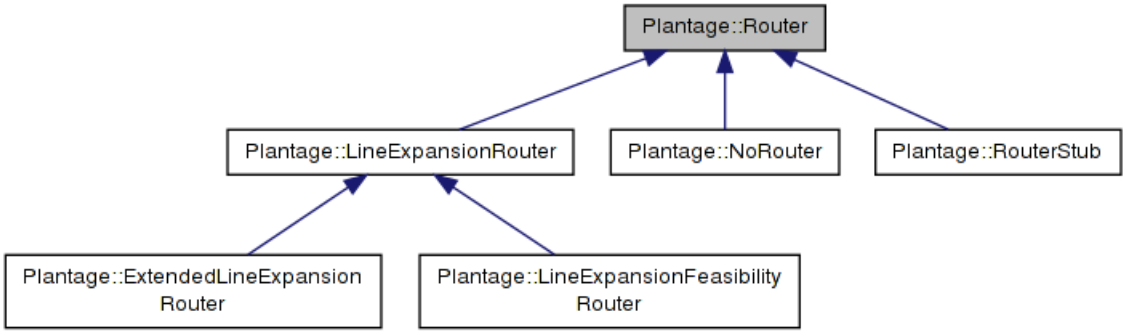


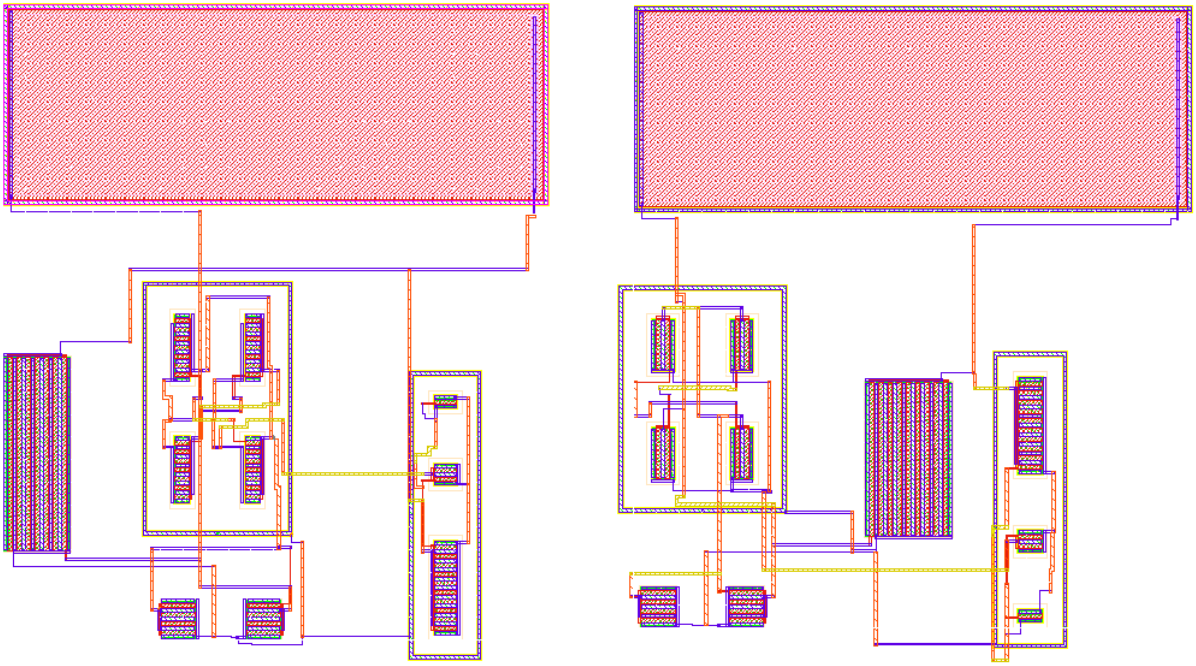
Figure 5.1.: inheritance diagram for the router

Finally, I was able to reach all the targets claimed in the introduction:

- a line expansion router, which creates, in most cases [1], design rule clean routes [2] and considers symmetries
- based on the previous router an extended version, which tries to reduce the parasitics
- a software design, where the routing algorithm can be exchanged easily

[1] In the cases where the layout is not design rule clean it is usually easy to fix the problem manually. For instance, in special cases the minimum notch rule may be harmed, but then the gap can just be filled up.

[2] For the capacitor the ICFBInterface does not provide enough information on the additional guard ring, therefore it cannot be considered



(a) basic line expansion router (b) extended line expansion router

Figure 5.2.: two similar placements routed

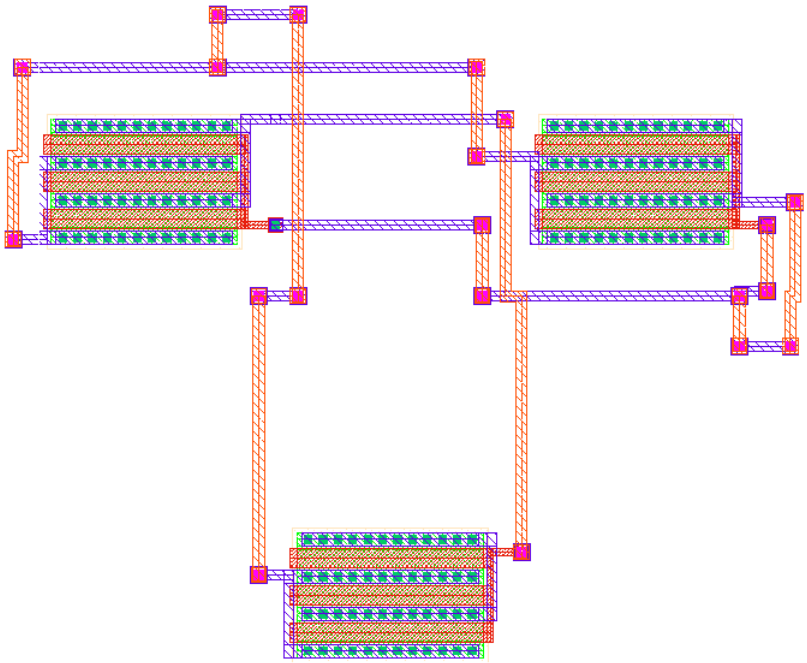


Figure 5.3.: routed layout of a symmetry

(a) basic line expansion router

(b) extended line expansion router

Figure 5.4.: two similar placements without a common centroid routed

List of Figures

List of Equations

Bibliography

Balasa, Lin, Chang, Strasser, Eick, Gräb, Schlichtmann, Dündar, Unutulmaz, Said, Dessouky, El-Adawi, Abbas, Shahein, Castro-Lopez, Roca, Fernandez, Jerke, Lienig & Freuer (2011): Analog Layout Synthesis.
`Springer,ISBN:978-1-4419-6931-6`

Eikland & Notebaert (08.09.2013): Introduction to lp_solve.
`http://lpsolve.sourceforge.net`

Strasser, Eick, Gräb, Schlichtmann & Johannes, M. (2008): Deterministic analog circuit placement using hierarchically bounded enumeration and enhanced shape functions.
`ICCAD,DOI:10.1145/1509456.1509530`

Wang, Chang & Cheng (2009): Electronic Design Automation.
`Kaufmann,ISBN:978-0-12-374364-0`