A New Interactive Analog Layout Methodology based on Rubber-band Routing

Kazuhiko Kobayashi
Wayne Wei-Ming Dai

UCSC-CRL-96-12
13 June 1996

Baskin Center for
Computer Engineering & Information Sciences
University of California, Santa Cruz
Santa Cruz, CA 95064 USA

ABSTRACT

In this report I formulate analog layout constraints and survey the state of the art of automatic analog layout systems, which can handle only few analog constraints, and generate less dense layout. To solve these problems I propose a new interactive analog layout methodology. It provides topological editing in the geometrical view based on Rubber-band routing. The purposes of this new methodology are i)to overcome the difficulty of control the layout parasitic elements with irregularities of analog devices and wiring effects and ii)to reduce the analog VLSI design period. After describing new concepts for that interactive methodology, I state some specific challenges.

Contents

List of Figures

Acknowledgements

I want to thank Kevin Karplus and Joel Ferguson for helping me improve this report.

This report is an extension of the SURF research project. I want to thank all of the people that have worked on SURF, especially David Joseph Staepelaere for introducing the layout system SURF and discussing our strategy; Man-fai Yu and Jeffrey Su for useful ideas on obstacle representation issues.

1. Introduction

1.1 Analog VLSI design flow

The design flow of analog and analog/digital mixed-signal VLSI design is depicted in Figure 1.1. Analog layout constraints are produced as either the output of circuit analysis or are based on designer's experience. These analog constraints are necessary for the successful design of such analog circuits as: A/D converters, D/A converters, filters, operational amplifiers, and so on. The layout step must obey these constraints in order to successfully realize these analog circuits. Once the layout is completed, parasitic circuit extraction and simulation steps are performed. If the circuit has the proper behavior, the mask pattern is then generated. Otherwise, the constraints are modified or the layout is adjusted and then the parasitic circuit extraction and simulation are repeated. This cycle continues until the circuit achieves the desired analog behavior. Therefore the most important goal of the layout step is to ensure the functionality and performance of the circuits—not to achieve minimum chip size. Small chip size is a secondary goal.

The problem is that automatic layout systems have difficulty converging to a solution due to the increasingly complex requirements of analog VLSIs. Even when using manual layout techniques, designers must satisfy numerous critical constraints and unskilled layout designers can take a long time to finish the loop.

The SURF layout system has been developed for digital circuit design and it has been applied to several real designs. My goal is to propose theoretical extensions to SURF that provide an interactive improvement method for analog layout which can help reduce the layout design period and the number of design iterations. For that goal I investigated analog layout constraints and surveyed the previous work on analog layout. Through the experience of checking the implemented editor functions, I considered the need for a new interactive method.

1.2 Organization

The rest of this report is organized as follows. The definition and electrical effects of analog layout constraints are introduced in Chapter 2. Chapter 3 shows the previous work in the field of automatic analog layout. Chapter 4 describes the characteristics of SURF for digital circuits. A data representation for a new interactive methodology is explained in Chapter 5 and detailed techniques are described in Chapter 6.

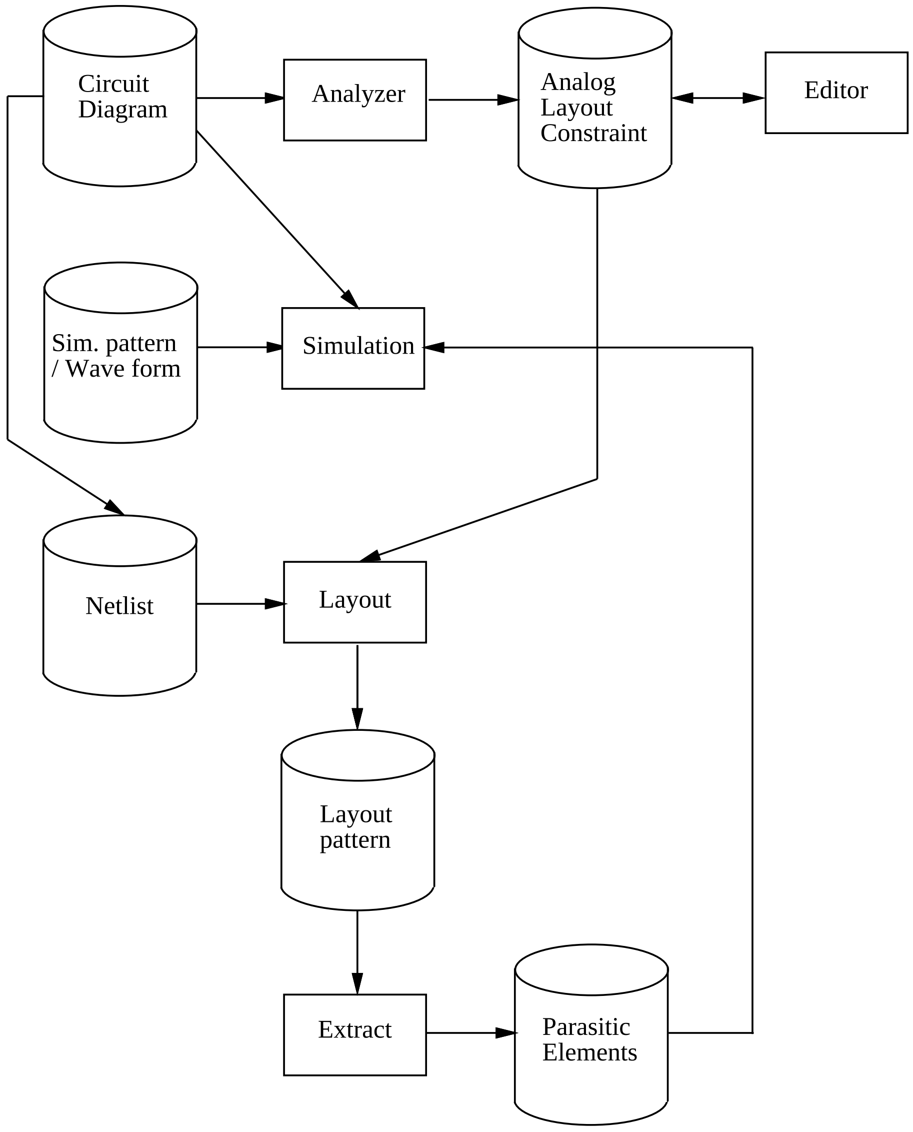


Figure 1.1: Analog and mixed-signal VLSI design Flow

2. Analog layout constraints

2.1 Definition

I define analog layout constraints as those constraints which must be met during the layout step in order to implement analog circuits with the desired behavior. Without proper attention to analog layout constraints, performance degradation due to device mismatching, parasitics, and noise coupling can occur. For example, in a layout containing both digital and analog functions, signal noise fed from the digital circuit to the analog circuit can degrade the analog behavior.

To create a successful analog layout system, a set of numerical constraints, which achieve the desired analog behavior by placing goals or limits on physical layout properties, must be constructed. The values of these constraints depend on process technology and are usually determined experimentally. Then a layout system can implement the analog circuits by satisfying these values.

Some constraints can be generated automatically by applying pattern recognition techniques to the circuit diagram. For example, the source-coupled pair and the current-mirror in Figure 2.1 can be extracted. The designer would also add other typical constraints to reflect the target specifications of each circuit.

2.2 Three types of analog layout constraints

I can categorize analog layout constraints into three groups: device matching, crosstalk, and symmetry. I introduce these constraints, including how the designer specifies each constraint, in this section. I explain the effects of each constraint in the next section.

2.2.1 Device matching

Device matching constraints restrict the shape of a device, the orientation, and placement location. Each device in a constraint group must have the same shape and orientation as all the the other devices in the group. By the ways the designer specifies constraints, they are divided into *device-proximity* and *device-separation* constraints. Devices with a device-proximity constraint should be placed close to each other (Figure 2.2(a)) and devices with a device-separation constraint should be placed in a limited region. For example, in Figure 2.2(b) the centers of the two transistors must be placed in a distance of at most $Xmax$. For device-proximity constraint the designer specifies the constraint type and the device instance names. For device-separation the allowed maximum distance of the device location region is specified additionally.

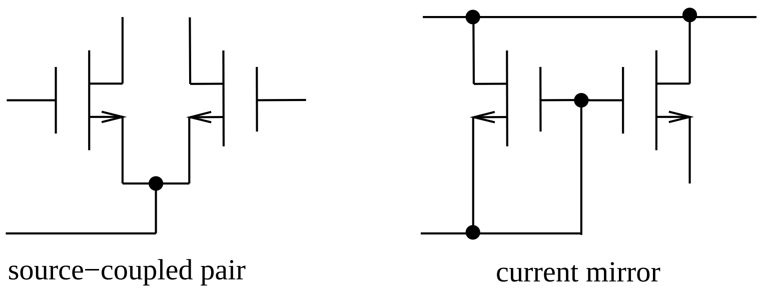


Figure 2.1: Examples of automatically extracted constraints

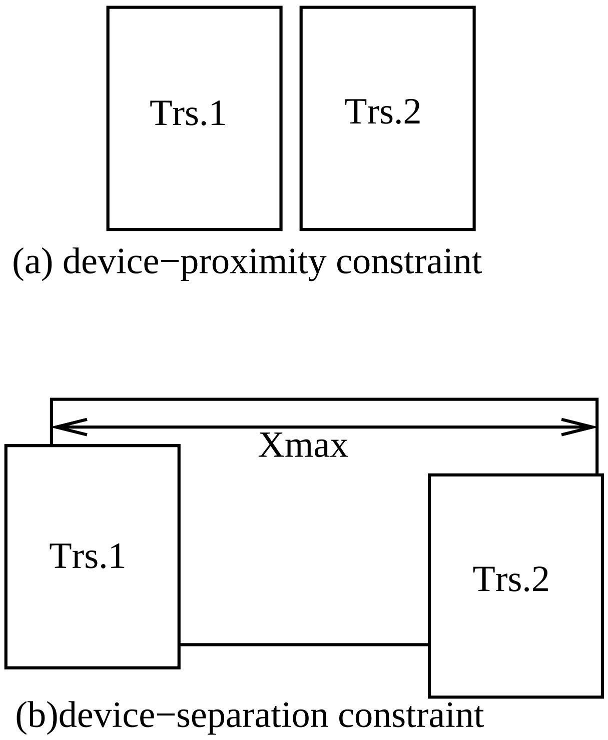

Trs.1
Trs.2
(a) device−proximity constraint
Xmax
Trs.1
Trs.2
(b)device−separation constraint

Figure 2.2: Device matching constraint

2.2.2 Crosstalk

Crosstalk constraints are used to avoid interference among neighboring nets. Some pairs of nets are restricted in the length that they may run parallel to one another. Intersection of the nets on different layers may also be prohibited. The designer specifies the constraint type, a set of net names, and some combinations of the separated distance and the muximum length which both wires can go in parallel for each layer.

2.2.3 Symmetry

A symmetry constraint requires that a component or set of components share a geometrically symmetric pattern. There are *mirror-symmetry*, *self-symmetry*, and *wiring-cross-symmetry* constraints. In each case the elements must be laid out symmetrically against a center line. In the mirror-symmetry constraint an even number devices are located symmetrically. A pair of devices in this constraint must have the same shape when one of them is reflected about the center line. On the other hand in the self-symmetry constraint one device makes a symmetrical pattern with itself. In the wiring-cross-symmetry constraint the wire pattern constitutes the symmetrical pattern with making different layers overlap on the center line (Figure 2.3). The designer indicates the constraint type and the device instance names for the mirror-symmetry and self-symmetry constraints and specifies the type and the net name for the wiring-cross-symmetry constraint.

2.3 Effects of analog layout constraints

2.3.1 Device matching effect

In this subsection I describe the electrical effect of device mismatching. I consider the situation shown in Figure 2.4. If two identical transistors, which must be placed by no more than *Xmax*, are

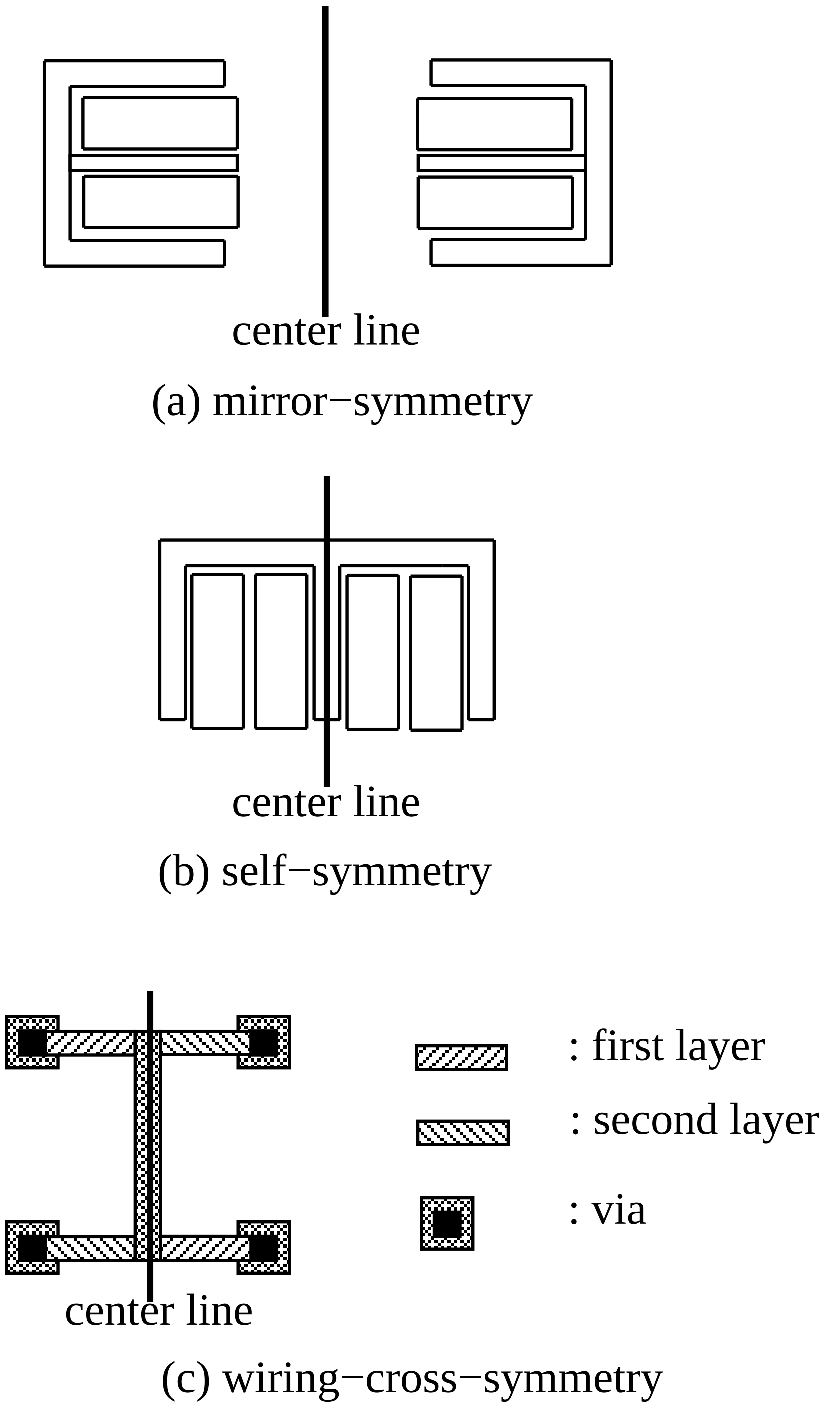


Figure 2.3: Symmetry constraint

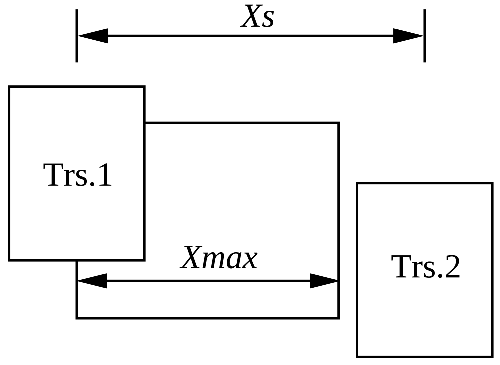


Figure 2.4: One example of device-separation constraint

placed so that their distance is Xs, the cost of device mismatching is:

$$Costmis = \begin{cases} 0 & \text{if } Xs \leq Xmax \\ (Xs - Xmax)^2 & \text{otherwise} \end{cases} \tag{2.1}$$

Because for two transistors with identical length L and width W, the variance of the mismatch of one electrical parameter P such as threshold voltage Vth between them is represented as:

$$\sigma^2(\Delta P) \simeq \frac{A_p^2}{2 \cdot W \cdot L} + S_p^2 \cdot D_x^2 \tag{2.2}$$

where
A_p : the area proportionality constant for parameter P.
S_p : the spacing proportionality constant depending on the process technology.
D_x : the distance between the centers of two transistors.
In this formulation the first term models non-spatially correlated effects due to local process variations, e.g. mobility variation and oxide charge variation. The second term represents spatial effects due to global process gradients, e.g. oxide thickness gradients and dopant diffusion gradients [CGRC94].

The circuit designer determines the value at which the mismatch variance given by equation 2.2 is no longer negligible and uses it along with other cost considerations to determine the maximum limit $Xmax$.

2.3.2 Crosstalk effect

Another important electrical behavior concern is crosstalk. Crosstalk appears as an undesired transient change on a victim signal by a close affecting signal such as a noisy net. In Figure 2.5 signal A is an affecting signal and signal B is a victim signal. If they are laid out in parallel within some distance and for some length, then crosstalk occurs when signal A switches.

The following symbols are used to explain crosstalk behavior.
C_m, L_m : the mutual capacitance and inductance between the two signals.
V_c, I_c : the capacitively coupled voltage and current.
V_l, I_l : the inductively coupled voltage and current.
V_i : the voltage of the affecting signal.
T_r : the rise time of the affecting signal.
I_{NE}, I_{FE} : the crosstalk effect current at the near and far end.
(The near and far end are represented as NE and FE in Figure 2.5, respectively.)
V_{NS}, V_{NL} : the crosstalk effect voltage with short lines and long lines at the near end.
Using the above notation, the crosstalk effect current at the near end I_{NE} and at the far end I_{FE} are :

$$I_{NE} = I_c + I_l. \tag{2.3}$$

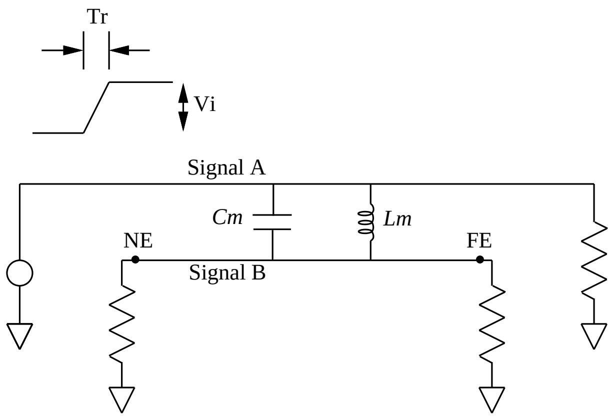


Figure 2.5: Crosstalk effect

$$I_{FE} = I_c - I_l. \tag{2.4}$$

If I assume these signal are isolated from other objects, these wire patterns are considered as homogeneous medium. Therefore $I_c = I_l$ and $I_{FE} = 0$.

Now I consider the voltage effect of crosstalk with short lines and long lines, V_{NS} and V_{NL}. In the case of short parallel lines : $2{\cdot}\mathrm{T}_d < T_r$, where T_d is signal propagation delay on the parallel lines, the resultant voltage effect V_{NS} on a victim line is represented as:

$$V_{NS} = K_b \cdot V_i \cdot (2 \cdot T_d/T_r) \tag{2.5}$$

where $K_b = (C_m/C + L_m/L)/4$.
C, L : the unit capacitance and inductance of the line.
In the case of long parallel lines : $2{\cdot}\mathrm{T}_d > T_r$, voltage saturation occurs and the resultant effect V_{NL} is represented as:

$$V_{NL} = K_b \cdot V_i. \tag{2.6}$$

The circuit designer must determine the length that the two signals may run in parallel when separated by a specific distance.

2.3.3 Symmetry effect

Symmetry constraints can help reduce a circuit's sensitivity to a radiating thermal radiating device. If two strongly thermally sensitive devices are placed randomly, a temperature difference between those devices can cause incorrect analog behavior. For example, in a power circuit the output power transistors are the key to the circuit's performance. Therefore the layout of the power transistors requires additional care. A power device placed close to sensitive devices serves as a concentrated heat source. So these sensitive devices must be placed symmetrically with respect to the thermal radiating device (Figure 2.6).

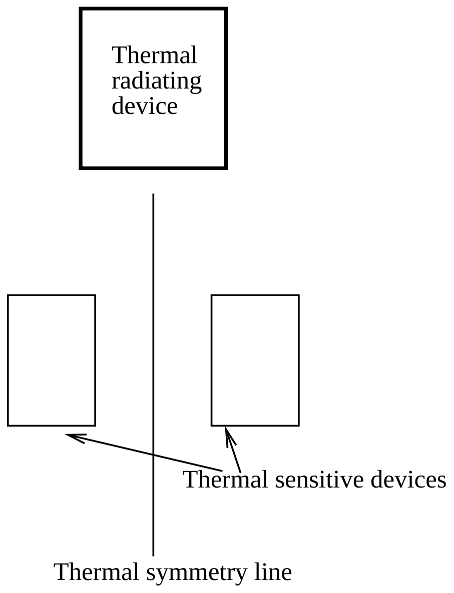


Figure 2.6: Thermal effect and symmetry pattern

3. Previous work

3.1 Channel Routing for an analog module

The channel router for an analog module ART [CSV93] uses a constraint-based algorithm. Analog layout constraints are generated by a parasitic constraint generator called PARCAR [CSV90] using sensitivity analysis from the performance information. ART handles matching constraints and bounding constraints based on wire capacitance. Matching constraints require elements to have the exact same parasitic values as the input constraint values. This includes symmetrical patterns. Bounding constraints limit elements to have values smaller or equal to the input constraint values. A Vertical-constraint graph is used to represent these two types of constraints. Each horizontal segment of a net is represented as a node in the graph. Then ART assigns each segment to a track to reduce the channel height while satisfying the constraints. After routing, if the layout pattern doesn't satisfy the constraints, the placement must be changed.

In [CSV93] the authors said they would implement a placement program in the future. Obviously it is necessary to restrict the placement of some cells in order to create a symmetric channel wire pattern. It is difficult to create a placement program that both produces a dense channel pattern and satisfies constraints simultaneously. The channel routing style is not suitable for analog circuits, because it requires a rectangler routing region—analog components come in many shapes and requiring a rectangular region is too restrictive.

3.2 Automatic Layout Tool

KOAN/ANAGRAMII [CGRC91] is an automatic layout tool for device-level analog placement and routing. The placement program KOAN handles device matching and complex symmetric layout, including mirror-symmetry and self-symmetry, as analog layout constraints. KOAN can also handle dynamic merging and abutment of individual devices and generate well and bulk contacts. KOAN has a procedural device generator. It produces several shapes for one function by considering restricted shapes for the symmetry constraints. The placer uses simulated annealing to choose a suitable one among these shapes according to the location and the circumstance. The placer searches for a good placement by considering relocating, reshaping, and group moving operations with the following cost function *CostPlace*.

$$\begin{aligned} CostPlace = {} & w1 \cdot Overlap \ + \ w2 \cdot Area \ + \ w3 \cdot AspectRatio \\ & + \ w4 \cdot NetLength \ + \ w5 \cdot Proximity \ + \ w6 \cdot Merge \end{aligned} \tag{3.1}$$

where
$Overlap$: the penalty to avoid illegal overlap.
$Area$: placement size.
$AspectRatio$: the penalty for deviating from the desired aspect ratio $R_{desired}$.
$$AspectRatio = (R_{current} - R_{desired})^2$$
$NetLength$: the sum of estimated length of each net.
$Proximity$: the sum of the dummy net length among proximity-desired groups.
$Merge$: a term to encourage the device abutment calculated as
$$\begin{aligned} Merge = {} & \mathrm{C1}\cdot(\mathrm{MergeableArea} - \mathrm{MergedArea}) \\ & + C2 \cdot (\mathrm{MergeablePerimeter} - \mathrm{MergedPerimeter}) \end{aligned}$$

The router ANAGRAMII handles crosstalk avoidance, mirror-symmetric, and self-symmetric wiring as analog layout constraints. It adapts a line expansion search method with a tile representation and ripup-reroute. The cost of a partial path in the line expansion search *CostPath* is represented as:

$$CostPath = \sum_{segment\ s \in path} Wire(s) + Direction(s) + Crosstalk(s) + Ripup(s) + Distance(s\ to\ target) \tag{3.2}$$

where
$Wire(s)$: the area cost of the probed segment.
$Direction(s)$: the cost to enforce preferred directions for each layer.
$Crosstalk(s)$: the penalty to avoid crosstalk with some specified nets.
$Ripup(s)$: the cost to delete nets that are obstacles to segment s.
To guarantee a symmetric pattern the router creates a symmetry line and checks paths on both sides of it. For example, I assume it puts the symmetry line on the y-axis (x=0) and the router checks the symmetry of horizontal segments. It checks a segment from the coordinates(x_1, y_1) to (x_2, y_1) and a segment from the coordinates(-x_1, y_1) to (-x_2, y_1) simultaneously.

This system is admirable because of the way it represents analog constraints in its cost functions. But the resulting layout densities are sparse [CGRC91]. The size of one example is one and half times that of the manual layout. They consider wire length in the placement step, but they might not consider the regions where the wires exist. When a member of my group evaluated KOAN/ANAGRAMII with a real circuit, it couldn't realize a layout that satisfied the complicated analog constraints.

3.3 Rule Based Approach

A cooperative human-computer approach was proposed by Mogaki et al. [MSKH93]. Their goal is to provide an entire chip layout system for bipolar analog VLSI with a rule-based strategy. Their analog layout constraints are cell pair placement, crosstalk avoidance and symmetric wire patterns. Cell pair placement means the cells in a pair placement constraint must be put adjacent to each other or within a limited distance with the same orientation. Some analog layout constraints are produced by matching constraint generation rules with the circuit diagram. Then an automatic block layout works as follows. The initial cell placement is based on the circuit diagram: each cell is placed at the relative position corresponding to its location in the circuit diagram. The place improvement step determines the exact coordinates of each cell using a device position graph. In that graph the relative cell positions on the circuit diagram are represented by directed edges. The router has two features: rule-based improvement following grid-free maze routing and space expansion rerouting. The rule-based improvement step deletes redundant patterns such as wiring detours and unnecessary layer switchings. The space expansion rerouting step reduces the unrouted nets by moving existing patterns to make room for new segments.

The system is novel in that it aims for a cooperative human-computer approach. The placement approach based on the circuit diagram might be adapted to our system. The results had some unrouted nets and the density is low. In one example the chip looked about twice the size that could be achieved by manual layout. The reason for this is the lack of flexibility and global view, because a maze router determines the wire pattern of one net at a time.

3.4 Interactive Layout Tool

On the other hand an analog CMOS layout generator ILAC [RLSD89] tried to adapt an interactive approach. It considers device matching, symmetry, and distance and coupling constraints. These constraint descriptions are output by an interactive design tool IDAC [DND$^+$87]. The layout is produced by three steps: 1)primitive device level layout using a specialized layout generator, 2)cell layout with a parametric library or with an interactive way to a text file, and 3)cell placement and routing at the circuit level. Device matching constraints are handled by creating the same shapes for elements grouped them in the device level layout generator. At the circuit level, matched groups are placed with the same orientation and geometrical form. Symmetry is also handled in the device level layout and the circuit level. The precise method by which symmetry constraints are handled was not clear from the paper. Distance and coupling constraints are handled with a penalty term in the cost function used during the placement optimization process. Routing is executed by using channel routing and compaction methods. ILAC divides nets into four categories: sensitive nets such as high-impedance nodes, noisy nets such as output nodes, non-critical nets, and power supply nets. The channel router tries to minimize the parasitics of sensitive nets and to reduce the coupling capacitance between sensitive and noisy nets. They implemented symbolic level interactive improvements that allow users to change cell locations and wire patterns and to check values including the parasitic capacitance.

But their results are 25% bigger than the manual layouts on the average. The reason is maybe that users can't improve both cell locations and wire patterns simultaneously due to the channel routing and compaction methods.

4. The layout system SURF

SURF (Santa Cruz ULSI Routing Framework) [SSDD93] is an automatic/interactive layout tool. It has the following four features: 1) a flexible topological wiring representation known as a rubber-band sketch, 2) incremental design rule checking, 3) a graphical topological layout editor with on-line DRC, and 4) an automatic topological to geometrical wiring transformation. I describe each feature individually.

4.1 Rubber-band sketch

The Rubber-band router produces a Rubber-band sketch (RBS). In a Rubber-band sketch each terminal is represented as a point and each single layer point-to-point connection is modeled as an elastic rubber band which realizes the shortest path in the same topology. Thus the Rubber-band router decides arbitrary angle wire topologies (Figure 4.1).

4.2 DRC with spoke creation

SURF's incremental design rule check (DRC) uses spoke creation to enforce the proper wire width and design spacing and to obey the geometrical restrictions on wire direction. It maintains the topology of the Rubber-band sketch. This spoke creation step pushes wires near each terminal away with open-ended lines called spokes [Kon92]. An illustration for the octilinear geometry is given in Figure 4.2. The length of each spoke depends on the space rule and the width of the wire being pushed. The direction of a spoke depends on the geometrical restriction. If spokes can't be created for some terminals, it means design rule violations (DRVs) are present.

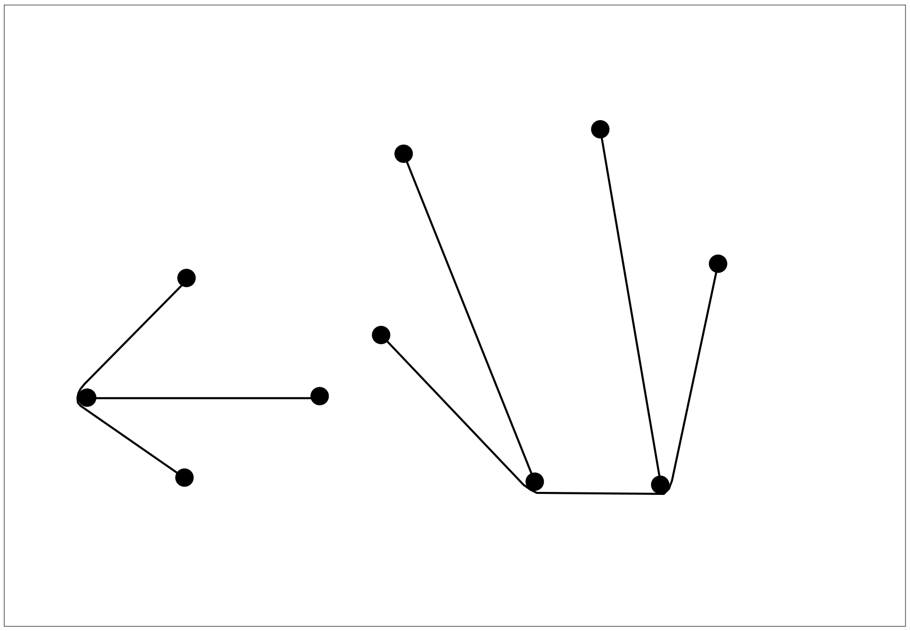


Figure 4.1: Rubber-band sketch

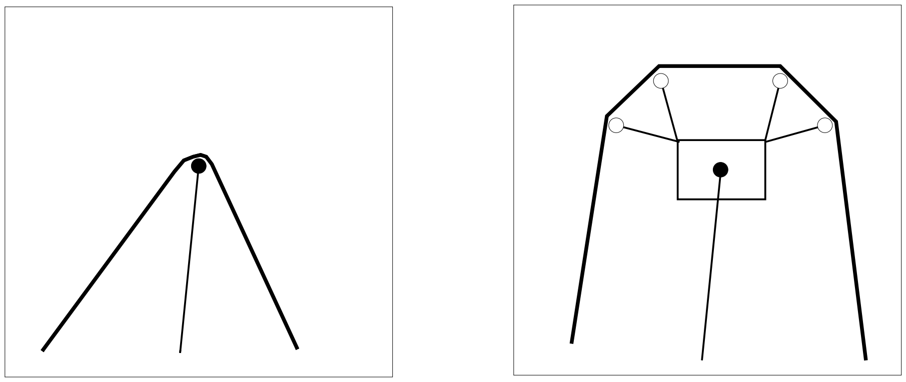
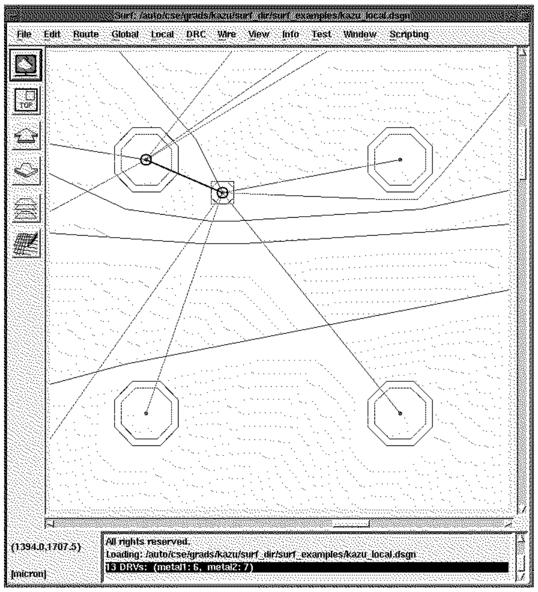
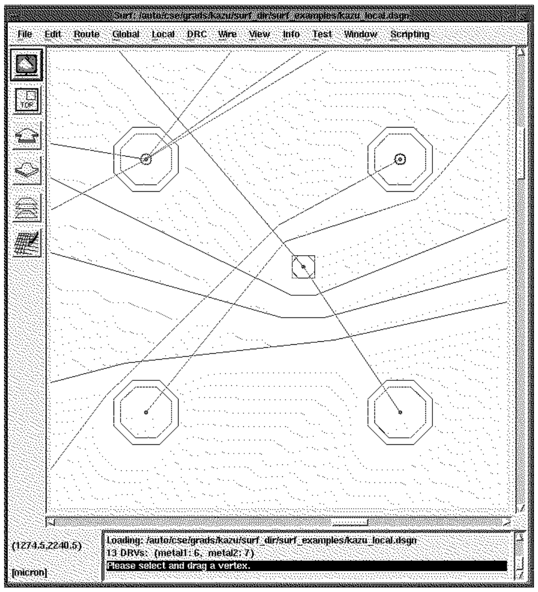


Figure 4.2: Spoke Creation


Surf: /auto/cse/grads/kazu/surf_dir/surf_examples/kazu_local.dsgn
File Edit Route Global Local DRC Wire View Info Test Window Scripting
(1394.0,1707.5)
[micron]

```
All rights reserved.
Loading: /auto/cse/grads/kazu/surf_dir/surf_examples/kazu_local.dsgn
13 DRVs:  (metal1: 6,  metal2: 7)
```

Before via movement


Surf: /auto/cse/grads/kazu/surf_dir/surf_examples/kazu_local.dsgn
File Edit Route Global Local DRC Wire View Info Test Window Scripting
(1274.5,2240.5)
[micron]

```
Loading: /auto/cse/grads/kazu/surf_dir/surf_examples/kazu_local.dsgn
13 DRVs:  (metal1: 6,  metal2: 7)
Please select and drag a vertex.
```

After via movement

Figure 4.3: Interactive operation in SURF

4.3 Manual layout editing with a good GUI

After the initial design rule check, if there are any violations, an automatic layout modifier named Automatic Resolve DRV works. It shifts the location of place-flexible components such as vias and wire junction points. It also modifies wire patterns connecting to those place-flexible components. Most of the violations are solved automatically, but sometimes some of them remain after Automatic Resolve DRV. Then the user can update the topological layout pattern interactively. This function is called topological editing in RBS. A powerful graphical manual editor allows the user to change the locations of place-flexible components and update the topology of the wire patterns. Illustrations of one interactive operation are given in Figure 4.3. In Figure 4.3 there are four fixed terminals and one via. Before the via is moved there is a DRV between the via and one terminal. The DRV is shown by a thick line. That DRV disappears after the via movement.

4.4 Geometrical view indication

Once all DRVs disappear, SURF transforms the Rubber-band sketch to the geometrical view. The geometrical view provides the final layout pattern to the user. So if there are restrictions on wire direction such as rectilinear or octilinear depending on the manufacturing process, SURF transforms the Rubber-band sketch to match those restrictions.

4.5 Extensions to SURF for analog layout and new concepts

Although the SURF autorouter and the topological editing in RBS provides a good layout solution for the digital circuit design problem, several extensions and modifications are necessary if SURF is to be successfully applied to the problem of analog and mixed analog/digital design. Specifically the following extensions need to be made:

- A more powerful obstacle representation is required to handle the complex terminal shapes and routing blockages found in analog design. This obstacle representation must support efficient design rule checking.
- Manual editing must be performed in the geometrical view instead of the one in the RBS. I call this function topological editing with the geometrical view (TEGV). Currently, the designer edits the layout in the topological view. However since the final wiring pattern produced by the geometrical transformation is different from the topological view, the designer must guess at the final effects of any editing changes. This is unacceptable for analog layout design because the designer must consider or control the final layout pattern. TEGV solves this problem. Because it is an interactive layout editing function for the user to update the analog layout pattern topologically with the geometrical view. Since the geometrical view indicates the final layout pattern including segments coordinates, the designer can update the layout pattern precisely.

I describe each function and how they work in the rest of the report.

5. Obstacles

5.1 What is an obstacle?

We must handle a variety of component shapes in analog layout. I introduce *obstacle* to treat with them. I define obstacle as polygon object which is routing blockage on a layer. Obstacle consists of devices such as transistors, resistors, and capacitors, and some parts of the devices, for example, a terminal. It also contains blocks, pads and pre-fixed wiring elements in the components. I assume the shape of one obstacle be rectilinear.

5.2 New obstacle data representation

To handle obstacles efficiently, we need an advanced data expression in which each corner of an obstacle is represented explicitly in the database with a vertex. As I mentioned previously, in the current SURF each terminal is represented as a single vertex at the center of its pad and spoke creation works based on these vertices. I depict this difference in Figure 5.1.

To explain the efficiency of this new data representation, I describe the spoke creation and DRC operation currently used by SURF in detail. When a branch to one vertex is moved to satisfy the width and spacing design rules, that branch might push other branches or move too close to other terminals. The current SURF uses region searches to locate other objects in the neighborhood of the vertex being expanded. The size of the search-region depends on the current width and spacing rules as well as the size of the largest pad in the database. The diagonal length of the search region L_d is calculated as:

$$\begin{aligned} L_d = \{&(the\ net\ width\ of\ an\ attached\ branch) + (space\ rule\ distance) \\ &+(half\ of\ the\ diagonal\ of\ the\ biggest\ object)\} \times 2 \end{aligned} \qquad (5.1)$$

The reason why the size of the largest pad is added to the search region size is because pads are modeled by a single center vertex. This extra search space is required to guarantee that the region search locates other pads that may have a design rule violation with the vertex in question. See Figure 5.2. Since every search region is bloated by the size of the largest pad, a single large pad in the database can drastically increase the number of vertices that need to be processed during DRC. If pads are represented by obstacles whose corners are explicitly stored in the database, the need for this extra search region bloating is removed. See Figure 5.3.

A drawback of the proposed obstacle representation is that the location that a wire exits from a pad must be modeled explicitly in the triangulation with a vertex. If a wire moves and its exit

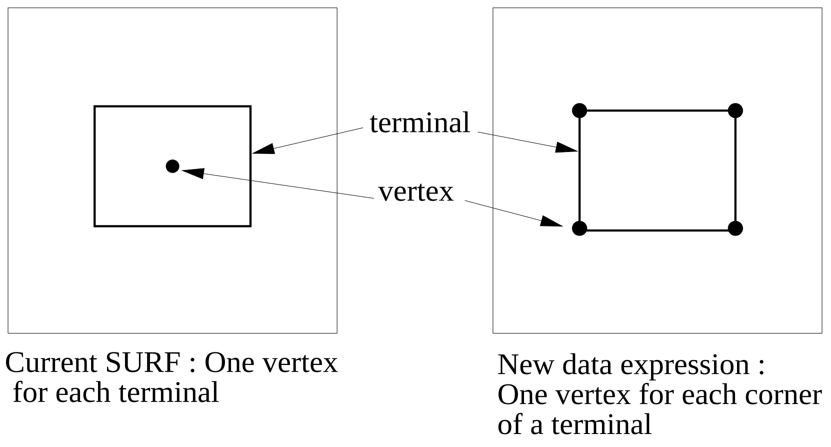
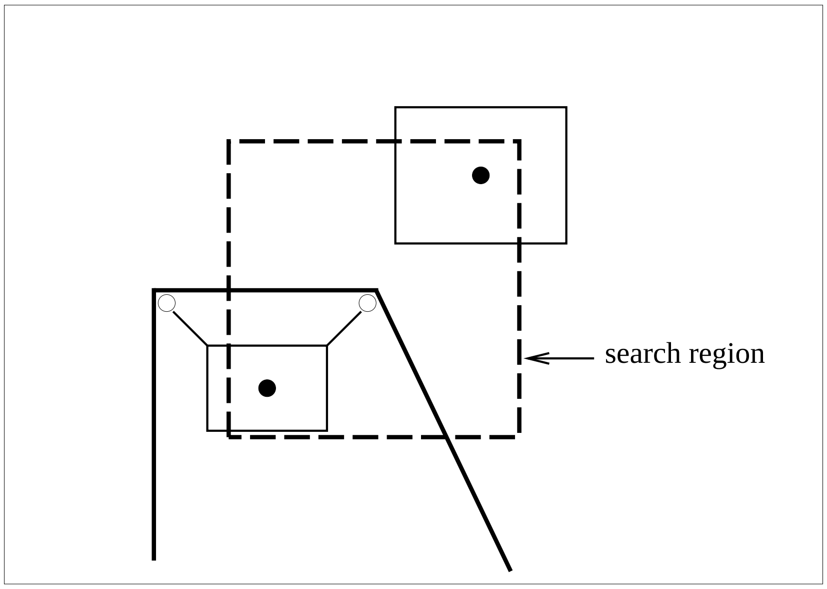
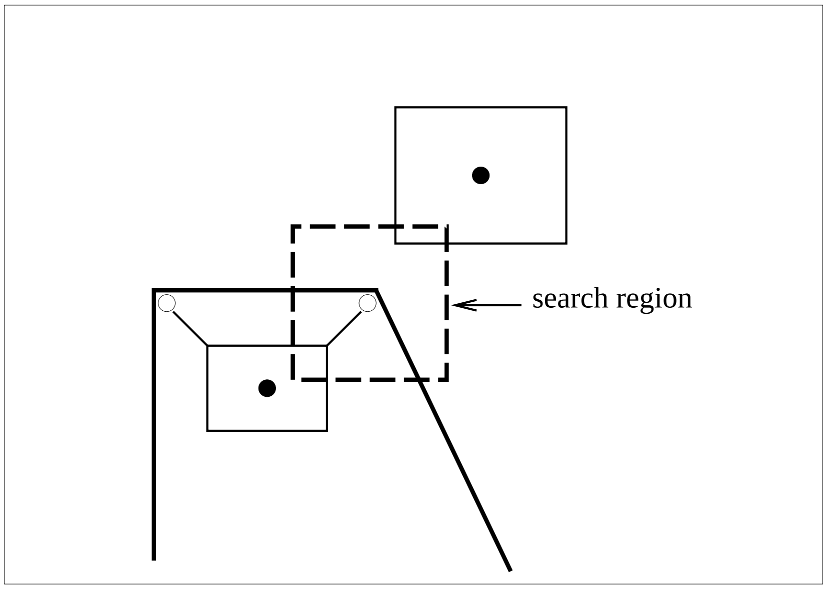

terminal
vertex
Current SURF : One vertex for each terminal
New data expression : One vertex for each corner of a terminal

Figure 5.1: The difference of data expression



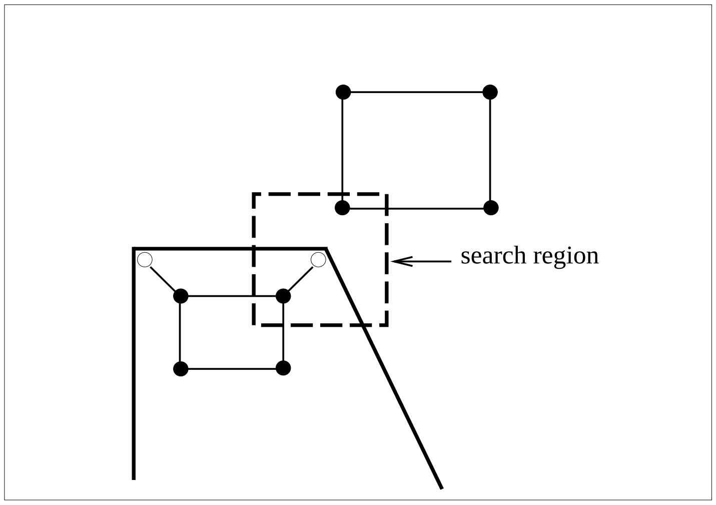
(a) A big search region can detect another object



(b) A small search region can't detect another object

Figure 5.2: Object search region with the current data expression

location is changed, the triangulation must be explicitly updated. However triangulating a single vertex has an expected complexity of $O(1)$, so this should be acceptable [Lu91]. Another drawback of explicit obstacle embedding is that the size of the triangulation is increased.



A small search region can detect another object with the new data expression

Figure 5.3: Object search region with the new data expression

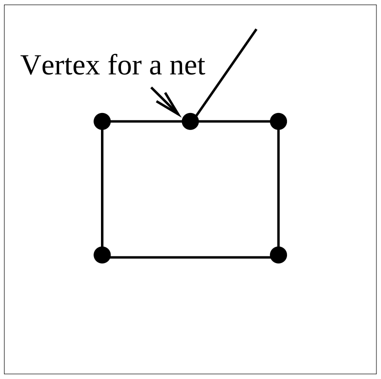


Figure 5.4: Vertices on one terminal

6. Topological editing with the geometrical view (TEGV)

6.1 What is TEGV?

The function *topological editing with the geometrical view* (TEGV) allows the user to perform topological editing operations while viewing the layout in its final precise geometrical form. The purpose of TEGV is to provide an efficient topological editing tool with the geometrical view to minimize the layout size under analog layout constraints.

The geometrical view indicates the exact layout pattern, so the user can update the pattern precisely. In addition to topological editing functions, TEGV also provides the ability to precisely control the final coordinates of the geometric layout in order to realize the user's requirements. This feature solves the common problem of geometric layout editing tools in which the user must perform time consuming polygon-level editing of the entire design. So TEGV is a novel editing tool.

TEGV is useful for both digital and analog design, but it is more useful for the latter, because analog design sometimes requires precise control of the final geometry in order to satisfy the analog constraints.

6.2 Design flow with TEGV

The entire TEGV analog layout system is based on the current SURF flow with three more operations: analog layout constraint check, topological editing with the geometrical view (TEGV), and update RBS (Figure 6.1). Analog layout constraint check is a program that determines whether a layout satisfies each analog layout constraint and reports the result. It is executed as one function of TEGV. If the analog layout constraint check finds no constraint violations or they are negligible violations, the design loop is finished and the mask pattern is generated. The update RBS function transforms the results of the TEGV step back into the rubber-band database used by the rest of SURF. It may be involved either automatically by the interactive operations described in the next section or by explicitly involved by the user. Once the RBS database has been updated, the spoke creation operation spaces the wiring to conform to the width and spacing constraints. Finally, the RBS can be retransformed back to the geometrical view for another round of analog constraint checks.

6.3 Interactive operations in TEGV

I propose adding the following interactive operations to SURF for analog layout. I explain each operation with example figures. They are categorized in two groups.
1) Analog layout constraint operations

- Display analog layout constraint information.
  If the user requests all analog layout constraint information, the components having analog layout constraints are highlighted in the layout window and a list describing the constraint specifications appears. If the user selects one component which has an analog layout constraint, that component and other components with which it shares constraints are highlighted and a constraint list for the selected component appears. For example, in Figure 6.3 the user selects component I4, then I4 and I6 are highlighted and the user can get the analog layout constraint information. Figure 6.2 shows the original layout result for reference.
- Analog layout constraint check.
  The analog layout constraint check program highlights the locations where constraint violations are detected and provides a list that indicates whether each constraint is satisfied or not. Figure 6.4 shows one example of the analog layout constraint check results. It shows the device-separation constraint is satisfied but the crosstalk between N1 and N2 is not satisfied.

2) Layout improvement operations

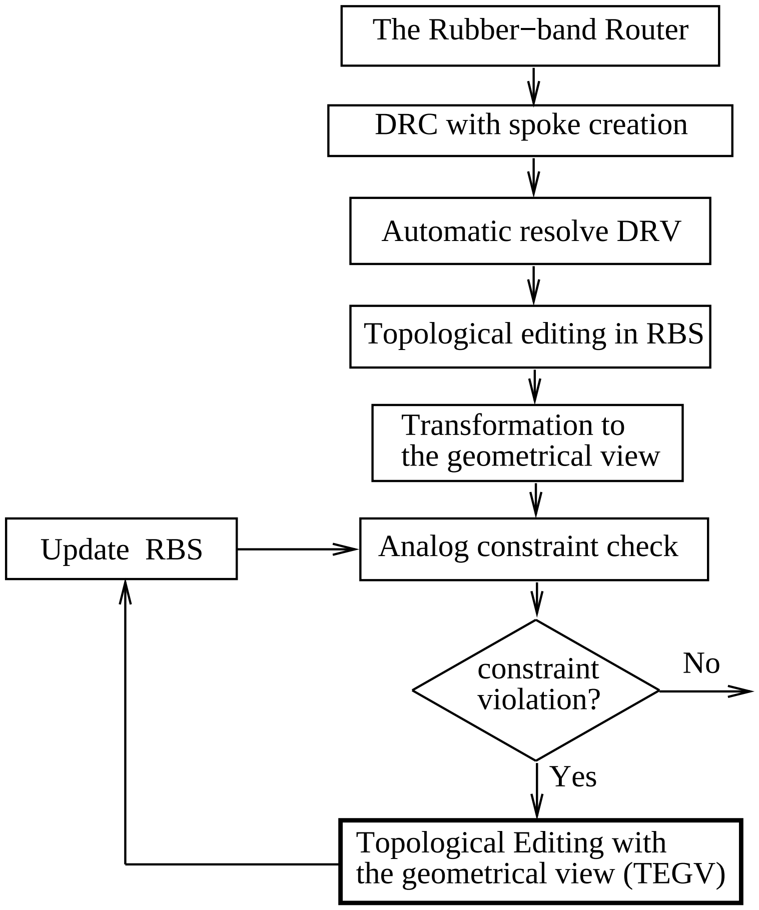


Figure 6.1: Design flow using TEGV

- Shape modification of flexible-shape component.
  The proper shape for a component may depend on its circumstances. So this function allows the user to modify the shape of flexible components by changing some parameters to form the new shape. If the component has an analog placement constraint such as device matching or mirror-symmetry, other components sharing a constraint with that component are also reshaped in the same shape. Figure 6.5 shows one example of this function. The component I4 has a device-separation constraint with I6, so when the user reshapes the component I4, the component I6 is reshaped automatically.
- Component movement.
  The user can move a component by moving an indicator in the window or by entering the coordinates of the destination location. The user can also move a group of elements in a rectangular region simultaneously. In Figure 6.6 the user moves components I2 and I3 and net N2 with one operation. If the user moves a component which has a placement constraint, such as device matching or mirror-symmetry, other components sharing a constraint with that component are also moved automatically if necessary to keep the analog layout constraint.
- Delete nets partially or entirely.
  The user can delete all segments of one net or delete some parts of one net in a region or all nets in that region. Figure 6.7 shows that one segment of net N1 is deleted. The deleted segment is represented by the shaded line.
- Topological wire modification and wire coordinates modification.
  By using not only topological relations but also the final coordinates of the net segment, the

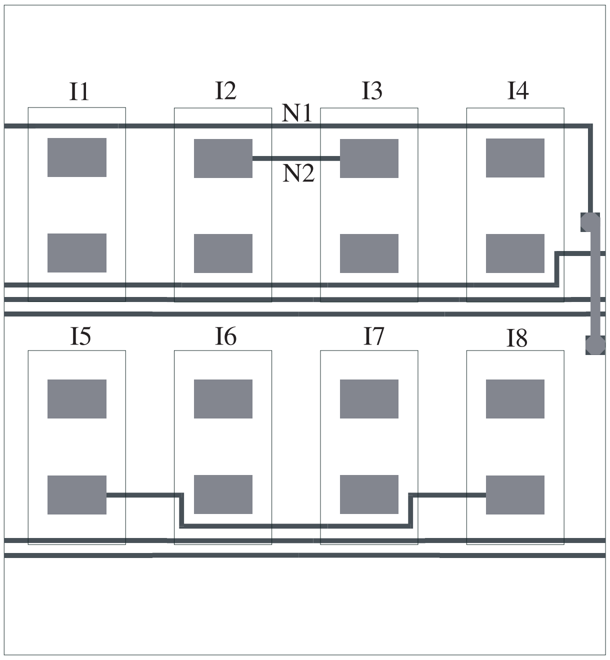


Figure 6.2: Analog layout example

user can modify the layout pattern precisely in order to avoid crosstalk or to realize wiring-cross-symmetry. In Figure 6.8 using the wire coordinates modification, the user avoids crosstalk while maintaining the same wire topology.

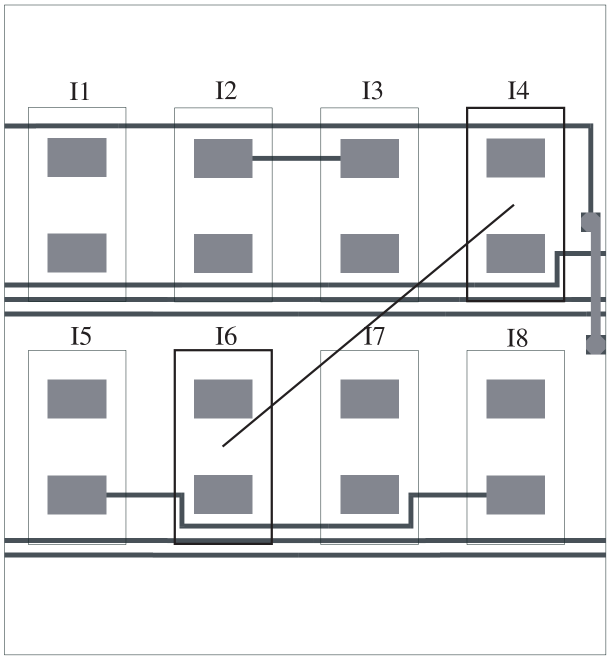
6.4 How TEGV works

6.4.1 Shape modification

When the user modifies the shape of a component using TEGV, the update RBS operation is always executed automatically to update the layout pattern. I describe a pseudo-code for this operation in Figure 6.9. This operation maintains the same wire topology as the results of the last TEGV operation. In this case a reshaped component and the nets connected to that component must be updated in RBS. If other affected or components are affected by the moved component, they are also updated.

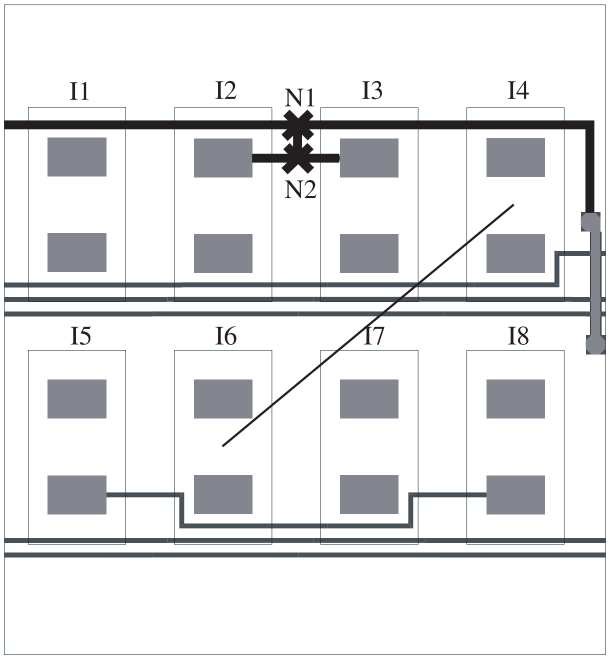
6.4.2 Component movement

This operation works similarly to the shape modification and executes the update RBS each time (Figure 6.10). If the moved component has an analog constraint and moving one component causes a constraint violation with other components, for example, it violates the allowed maximum distance in a device-separation constraint, the other components involved in the constraint violation are moved by the original movement vector. If the parallel movements causes placement DRVs, all moved components are moved from the destination location back to the original location gradually until they can avoid DRVs.



Analog Constraint Info.			
Device Separation			
ID	ID	Spec.	Result
I4	I6	80	60

Figure 6.3: When I4 is chosen, I4 and I6 are highlighted, together with the constraint specification



Analog Constraint Check result						
Device Separation						
ID	ID	Spec.		Result		Judge
I4	I6	80		60		good
Crosstalk						
ID	ID	Spec.		Result		Judge
		Sp.	L.	Sp.	L.	
N1	N2	30	20	25	40	N.G.

Figure 6.4: The constraint check result indicates the device-separation is satisfied, but the crosstalk constraint is violated.

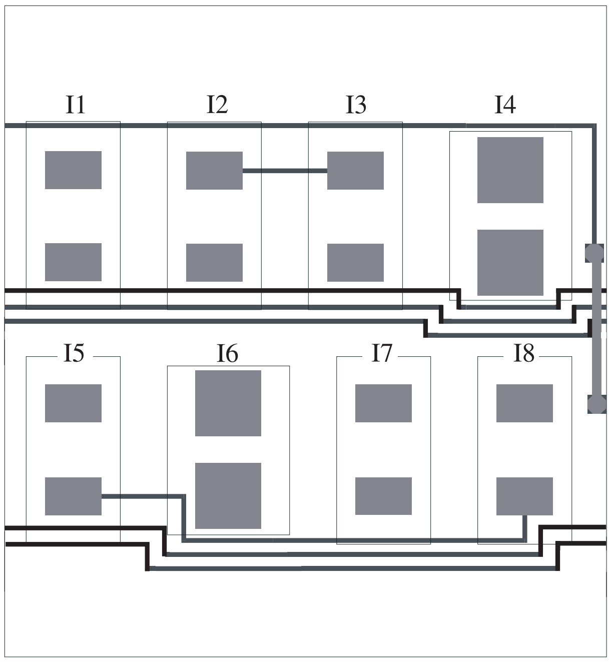


Figure 6.5: When I4 is reshaped, I6 in the same device-separation constraint is reshaped automatically.

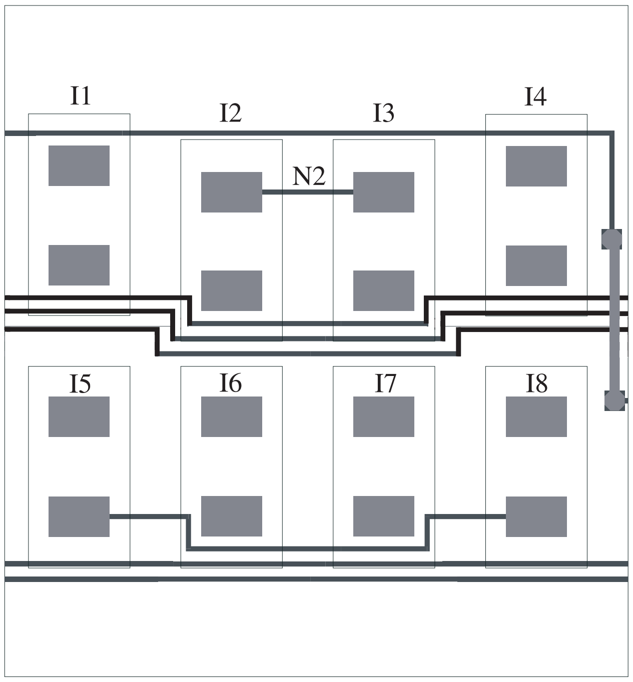


Figure 6.6: The user can move I2, I3 and N2 in one operation.

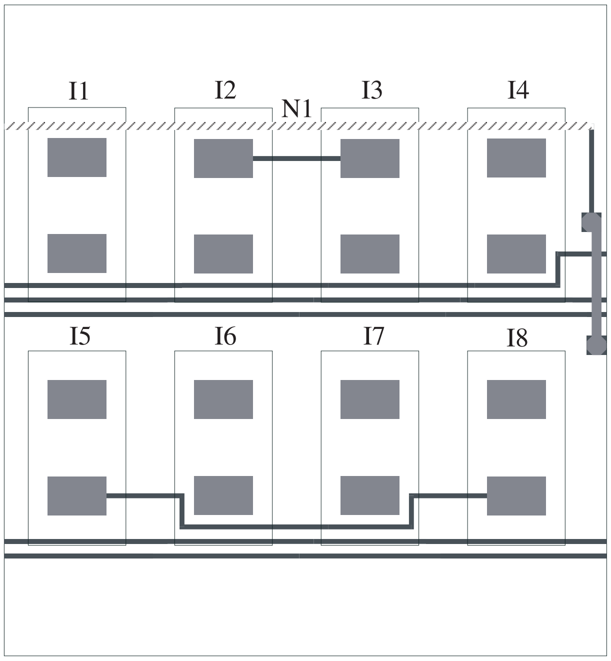


Figure 6.7: One segment of N1 is deleted and it is shown by a shaded line.

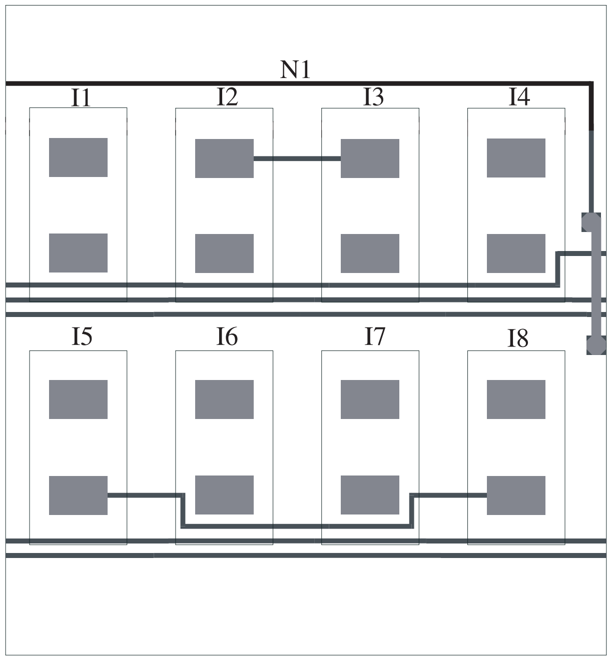


Figure 6.8: The net N1 is added by the wire coordinates modification function.

```
RESHAPE_COMPONENT( C )
/* C : a component to be reshaped */
{
    Change the shape of C;
    Update RBS;
    If ( C has an analog constraint ){
        Foreach ( Component C' in that constraint ) RESHAPE_COMPONENT( C' );
    }
}
```

Figure 6.9: Pseudo code for RESHAPE_COMPONENT

```
MOVE_COMPONENT( C )
/* C : a component or some components to be moved */
{
    Move a component or components C;
    Move_back_distance = 0;
    If ( Movement of C causes analog constraint violation)){
        Foreach ( Component $C_{move}$ in that constraint ){
            Set the location of $C_{move}$ to Ori;
            Move $C_{move}$ with the same moving vector of C and set that location to Dst;
            If ( movement of component $C_{move}$ causes placement DRV )
               Calculate the move back distance of $C_{move}$ from Dst to Ori to avoid DRV;
            If ( (The current move back distance) > Move_back_distance )
               Move_back_distance = (The current move back distance);
        }
        If ( Move_back_distance > 0 ){
            Move C with the distance Move_back_distance to the opposite direction
           against the moving vector;
            Foreach ( Component $C_{move}$ in that constraint ){
                Move $C_{move}$ with the distance Move_back_distance to the same direction
                 with the movement of C;
            }
        }
    }
    Update RBS;
}
```

Figure 6.10: Pseudo code for MOVE_COMPONENT

6.4.3 Recognition of a hint-point in wire pattern modification

There are two operations for modifying wire patterns: wire topology modification and wire coordinates modification. Before I explain how TEGV works for each modification, in this subsection I show how TEGV should recognize hint-points to transform the user input information in TEGV to the RBS. A hint-point is a point input by the user to indicate or guide the route of a net.

When the user enters a hint-point, TEGV must recognize two properties: the number of branches and the crosspoint. Both of them are important, because branches are checked by spoke creation in RBS and having fewer branches to be checked makes spoke creation run faster. Without properly recognizing crosspoints the resulting modification might not correspond to the user's intentions.

Now I describe the two properties in detail.

(1) Number of branches

Once a net or part of a net is deleted, the program divides that net into two subnets. A subnet is a terminal or a segment between two hint-points. The number of branches in a net depends on how the user connects the terminals, because a sequence of branches on the same layer can be merged into one branch. For example, in Figure 6.11 if netA is modified on just one layer with four hint-points (X_{a1}, Y_{a1}), (X_{a2}, Y_{a2}), (X_{a3}, Y_{a3}), and (X_{a4}, Y_{a4}) (Figure 6.11(d)), then the number of subnets is three. They are merged into one branch. But in the case of Figure 6.11 (f), the number of the branches in the first layer is two and they are transformed separately to update RBS, because they are not continuous.

(2)Crosspoint

TEGV can recognize a crosspoint between a subnet and other same layer wire elements by the following rule. I depict this rule using an example in Figure 6.12.

Rule 1 If a subnet (subnetA in Figure 6.12) has a crosspoint with another object of the same layer (segmentB in Figure 6.12) and that crosspoint exists on the left half of the object, then the subnet goes to the left side of the object (Figure 6.12(b)). Otherwise, the subnet goes to the right side of the object (Figure 6.12(c) and (d)).

6.4.4 Wire topology modification

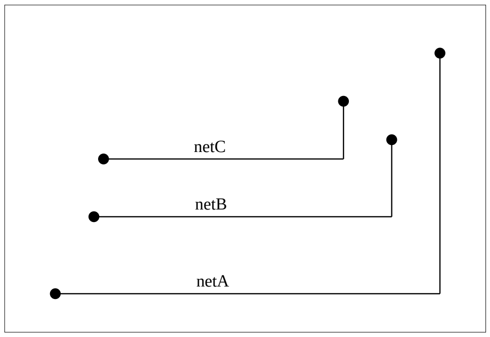
The user enters hint-points in order to choose a new route for a deleted net. The previous routes of the deleted nets are shown by shaded lines as a guide. See netA in Figure 6.11(b). After connecting the deleted net, the user invokes the update RBS, and the continuous same layer subnets in one net are merged into a branch.

I describe the pseudo-code to realize this modification including recognition of the two hint-point properties in Figure 6.14. In Figure 6.14 S_branch is a set of branches to be transformed to the update RBS. The crosspoint check is done in lines from 12 to 18 and the relative location with the other non-crossing object is checked in line 19. That check can be executed as follows. First I make a horizontal segment with length X_{width}, where X_{width} is the sum of the length of the subnet, the width of the net, and twice the space rule. Then I sweep that segment tupwards and downwards until it catches other objects(Figure 6.13).

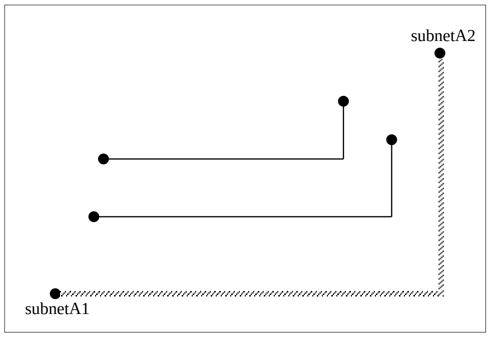
6.4.5 Wire coordinates modification

If the user applies the fixed wire coordinates modification in TEGV, the pseudo-code in Figure 6.15 is executed and S_branch is produced. In this modification there are no subnet merges and each subnet becomes one branch. This modification doesn't allow crosspoints and if there are crosspoints of a subnet and other objects, TEGV shows a warning to urge a new coordinates choice that avoids crosspoints (6 and 7 lines in Figure 6.15).

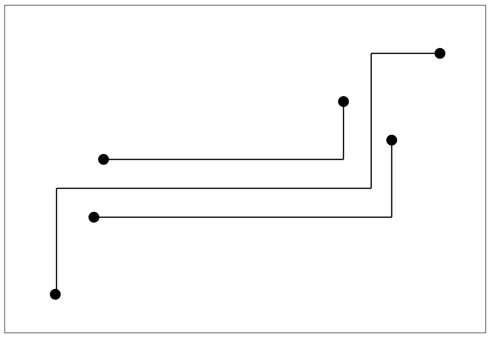
The update RBS is invoked explicitly by the user. In the RBS, spoke creation tries to place each segment on the specified coordinates. If there is space between the coordinates indicated by the user and the segment coordinates calculated by the design rules during spoke creation, SURF puts the



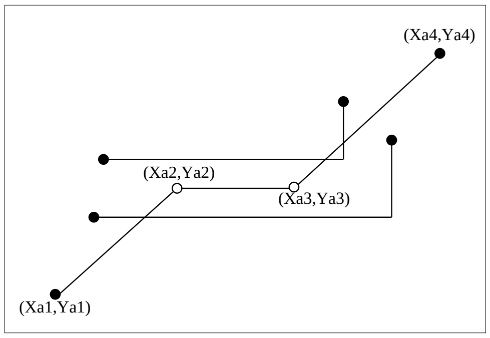
(a) The original layout pattern



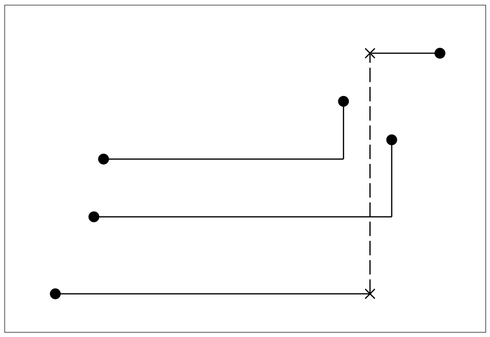
(b) The layout pattern after deleting segments of netA



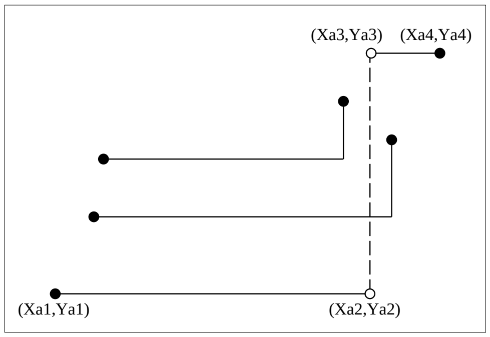
(c) A required pattern with one layer



(d) Operations for modification of (c)



(e) A required pattern with two layers



(f) Operations for modification of (e)

" $\bullet$ " : the center point of a terminal. " $\circ$ " : hint-point not on the terminal.
line : the first layer. dot line : the second layer. $\times$: via between the first layer and the second layer.

Figure 6.11: Subnet number for wire pattern modification

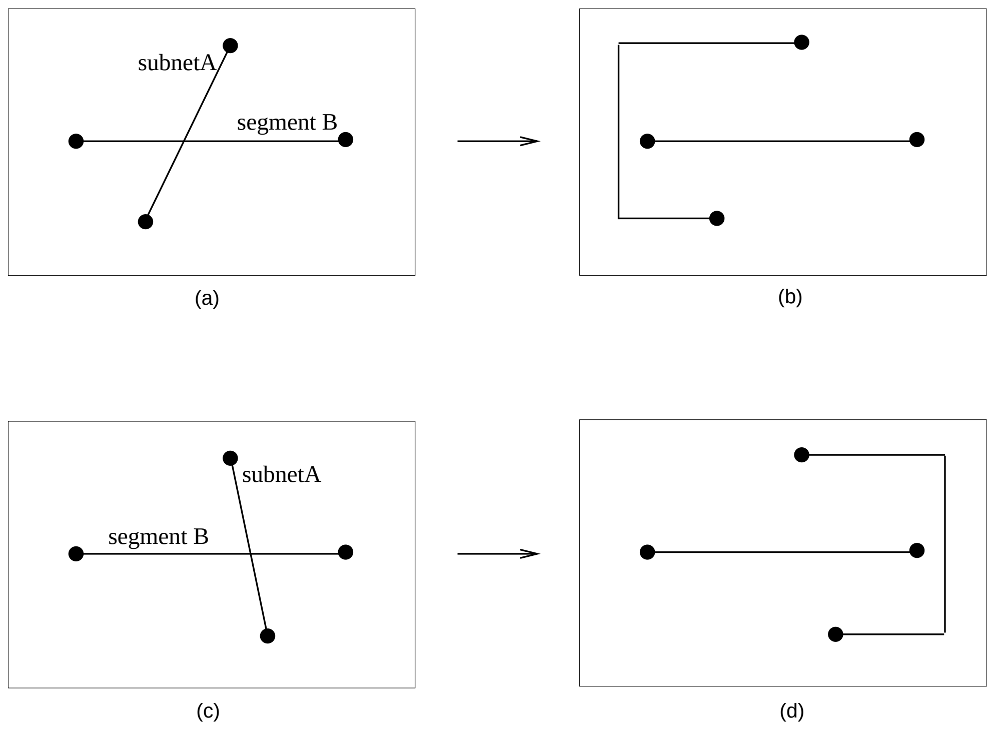


Figure 6.12: A rule about crosspoint

segment on the indicated coordinates. Otherwise SURF shows an error message in the geometrical view saying the fixed coordinates can't be achieved.

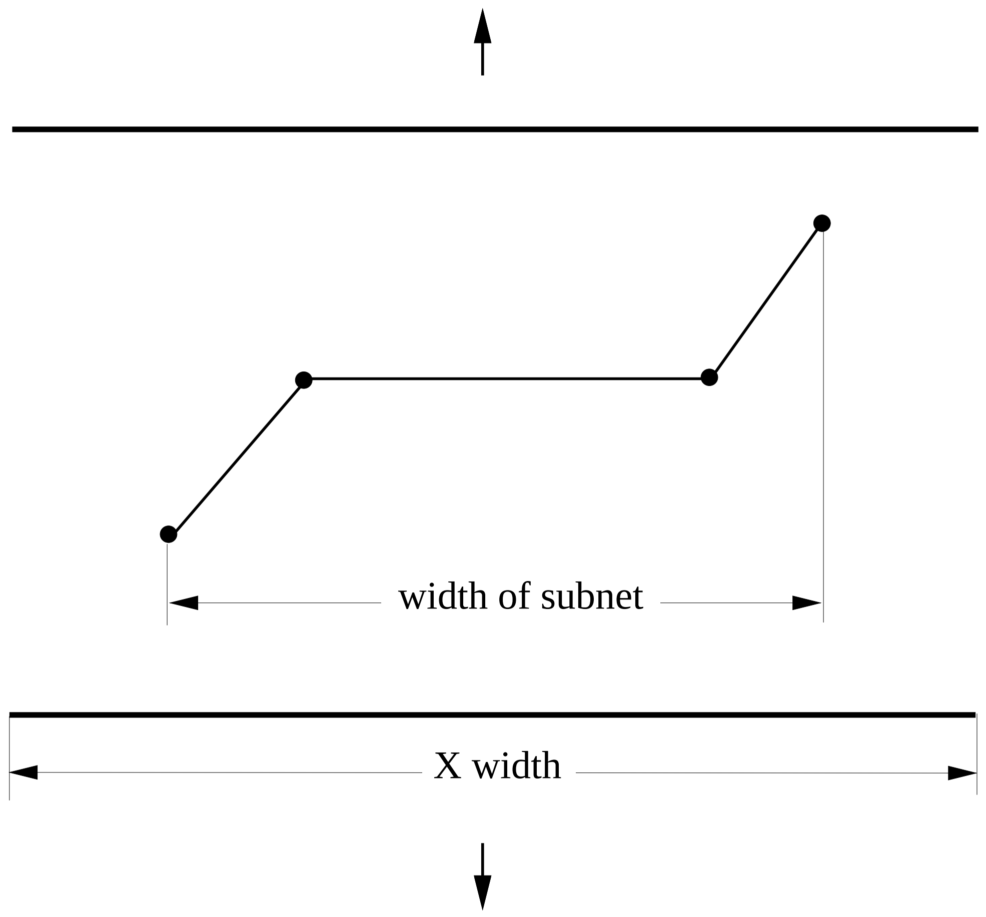


Figure 6.13: Check with non-crossing objects

```
WIRE_TOPOLOGY_MODIFICATION( XY_hint, S_branch)
/* XY_hint : coordinates of hint-point, S_branch : set of branches */
{
    If ( XY_hint is on the terminal of a new net ) {
        Initialization for a new branch;
    }else{
        /* layer analysis */
        If ( a hint-point layer is different from the previous one ){
            If ( there is a previous branch ) Add previous_branch to S_branch;
            Initialization for a new branch;
        }
        /* topology analysis */
        If ( a new subnet has a crosspoint with other objects){
            If ( the crosspoint is located in the right half of the object){
                Recognize the new subnet runs at the right side of that object;
            }else{
                Recognize the new subnet runs at the left side of that object;
            }
        }
        Check the relative location between the new subnet and neighbors;
        /* Last terminal */
        If ( XY_hint is on the terminal ) Add current_branch to S_branch;
        }
    }
}
```

Figure 6.14: Pseudo code for WIRE_TOPOLOGY_MODIFICATION

```
WIRE_COORDINATES_MODIFICATION( XY_hint, S_branch)
/* XY_hint : coordinates of hint-point, S_branch : set of branches */
{
    If ( XY_hint is on the terminal of a new net ) {
        Initialization for a new branch;
    }else{
        If ( a new subnet has a crosspoint with other objects){
            Make_Warning_message;
        } else {
            /* new subnet */
            Add subnet as a branch to S_branch;
            Initialization for a new branch;
        }
    }
}
```

Figure 6.15: Pseudo code for WIRE_COORDINATES_MODIFICATION

7. Summary and Future Work

7.1 Summary

This report provides a survey of the previous work for analog layout and proposes new concepts for efficient and user friendly interactive analog layout. To bridge the existing gap between SURF and the requirements for analog layout, I introduced topological editing with the geometrical view (TEGV). I clarified the necessary operations in TEGV and their function specifications.

7.2 Future Work

Although my proposed ideas should work efficiently, we must check the efficiency after implementing TEGV including the analog layout constraint check step.

There is one more future work to make SURF practical for both digital and analog layout designs. It is development of an automatic layout program that considers analog layout constraints. In my survey I found a couple of good features in other systems, for example, circuit diagram relative placement and how to incorporate analog constraints into the entire cost evaluation. We should develop an automatic layout program that includes these good features and harmonizes with TEGV.

References

[CGRC91] John M. Cohn, David J. Garrod, Rob A. Rutenbar, and L. Richard Carley. KOAN/ANAGRAMII: New tools for device-level analog placement and routing. *IEEE Journal of solid-state circuits*, 26(3):330–341, March 1991.

[CGRC94] John M. Cohn, David J. Garrod, Rob A. Rutenbar, and L. Richard Carley. *Analog device-level layout automation.* Kluwer Academic Publishers, 1994.

[CSV90] Umakanta Choudhury and A. Sangiovanni-Vincentelli. Constraint generation for routing analog circuits. In *Proc. 27th IEEE/ACM Design Automation Conf.*, pages 561–566, 1990.

[CSV93] Umakanta Choudhury and Alberto Sangiovanni-Vincentelli. Constraint-based channel routing for analog and mixed analog/digital circuits. *IEEE Journal on computer-aided design of integrated circuits and systems*, 12(4):497–510, April 1993.

[DND$^+$87] Marc G. R. Degrauwe, Olivier Nys, Evert Dijkstra, Jef Rijmenants, Serge Bitz, Bernard L. A. G. Goffart, Eric A. Vittoz, Stefan Cserveny, Christian Meixenbeger, Van Der Stappen, and Henri J Oguey. IDAC: An interactive design tool for analog CMOS circuits. *IEEE Journal of solid-state circuits*, sc-22(6):1106–1115, 1987.

[Kon92] Raymond Kong. Constructive routability test for planar topological routing. Master's thesis, University of California, Santa Cruz, 1992.

[Lu91] Yizhi Lu. Dynamic constrained delaunay triangulation and application to multichip module layout. Master's thesis, University of California, Santa Cruz, 1991.

[MSKH93] Masato Mogaki, Youichi Shiraishi, Mitsuyuki Kimura, and Tetsuro Hino. Cooperative approach to a practical analog LSI layout system. In *Proc. 30th IEEE/ACM Design Automation Conf.*, pages 544–549. IEEE Computer Society Press, 1993.

[RLSD89] Jef Rijmenants, James B. Litsios, Thomas R. Schwarz, and Marc G. R. Degrauwe. ILAC: An automated layout tool for analog CMOS circuits. *IEEE Journal of solid-state circuits*, 24(2):417–425, 1989.

[SSDD93] David Staepelaere, Jeffrey Su, Tal Dayan, and Wayne Wei-Ming Dai. Surf: A rubber-band routing system for multichip modules. *IEEE Design and Test of Computers*, pages 18–26, December 1993.