An Automatic Layout Generator
for Integrated Circuit Design

by
Lan Lin
B.Eng., Nanjing University of Aeronautics and Astronautics, 1996
M.Eng., Nanjing University of Aeronautics and Astronautics, 1999

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
Master of Science
in
THE FACULTY OF GRADUATE STUDIES
(Department of Computer Science)

we accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA
August 2001
© Lan Lin, 2001
In presenting this thesis/essay in partial fulfillment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying for this thesis for scholarly purposes may be granted by the Head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science  
The University of British Columbia  
2366 Main mall  
Vancouver, BC  
Canada V6T 1Z4
In integrated circuit design, one of the most tedious and time-consuming steps is the generation of the layout. During the last decade, considerable effort has been invested in the development of CAD tools dedicated to the automation of this step. This effort has been largely motivated by a need for alternatives to manual layout to greatly reduce the development time and cost. This thesis describes my contribution through the implementation of a flexible and automatic integrated circuit layout generator. With this tool, the designer only needs to depict the circuit at a high level, while the tool works out the details of the design and produces the final layout. In comparison with most of the current layout synthesis tools, my tool aims to realize the generality while still preserving most of the efficiency of the hand design, and facilitate greater reuse. The solution is based on constraint solving. The tool is written in Java. Two architectural styles are followed in the whole design, call-and-return and object-oriented. Experimental results demonstrate the effectiveness of the tool in generating layouts comparable to manual designs, with very quick turn-around time and no manual intervention.
Table 2.1 Interface Signal ................................................................. 17
Table 2.2 Class Circuit ................................................................. 18
Table 2.3 Class Row ................................................................. 18
Table 3.1 Class GeomRowProto ......................................................... 25
Table 4.1 Class ConstraintSet ............................................................. 33
Table 4.2 Interfaces LPyfactory and OPFactory, Classes Lpabo and Opt .................. 35
Table 5.1 Class MagicOutput ............................................................. 39
Table 5.2 Class MagicWriter ............................................................. 40
Table 6.1 Example Cell Density ........................................................... 45
Table A.1 Index of Layout Generation Tool Source .................................. 52
Table B.1 Index of Layout Generation Example Source .............................. 101
Table C.1 Index of LPABO Input and Output Data Format ............................. 106
LIST OF FIGURES

Figure 1.1 The Architecture of the Tool ................................................................. 8
Figure 2.1 A One-bit Full Adder (FA) .................................................................... 9
Figure 2.2 Implementation of A One-bit Full Adder ............................................. 10
Figure 2.3 User Specification of A One-bit FA ....................................................... 11
Figure 2.4 An n-bit Ripple-carry Adder ................................................................. 12
Figure 2.5 Code Fragment of An n-bit Ripple-carry Adder .................................... 12
Figure 2.6 Implementations of the Signal Interface ............................................... 13
Figure 2.7 Subclasses of Circuit ........................................................................... 14
Figure 2.8 Class Hierarchy of Row ..................................................................... 15
Figure 2.9 Code Fragment of Class SumBlock ..................................................... 20
Figure 2.10 Code Fragment of Class OneBitFA .................................................... 21
Figure 3.1 Object and Constraint Generation ...................................................... 23
Figure 3.2 Member Variables of Class GeomRow ................................................. 24
Figure 3.3 Device Merging Cases and Transistor Building Bricks ....................... 28
Figure 6.1 Example Layout for One Single Inverter Placement ......................... 42
Figure 6.2 Example Layout for Inverter Row Placement .................................... 43
Figure 6.3 Example Layout for Inverter Array Placement .................................. 43
Figure 6.4 Comparison of Generated Layout with Manual Design for FifoStage .. 44
I would like to gratefully acknowledge my supervisor, Dr. Mark Greenstreet, for opening the new field for me, as well as for staying with me with extensive patience and encouragement with every bit of my progress during the past year. This work would not have been possible without his enthusiastic supervision, invaluable guidance, and generous financial support. I also wish to thank Dr. Mike Feeley and Dr. Kris De Volder for taking the time being my second readers, and for providing me with many insightful comments.

I owe my gratitude to Albert Lai, who taught me many valuable lessons in this project, and was always generous in his help. Special thanks go to Dr. Alan Hu, Brian Winters, Alvin Albrecht, Marius Laza, and my project partner, Yang Lu, for discussing every problem with me with their great ideas and creating such a harmonious atmosphere in the ISD lab.

I am grateful to all my friends at UBC, for being my surrogate family during the years I have stayed here and for their continued moral support. Their friendship makes me feel warm and makes my life in this alien land much easier.

Finally I am forever indebted to my parents, Tao Lin and Xiuying Zhou, my twin brother, Song Lin, and my husband, Fengguang Song. My present status in life, this M.Sc. included, is a result of their extraordinary will, efforts and sacrifices. I am only that much more grateful for all that they have done for me. Mere words cannot express my deep appreciation enough.

Lan Lin

The University of British Columbia
August 2001
To my parents, my twin brother, and my husband,
for their endless love, understanding and support.
Modern technology changes toward deep sub-micron process technologies, and the accompanying increase in the complexity of VLSI designs, have driven an ongoing conversion from hand design to the extensive use of computer aided layout tools to support the layout process. Manual layout using polygon layout editors is both time-consuming and error-prone; therefore, the development of automatic layout tools is an area of ongoing interest to free the designers from design rule considerations and allow them to focus on the topology of the design, thus increasing design productivity [Dunlop 80] [Lotvin 84] [Marple 88] [Dao 93].

1.1 Motivation and Problem Statement

In integrated circuit design, one of the most tedious and time-consuming steps is the generation of the layout. Invariably, it requires that the designer be quite familiar with the physical design process, and the impact of layout geometry on circuit functionality and performance. Such a designer must know in advance every detail of the design process. During the last decade, considerable effort has been invested in the development of CAD tools dedicated to the automation of this step. This effort has been largely motivated by a need for alternatives to manual layout to greatly reduce the development time and cost.

Although the regularity of some inner structures in these circuits has made such automation more feasible, it is not a trivial problem to automatically generate layouts competitive with handcrafted ones. The main problems are the increasing number of gates to be placed and interconnected, the optimization of the global size of the chip [Mathias 98], and the availability of more compact layouts while still ensuring performance close to that of a manual design.

Indeed, the goal of any technique, whether it is behavior-, logic-, or layout-related, is to produce optimum circuits that meet the designer's specifications. By
optimum circuit, I mean an implementation where the designer and the tool achieve the best tradeoff of design metrics for the entire circuit. Important metrics for CMOS circuits today include delay (e.g., input to output propagation, clock period), silicon area, and power [Lefebvre 97].

As Steve Duvall pointed out in his talk at UBC Commerce in 1999 [Duvall 99], “optimization has, or promises to have, a significant impact on design productivity”. Physical design involves the placement of circuit blocks and transistors on the integrated circuit “floorplan” and the routing of wires providing the electrical connections among these elements. Automatic place and route tools, which have been in use by the semiconductor industry for years, are faced with new challenges as the need for performance-based physical design has increased and as new layout strategies have been developed. Furthermore, circuit design involves the translation of logic functions into gate- or transistor-level equivalents. Until recently, most circuit design was performed manually by design engineers using schematic editors and circuit simulators. With the increasing magnitude of the circuit design problem, the industry has been moving toward automation of most of the circuit design process. This leads to the development of automatic circuit optimization capabilities spanning a broad class of problems, from transistor and wire sizing to direct synthesis of circuit topologies from logic statements. Last but not least, architectural design involves specifying the high level structure of an integrated circuit, which remains a largely manual process. Because architectural design decisions have an overwhelming impact on downstream design activities, models, tools, and techniques for architectural-level optimization are gaining more and more attention.

1.2 Related Work

The most common approaches to layout automation have been layout generators, recompaction of existing libraries, and automatic layout synthesis [Lefebvre 97]. Many papers have been published in this area and several systems developed in industry have covered a variety of layout styles and circuit structures and addressed many practical problems [Hill 85] [Wimer 87] [Chen 89] [Domic 89] [Ong 89] [Poirier 89] [Tani 91] [Sadakane 95] [Gupta 96] [Guruswamy 97].
Procedural layout generators are a means of capturing the layout design in a somewhat design-rule independent fashion. However, each cell has to be supported by its own specific generator, which must be created by an experienced cell designer. On the other hand, generators also tend to be unfriendly to drastic changes in cell architecture and interconnect technology [Lefebvre 97].

Compaction of existing layout data provides a somewhat more elegant solution. A significant drawback mitigates the benefit by requiring the availability of existing layouts substantially the same as the intended library as a starting point. Again, like procedural generators, compactors do not lend themselves well to architectural changes [Lefebvre 97].

Layout synthesis, in contrast to the above methods, is concerned with creating layouts starting with only a transistor level netlist. Without requiring any pre-existing specific layout information, it provides an overall flexibility in the context of future advances in the state of the art in circuit synthesis. Nevertheless, in some cases, these solutions are not fully automatic, are inadequate for standard cell synthesis, or are obsolete due to rapidly changing process and design technology. Finally, no system has been reported in the literature that provides robust solutions to the practical problems essential to synthesizing high performance layout in a fully automatic manner with handcrafted density in current process technologies [Guruswamy 97].

The realization of any custom circuit requires three primary implementation phases:

- Creation of a transistor circuit topology that provides a specific digital function. By topology I mean the allocation of N and P type transistors (or other devices) and their interconnections.
- Sizing and ordering the transistors in the circuit topology.
- Placing, routing and compacting the above transistors into a layout.

All of the phases involve tradeoffs of design metrics that must be optimized not only within each phase but also across all phases [Lefebvre 97] (e.g., performance-driven transistor placement, performance-driven detailed routing).
CAD layout tools often represent transistors and contacts as the intersection of polygons on separate layers (e.g., Magic\(^1\) does this with its "active" and "contact" layers). More recently, the trend has been toward object-based layout [Matheson 85] [Draney 89] [Duh 95] [Lakos 97]. Interesting advances have also been achieved in the field of constraint generation [Mogaki 91] [Choudhury 93]. In treating devices as objects and grouping semantically related geometry on multiple layers to form a single cohesive unit, we manipulate devices at a level of abstraction higher than that of intersecting polygons. The netlist is also explicit in the internal representation. Each device type can be parameterized to form a generator. In this sense, a single generator element can adapt to any valid size based on the current value of its arguments, which further increases the flexibility [Lakos 97].

1.3 Overview of the Current Solution

This thesis describes my contribution through the implementation of a flexible and automatic integrated circuit layout generator, the purpose of which is to exempt the designer from the tedious and time-consuming routine of manual layout. With this tool, the designer only needs to depict the circuit at a high level, while the tool works out the details of the design and produces the final layout. The solution is based on constraint solving. The tool is written in Java. From a software engineering perspective, I apply object-oriented methodology to VLSI chip design. Classes and interfaces are both carefully constructed to share the properties of abstraction, encapsulation, openness, and equilibrium, and follow the two principle aspects of OO design [Lea 94]:

- **The external, problem-space view**: Descriptions of properties, responsibilities, capabilities and supported services as seen by software clients or the outside world.
- **The internal, solution-space view**: Static and dynamic descriptions, constraints, and contracts among other components, delegates, collaborators and helpers, each of

----
\(^1\) Magic is an interactive layout editor supporting on-line design rule checking and circuit extraction. It was developed by the group of John Ousterhout at the University of California at Berkeley [Ousterhout 84].
which is known only with respect to a possibly incomplete external view (e.g., a
class, but where the actual member may conform to a stronger subclass).

The architecture of my tool is illustrated in Figure 1.1 along with the design flow.
Applying the user API, the designer writes a Java program to describe the design. In his
eyes, the circuit consists of a list of electrical connections and a list of relative
placements. Under the user interface, four packages are defined to turn this high-level
abstract description into its concrete geometry generation:

- **The Circuit package**: Provides the user-level API to describe the circuit.
- **The Constraints package**: Focused on constraint generation and solving.
- **The Router package**: Does the routing.
- **The Magic package**: Outputs the circuit in Magic format.

As stated in Section 1.2, most of the previous layout synthesis tools have two
common problems. First, they are normally designed for a schematic layout style, and
thus are inefficient in producing general layouts. For example, some regular inner
structures, such as the memory cell and the data-path, have their corresponding synthesis
tools. However, the designer needs to define and figure out details of the remaining
irregular part of the whole layout. Second, they are usually too specific in a domain.
These synthesis tools, such as those for the Arithmetic Logic Unit (ALU), the register
file, and the arithmetic units in digital signal processing, all have a few parameters in
their templates. In this way, these tools write into the code the topological solution,
which makes them work in their own very narrow application domains. There are 10 to
20 such tools to produce efficient layouts for different parts of the whole circuit design.
Nevertheless, for the other parts, designers still need to do them by hand, which is quite
inefficient. In comparison, our tool aims to realize the generality while still preserving
most efficiency of the hand design. It can relieve the designer from much low-level
design tedium and generate more reusable components and promote reuse for any kind
of manual design. The first step in using our tool may take the designer some time to
describe the geometry of some building blocks, but later steps are simplified by assembling such blocks, which facilitate greater reuse.

1.4 Thesis Outline

This chapter gives an introduction of the problem to solve, the basic idea behind my work, the overview of my tool, and its main contributions. In Chapter 2, I explain the user-level API. It is what the designer uses to write an abstract high-level description of the circuit that she desires to build. Chapter 3 presents the interfaces to the primitive object generator and the Router package. Chapter 4 describes the interface to the Linear Program (LP) package. Chapter 5 illustrates the Magic Interface. Chapter 6 describes some experiments with this tool and presents some test results. Finally, I present my conclusions in Chapter 7 along with some suggestions for possible future research. Appendices A and B contain listings of all the source code for the prototype of the tool.

1.5 Contributions

Today's highest performance designs require extensive manual layout and layout optimization for performance reasons. Tomorrow's designs will require a similar level of physical design quality simply to ensure that the circuit works at all, due to the challenging electrical problems of DSM (Distributed Shared Memory) design. However, the design time for full-custom layout is prohibitive, and full-custom design flows tend to be incompatible with the verification required for complex designs. My supervisor, Mark Greenstreet, has the conjecture that a novel approach could be developed to raise physical design to a higher level, automating many tasks such as routing, compaction, and transistor and wire sizing. My work is to test the hypothesis and prove that we can do layouts productively. Preliminary results on small circuits are approaching the density of full-custom, expert-crafted layouts with far less effort. The key insight in this research is that behavioral descriptions are not an abstraction of physical designs. In practice, designers employ abstraction hierarchies for behavioral, geometrical, and timing aspects of their designs. While traditional CAD tools focus on the behavioral to sub-optimal
designs and/or manual layout, my tool aims to provide a general high-level solution. I come up with a novel exploratory implementation of a practical piece of nontrivial software. The primary contributions of my work lie in:

- The specification, design and implementation of an architecture and prototype for the tool (Chapter 2, 3, 4, 5).
- Demonstration of the automation of a layout optimization typical of handcrafted layout. In particular, allowing diffusion and contact sharing in transistor placement and geometry generation (Chapter 3).
- Demonstration of parameterized layout objects and the encapsulation of design rules into a separate module. In particular, flexibility in drawing optimal layouts with user-defined signal width and transistor width, and in supporting a wide variety of process technologies (Chapter 2).
Figure 1.1 The Architecture of the Tool
This chapter illustrates the user-level API. As for many other layout automation systems, the inputs to my tool consist of a netlist of sized transistors and signals with their interconnections, as well as a list of relative placements, which define the layout topology. The drawing methodology is based on placing small elementary parts side by side or below each other, while allowing diffusion and contact sharing along either side of generated transistors. Since the key components such as transistor placement, detailed routing, and layout compaction must be flexible enough to support a wide variety of processes, this tool aims to generate layouts for real designs including support of design using new deep sub-micron process technologies and enable design rule changes to existing designs or processes. A description of the technology being used, which specifies the width/spacing rules for all mask layers, is included as a parameter for the layout generator. Two architectural styles are followed in the whole design, call-and-return and object-oriented.

2.1 An Example

I will begin with a simple example, a one-bit full adder (FA). As shown in Figure 2.1, it has three inputs and two outputs.

\[
\begin{align*}
\text{Sum} &= \text{A XOR B XOR } C_{\text{in}} \\
C_{\text{out}} &= \text{AB } + \text{BC}_{\text{in}} + C_{\text{in}} A
\end{align*}
\]

Figure 2.1 A One-bit Full Adder (FA)
A simple circuit implementation of a one-bit full adder is shown below (Figure 2.2) using two complex gates and two inverters:

\[
C_{out} = AB + C_{in}(A + B) \\
Sum = C_{out}(A + B + C_{in}) + ABC_{in}
\]

Accordingly, the designer needs to define a sum block and a carry block, for the generation of the sum and the carry respectively. Inside each block the geometrical object can be viewed as a stack of slices. A slice can be a wire (e.g., power, ground), a row of transistors, or a region for channel routing (between P and N transistor regions). What he needs to provide at this stage is no more than a description of the design in terms of different row slices, the relative positions of adjacent transistors, as well as all the electrical connections between any two terminals or between any terminal and any signal wire. Thus the user only needs to describe the electrical connections and the relative positions of all the cells and sub-cells in the circuit partitioning; the routing and physical placement are handled by my tool. In this case, he assembles the one-bit adder by simply mirroring the carry block and putting it below the sum block. The two blocks share the ground rail and fit nicely into a one-bit FA cell (Figure 2.3). The layout generator for the merged ground wire ensures that the width of the wire is sufficient for the sum of the currents on the two component wires.

It is then easy to build an n-bit ripple-carry adder from n one-bit full adders (Figure 2.4). This can be written as a for-loop with the creation of n instances of the pre-
defined OneBitFA cell (Figure 2.5). My tool performs the routing between cells, using the channels of the individual cells. The designer is thus freed from concerns of specifying additional routing regions. The tool determines the details of the layout geometry to implement the routing as well as the physical placement of transistors as a part of the automatic layout generation process.

Figure 2.3 User Specification of A One-bit FA
2.2 The User’s View

From the example mentioned above, we are able to understand the objects and operations needed in the API from the user’s point of view. We need a class to specify a circuit. A circuit can be placed to the left, right, bottom or top of any other circuit. A circuit can also be mirrored vertically or horizontally. Implementation of the circuit requires the description of different electrical terminals, which can be connected to each other.

I will describe the Signal interface first, as seen by the user, because it is simple and I will need it later to define a circuit cell. A Signal is something that can be
connected to other Signals. Therefore it has a connect method, which takes a Signal as a parameter and returns a Signal to make chaining of the connect operations easier. For example, to connect Signals u, v, and w together, one writes: u.connect(v).connect(w).

There are four implementations of the Signal interface (see Figure 2.6): DefaultSignal, Power, Ground, and Terminal.

Class DefaultSignal is the basic one that we can use to define any kind of signal in the circuit (e.g., the global signal, the clock signal, the input/output signal).

Class Power and Class Ground are both subclasses of Class Row (see Section 2.3); they need to implement Signal because they can be electrically connected to other signals or to transistor terminals. The fact that Java does not support multiple-inheritance also explains why I chose to make Signal an interface.

Class Terminal represents any kind of terminal (source/gate/drain) of a transistor. Similarly, a Terminal can be connected to Signals as well as other Terminals.

Now, let us take a look at the high-level description of a circuit. In the designer’s eyes, a circuit consists of smaller and more elementary parts; we call them cells. A circuit in this view is nothing but a combination of these cells. There are four mechanisms for arranging cells:

- Put one cell to the left of the other cell.
- Put one cell below the other cell.
- The left-and-right mirroring of a cell.
- The up-and-down mirroring of a cell.
Correspondingly there are four inner subclasses of the abstract class Circuit. Circuit is made an abstract class instead of an interface to support the definition of these subclasses. One more subclass of Circuit, CircuitRow, represents a circuit composed of only one row. Based on the above-mentioned combination methods, single-row cells are placed together to build the final circuit (Figure 2.7).

Inside the cell, each slice is defined as a subclass of abstract class Row. Both Power and Ground have parameters in their constructors for passing a user-defined signal width, which can be larger than the minimum width of metal in order to support both IR drop and electromigration requirements. Power takes one more parameter of a Signal in its constructor to distinguish between different electrical conventions. RoutingRow represents the channel routing region between P and N transistor regions. Global wires such as clocks are included in the routing region. This allows, for example, the same gate design without any manual changes in designs with different clocking methodologies. The merging of two cells next to each other with different global signals is also made much easier by expanding the routing regions of both.

![Diagram of Circuit subclasses](image)

**Figure 2.7 Subclasses of Circuit**

Classes RowNTran and RowPTran are used to represent N-type and P-type transistor rows. Like CircuitRow, either contains only one transistor in a row.
Accordingly I defined Class Transistor, and its subclasses NTransistor and PTransistor. A Transistor is composed of three Terminals representing its source, gate, and drain. It also supports the mirrorX operation. A user-defined transistor width is passed as a parameter to the constructor to support optimal transistor sizing. A row of an arbitrary number of transistors, therefore, is obtained by putting these rows next to each other with the Left method. The hierarchy of Class Row and its subclasses is shown in Figure 2.8.

Class Row is a lot like Class Circuit. It also has two inner subclasses Row.Left and Row.MirrorX to support the merging of two rows next to each other and the left-right mirroring of a row. Like Circuit, Row is made an abstract class instead of an interface to support the definition of the two subclasses. In retrospect, I could have made Row a subclass of Circuit, thus omitting the need for the definition of CircuitRow.

![Figure 2.8 Class Hierarchy of Row](image)

2.3 The Layout Generator's View

The operations of my tool's API that the user invokes construct data structures that describe the desired design for the layout generation methods. Thus the rest of the tool
also needs to obtain data from the API. First, it needs to find the set of electrical nodes and the terminals connected to each node. Class Node is designed for this purpose, to describe a list of electrically connected Signals. The Signal interface defines two methods, node and setNode, which are invoked primarily by router classes. Method node returns the Signal’s Node, and Method setNode sets its Node to a specific value. The node method provides a way to record all the connections in the netlist. The netlist is used by the router methods to determine which connections to make. The details of the netlist and the router, however, are hidden from the typical user.

For each circuit, the tool needs to get the rows of the circuit. And similarly, for each row, it needs to get the objects of the row. In either case, the target can be a circuit or a row that can be further expanded, or the target is a primitive circuit or row object. The getRows method returns an enumeration of the rows in the circuit cell in a bottom-first or top-first order, according to a parameter passed by the caller. Circuit is made an abstract class since every inner subclass implements this method in its own specific manner. The getTerminals method returns an enumeration of the terminals in a row, if there are any, in a left-first or right-first order (also passed as a parameter). Accordingly, Row is also an abstract class. There is one more abstract method, toGeom, in the definition of the class Row. It transforms the high-level description of a row to a geometrical representation, in terms of polygons and rectangles. I will address this transformation in Chapter 3.

In the implementation of the tool, I tried to make it technology independent. Three classes are designed for this purpose, Technology, Scmos and Layer. Technology has four member variables, which describe its associated layers, the minimum width of each layer, and the minimum spacing (or overhang) between every two. Method addLayer adds layers to the technology description. Methods addMinWidth, addMinOverhang, and addMinSep add different design rules to the particular technology file to ensure geometric design rule correctness in generated layouts. Technology also has a setTech method to specify the technology to be used in this tool. In this way designers can select the design rules for other processes by simply writing a technology class, and specifying the new technology through the calling of setTech. The current implementation provides one technology class, Scmos, which corresponds to the
MOSIS scalable CMOS (SCMOS) design rules. For the detailed implementation on this, please refer to Appendix A.

2.4 The Complete View

The user describes the circuit at a high level in terms of relative placements and electrical connections. Using the methods in the API, she then creates a Circuit object that can be passed to the MagicOutput.outputToMagic method to produce a layout file. This layout generation deals with many implementation details such as decomposing the user description into rows and flattened structures. To support sophisticated users, I include methods that are primarily for layout generation in the API. Table 2.1, Table 2.2, and Table 2.3 show the complete set of methods and structures of the key classes and interfaces in the API. They are given to make some comparison and for a better understanding of the architectural design. This is also how a sophisticated user might view the API and what she needs to know about before building any advanced application on it. In the tables below and throughout the following chapters, I use Sans Serif font to illustrate methods that typical users need to know, and Roman Italic font to illustrate those used primarily by the layout generator.

<table>
<thead>
<tr>
<th>Method Summary</th>
<th>Signal connect(Signal x)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Makes a connection with the specified signal.</td>
</tr>
<tr>
<td>Node node()</td>
<td>Returns the signal's electrical node.</td>
</tr>
<tr>
<td>void setNode(Node nd)</td>
<td>Sets the signal's electrical node to the specified value.</td>
</tr>
</tbody>
</table>

Table 2.1 Interface Signal

---

public abstract class Circuit

<table>
<thead>
<tr>
<th>Inner Class Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protected class</td>
</tr>
<tr>
<td>Circuit.Left</td>
</tr>
<tr>
<td>Protected class</td>
</tr>
<tr>
<td>Circuit.Below</td>
</tr>
<tr>
<td>Protected class</td>
</tr>
<tr>
<td>Circuit.MirrorX</td>
</tr>
<tr>
<td>Protected class</td>
</tr>
<tr>
<td>Circuit.MirrorY</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circuit</td>
</tr>
<tr>
<td>left(Circuit that)</td>
</tr>
<tr>
<td>Puts this circuit to the left of a specified circuit and returns the new circuit.</td>
</tr>
<tr>
<td>Circuit</td>
</tr>
<tr>
<td>below(Circuit that)</td>
</tr>
<tr>
<td>Puts this circuit below a specified circuit and returns the new circuit.</td>
</tr>
<tr>
<td>Circuit</td>
</tr>
<tr>
<td>mirrorX()</td>
</tr>
<tr>
<td>Returns the mirroring of the circuit in the X direction.</td>
</tr>
<tr>
<td>Circuit</td>
</tr>
<tr>
<td>mirrorY()</td>
</tr>
<tr>
<td>Returns the mirroring of the circuit in the Y direction.</td>
</tr>
<tr>
<td>abstract</td>
</tr>
<tr>
<td>Enumeration</td>
</tr>
<tr>
<td>getRows(boolean bottomFirst)</td>
</tr>
<tr>
<td>Returns an enumeration of the rows in the circuit in the specified order.</td>
</tr>
</tbody>
</table>

Table 2.2 Class Circuit

public abstract class Row

<table>
<thead>
<tr>
<th>Inner Class Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protected class</td>
</tr>
<tr>
<td>Row.Left</td>
</tr>
<tr>
<td>Protected class</td>
</tr>
<tr>
<td>Row.MirrorX</td>
</tr>
</tbody>
</table>

(Table 2.3, continued on next page)
(continuation of Table 2.3)

<table>
<thead>
<tr>
<th>Method Summary</th>
<th>Row</th>
<th>left(Row that)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Puts this row to the left of a specified row and returns the new row.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Row</td>
<td>mirrorX()</td>
<td>Returns the mirroring of the row in the X direction.</td>
</tr>
<tr>
<td>abstract Enumeration</td>
<td>getTerminals(boolean leftFirst)</td>
<td>Returns an enumeration of the terminals in the row in the specified order.</td>
</tr>
<tr>
<td>abstract GeomRow</td>
<td>toGeom(Technology t, boolean leftFirst)</td>
<td>Returns a geometrical representation of the row with the specified technology and order.</td>
</tr>
<tr>
<td>abstract Row</td>
<td>toTranrow(boolean leftFirst)</td>
<td>Returns the symbolic representation of the row if it is a transistor row, otherwise returns null.</td>
</tr>
</tbody>
</table>

Table 2.3 Class Row

For the complete code of the API, please refer to Appendix A. Furthermore, Class **FifoStage** in Appendix B shows how the API can be used to describe a simple target circuit.

### 2.5 Example Code for the Adder

I conclude this chapter by showing code fragments written with this API for the adder example. Figure 2.9 shows a code fragment of Class **SumBlock**. Class **CarryBlock** is similar. Based on them we write the code for Class **OneBitFA** as shown in Figure 2.10. The code for Class **Adder** is based on the code fragment in Figure 2.5.

---

3 Another implementation might be providing a default that does nothing and is overridden by transistor row subclasses.
public class SumBlock {
    private Signal A, B, Cin, Sum, Cout;
    private Circuit c;

    public Signal a() {
        return A;
    }

    public Circuit circuit() {
        return c;
    }

    public SumBlock(Signal _Vdd) {
        A = new DefaultSignal();
        ...
        Power Vdd = new Power(_Vdd, 6);
        Ground Gnd = new Ground(6);
        RoutingRow rr = new RoutingRow();

        Transistor[][] t = {
            {new PTransistor(8), //create 8 instances of PTransistor
                ...
            },
            {new NTransistor(4), //create 8 instances of NTransistor
                ...
            }
        };

        t[0][0].source().connect(t[0][1].drain()).connect(t[0][2].source()).connect(Vdd);
        t[0][0].drain().connect(t[0][1].source()).connect(t[0][2].drain());
        //more remaining connections
        ...

        Row prow = (new RowPTran((PTransistor)(t[0][0])))
            .left(new RowPTran((PTransistor)(t[0][1])))
            ...
            .left(new RowPTran((PTransistor)(t[0][7])));

        Row nrow = (new RowNTran((NTransistor)(t[1][0])))
            .left(new RowNTran((NTransistor)(t[1][1])))
            ...
            .left(new RowNTran((NTransistor)(t[1][7])));

        c = (new CircuitRow(Gnd))
            .below(new CircuitRow((NRow)(nrow.toTranrow())))
            .below(new CircuitRow(rr))
            .below(new CircuitRow((PRow)(prow.toTranrow())))
            .below(new CircuitRow(Vdd));
    }
}

Figure 2.9 Code Fragment of Class SumBlock
public class OneBitFA {
    private Signal A, B, Cin, Sum, Cout;
    private Circuit c;

    ...

    public OneBitFA(Signal _Vdd) {
        SumBlock sb = new SumBlock(_Vdd);
        CarryBlock cb = new CarryBlock(_Vdd);
        A = sb.a().connect(cb.a());
        B = sb.b().connect(cb.b());
        Cin = sb.cin().connect(cb.cin());
        Cout = sb.cout().connect(cb.cout());
        Sum = sb.sum();
        c = cb.circuit().mirrorY().below(sb.circuit());
    }
}

Figure 2.10 Code Fragment of Class OneBitFA
This chapter describes the interfaces to the primitive object generators and the Router package. They are used to convert the description of relative placements and electrical connections into the geometrical representation, in terms of polygons and rectangles. This description is primarily for software developers, who are extending or enhancing this tool. Normally the details of placement, primitive object generation and routing should be hidden from the typical user. However, an advanced user might use these features to build a sophisticated application above it, or to improve the current method.

This chapter describes optimizations applied in the generation of geometry, including the incorporation of cost functions in the constraint solver and a device merging technique that constructs optimal transistor chains with the minimum number of diffusion gaps.

Figure 3.1 shows the architecture of the code for object and constraint generation in the expansion from the user description into rows and flattened structures. It takes a Circuit object as input and produces a collection of constraints and rectangles as output. The constraint set is sent to the solver (see Chapter 4) together with an objective function to produce the optimal layout (optimal with respect to the objective function that currently permits minimization). Based on the estimated initial placement of components, the Router package computes the channel density it will need, introduces new constraints when necessary, and does the routing. Then the linear program is solved again with a new objective function to minimize the total wire length. Once a solution is found and sent to the Magic package (see Chapter 5), the designer gets the final generated layout.

The interfaces described in this chapter transform circuit description from the abstract rows, stacks of rows, and electrical connections used by the designer to collections of rectangles, constraints, and signals used by later phases of layout generation. This includes:

- **Expansion into rows and flattened structures**: Decomposing the user description of a circuit into rows, and in each row obtaining a collection of symbolic rectangles
with appropriate constraints (see Section 3.1). In general, power or ground rows consist of rectangles on metal 1, routing rows consist of geometry generated by routing models, and transistor rows have rectangles on seven different layers.

- **"Peep hole" optimization**: Merging of two primitive rows of the same type placed together, and merging inside a primitive row object. In particular, power or ground wires are made long enough to span both objects and wide enough to support the total current required. Routing regions are extended to include the global signals of both. Transistor rows are constructed with special care to ensure proper source/drain sharing (see Section 3.3).

![Figure 3.1 Object and Constraint Generation](image)

### 3.1 Interfaces
The geometrical circuit can be seen as a stack of different geometrical rows subject to certain additional constraints. Class \textit{GeomCircuit, GeomRow,} and \textit{GeomRowProto} provide the interfaces to this transformation and the Router package.

Figure 3.2 shows the member variables of Class \textit{GeomRow,} corresponding to what should be produced by \textit{Row.toGeom().}

A \textit{GeomRow} is a collection of rectangles, constraints, and signals. Each \textit{GeomRow} object is used in two phases. First, the \textit{toGeom} method of the \textit{Row} object that the designer has created creates one empty \textit{GeomRow} object and then adds rectangles, constraints, and signals to this object. The \textit{GeomRow} object enters its second phase when it is included into a \textit{GeomCircuit} object, from which it enumerates its rectangles, constraints and signals for later phases of layout generation. I enforce this two-phase model by introducing another class \textit{GeomRowProto}. The \textit{toGeom} method creates \textit{GeomRowProto} objects and uses their \textit{addRectangle, addConstraint,} and \textit{addSignal} methods to build its collections. These methods are summarized in Table 3.1. When finished, the \textit{toGeom} method creates a \textit{GeomRow} object from the \textit{GeomRowProto}. This allows routers, constraint solvers, and other aspects of layout generation to assume that \textit{GeomRow} objects are immutable, which fits naturally with the declaration style of constraint programming.
public class GeomRowProto

Constructor Summary

GeomRowProto(Technology t)

* Constructs a new GeomRowProto object with the specified technology.

GeomRowProto(GeomRow g1, GeomRow g2, Technology t, boolean leftFirst)

* Constructs a new GeomRowProto object merging the two GeomRow objects in the specified order and using the design rules of the specified technology.

Method Summary

<table>
<thead>
<tr>
<th>Type</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>addSignal(Signal s)</td>
<td>Adds the specified signal to the geometrical representation of this row.</td>
</tr>
<tr>
<td>void</td>
<td>addSignal(List s)</td>
<td>Adds the list of signals to the geometrical representation of this row.</td>
</tr>
<tr>
<td>void</td>
<td>addRectangle(Rectangle r, Layer layer)</td>
<td>Adds the specified rectangle to the specified layer.</td>
</tr>
<tr>
<td>void</td>
<td>addRectangle(Variable x0, Variable y0, Variable x1, Variable y1, Layer layer)</td>
<td>Adds a rectangle with the specified bottom-left and top-right coordinates to the specified layer.</td>
</tr>
<tr>
<td>void</td>
<td>addRectangle(Map m)</td>
<td>Adds all the rectangles in the specified map.</td>
</tr>
<tr>
<td>void</td>
<td>addConstraint(ConstraintSet c)</td>
<td>Adds the specified constraint set.</td>
</tr>
<tr>
<td>void</td>
<td>addConstraint(Variable v0, Variable v1, double d)</td>
<td>Adds constraint: v1 - v0 &gt; d.</td>
</tr>
<tr>
<td>void</td>
<td>addTop(Variable t, Layer layer)</td>
<td>Specifies t as the top variable in Layer layer.</td>
</tr>
</tbody>
</table>

(Table 3.1, continued on next page)
(continuation of Table 3.1)

<table>
<thead>
<tr>
<th>void</th>
<th>addBottom(Variable b, Layer layer)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Specifies b as the bottom variable in Layer layer.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>addLeft(Variable l, Layer layer)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Specifies l as the left variable in Layer layer.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>addRight(Variable r, Layer layer)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Specifies r as the right variable in Layer layer.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>nullify()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nullifies the GeomRowProto object and gets it garbage collected.</td>
</tr>
</tbody>
</table>

Table 3.1 Class GeomRowProto

GeomRowProto has another constructor that accepts two GeomRow objects. This is used when two rows are put next to each other. Both of their geometrical representations are derived first. Method addConstraint and the overridden version of methods addSignal and addRectangle merge the constraints, signals and rectangles into the combined geometrical row object. The left and right boundary limits are determined by their relative positions to each other indicated by the leftFirst parameter, and the bottom and top boundary limits are ensured to satisfy those of the two component objects. Horizontal spacing constraints are added as required, to keep the right boundary limit of one layer of the left component separated from the left boundary limit of the other layer of the right component by at least the minimum separation of the two layers in the design rule definition. The leftFirst parameter is again used here to tell which is the left component and which is the right.

Class GeomCircuit takes a Circuit object as a parameter in its constructor. By invoking the toGeomCircuit method, it turns each row into its geometrical representation, accumulates the rectangles and constraints involved, adds more vertical constraints between adjacent rows when necessary to keep them sufficiently separated, and adjusts power and ground rails to match the length of adjacent transistor rows. Vertically adjacent cells share the same global signal (e.g., power or ground rails) or
routing region when merged into a bigger circuit. Otherwise, some spacing constraints must be added to keep one cell sufficiently above the other to ensure that all design rules are satisfied. At this point we get the geometrical representation of the whole circuit. As in the combination of two rows, the bottom and top boundary limits are used here to add some vertical spacing constraints to keep one row sufficiently above the other in the stacking of the rows into the final circuit.

3.2 Cost Function

In order to achieve a layout as dense as manual layout after the component placement, I applied a cost metric that reduces the cell perimeter (width plus length) and minimizes diffusion in transistor placement. This is guaranteed not only in the primitive object generator (e.g., power, ground, transistor rows), but also in the merging of two rows next to each other and in the stacking of a number of row slices into the whole circuit.

3.3 Device Merging

Merging of transistor terminals can achieve significant savings in layout area. Inside a transistor row, there are three ways to connect a terminal to other terminals or signal wires in the circuit netlist:

- If one source/drain is not connected to its neighboring drain/source, then there is no sharing in between. Two contacts with sufficient horizontal separation should be drawn (Figure 3.3(a)).
- If one source/drain is connected to its neighboring drain/source and nothing else, then these two terminals share the diffusion region between them. No contact needs to be drawn (Figure 3.3(b)).
- If one source/drain is connected not only to its neighboring drain/source, but also to at least one other terminal or signal wire, then they share the contact between them. Only one contact needs to be drawn (Figure 3.3(c)).
To automate terminal sharing, I divided the drawing of a transistor's layout into the drawing of small fixed elementary parts, which I call bricks, assembled to constitute any desired transistor structure. Many of the possible MOS structures may be obtained using only two different bricks presented in Figure 3.3(d); I call them “G” and “C” (representing gate structure and contact) respectively. By this means the three cases in device merging can be simply represented by a string of symbols (e.g., “CGCCGC” for (a), “CGGC” for (b), “CGCGC” for (c)).

The geometrical generation of a transistor row is implemented in two steps. First, a symbolic representation is derived from the high-level description, based on the interconnections defined by the user. Method \texttt{Row.toTranrow} makes this transformation and creates an equivalent \texttt{NRow} or \texttt{PRow} object. \texttt{NRow} and \texttt{PRow} are transistor rows retrieved from a symbolic representation; they are defined to implement the source and drain sharing between transistor terminals. Second, the transistor row layout with device merging is drawn from this symbolic representation. This is done by the \texttt{toGeom} method of Class \texttt{NRow} and Class \texttt{PRow}. Special attention needs to be paid when the two merging terminals are different in width, in which case the method aligns them carefully and determines the width of the sharing contact or diffusion.
3.4 Routing

The next step after the placement of all the transistors and other structures in the layout is to connect the layout using wires on available layers according to the netlist. Routing has a profound impact on the quality of the final layout. Poor routing of nets includes unnecessary crossover of wires, circuitous routes, and redundant vias and contacts, all of which impair electrical performance and adversely affect yield and area [Guruswamy 97].

I considered a simple approach where three layers are available for routing: polysilicon, metal1, and metal2. Owing to the high resistance of diffusion, it is primarily used to interconnect adjacent transistors that share common signal nets. Similarly, due to the high resistance of polysilicon, polysilicon wires are limited to connect transistor gates. Channel routing algorithms are often used in layout automation systems.

Another graduate student in our department, Yang Lu, is implementing routers using the API provided by my prototype. In the pre-routing step, power/ground supply routing is performed by placing vertical taps to source/drain connections. Currently she has successfully finished the detailed routing in the single routing row between two transistor rows, and the generalization to the global routing which supports routing with more than two rows of transistors, and is still working on the compaction of the routing results to produce smaller designs.

Like the other layout generators described earlier, the router generates geometry in terms of constraints. Consider a design with a single routing row. The channel consists of several parallel routing tracks. The channel density provides a lower bound on the number of the tracks as follows. Sweep the channel from left to right. Every time we see the leftmost terminal of a net, we increment the density. Every time we see the rightmost terminal of a net, we decrement the density. The channel density is the highest density encountered in the sweep. Then we are able to do acceptable routing based on the initial transistor placement. We create a Vector of routing tracks. Each element of this Vector is either empty (i.e., null) or occupied by a particular net. Again we sweep from left to right. Each time we see the left end of a net, we allocate it to an empty track. If there is no empty track (routing can require more tracks than the channel density), we introduce
a new track, which also explains our use of a Vector (that can expand) instead of an array (that can't grow). Each time we see the right end of a net, we mark the corresponding track as empty. For any terminal of the net, we add a vertical wire (in polysilicon or metal2) from the terminal in the row of transistors to the track. To produce an efficient router, this simple algorithm can be augmented to support dog-legging (wires with jogs) and other optimizations. The router is implemented using my tool's interface to obtain the netlist, ordered lists of terminals, and the initial (unrouted) placement. The router uses the `addRectangle` and `addConstraint` methods to specify the geometry of its solution.
This chapter deals with the linear program interface. First, I describe constraint generation. Then, I present constraint solving.

A linear program consists of a set of linear constraints and an optimization vector, which is called "the cost function":

\[
\begin{align*}
\text{Find } x \text{ that minimizes } & c'x \\
\text{subject to } & A_{\text{eq}}x = b_{\text{eq}} \\
& A_{\text{ineq}}x \geq b_{\text{ineq}},
\end{align*}
\]

where \( c \) is the optimization vector; \( A_{\text{eq}} \) and \( A_{\text{ineq}} \) are matrices; and \( b_{\text{eq}} \) and \( b_{\text{ineq}} \) are vectors.

### 4.1 Constraint Generation

As stated in Chapter 3, the layout generator produces a circuit in terms of constraints and rectangles. A rectangle has variables defining its left, right, bottom and top limits. At the beginning it is symbolic (just in terms of variables, and values have not yet been bound to the variables). After constraint solving it is concrete (in terms of real values of the variables).

A key idea in the constraint solver is the distinction between a variable and a valuation. For example, given the inequalities: \( Y \geq X + 2 \) and \( X \geq Z + 3 \) (\( X, Y, Z \) are variables), these two constraints are satisfied by the valuation \( X = 3, Y = 5, Z = 0 \). They are also satisfied by the valuation \( X = 10, Y = 100, Z = 2 \). Typically, the sets of constraints that arise in our tool have an infinite set of satisfying valuations. We use an objective function to select an optimal valuation.
By separating variables from their values, additional constraints can be added as layout generation proceeds. For instance, this allows the router to add constraints to ensure that there is enough space to complete the routing successfully. In particular, we first solve the constraints prior to any routing. This provides initial placement hints for the router. The router adds constraints, and the final placement after routing may be much different with the initial placement prior to routing.

In this framework, each rectangle is defined by a layer and four symbolic variables: left, right, bottom and top. The layout generation methods introduce constraints for each of these variables. The constraint solver then finds a valuation that satisfies these constraints and optimizes the objective function. This valuation is used to output the rectangle with concrete coordinate values.

The `Variable` class implements `java.lang.Comparable` as `Variable` objects are used as keys for a map. Other than that, `Variable` objects only need to have a unique identity. This is because values come from the constraint solver later.

The matrices $A_{eq}$ and $A_{ineq}$ in (1) tend to be very sparse for this application and this might create linear programming problems with hundreds or thousands of variables, but only two variables in each constraint (i.e., each row in the matrix) (e.g., in the form of "$V_1 - V_0 \geq 4\), "$V_3 - V_2 = 2\)\)). We do not want to create a matrix that explicitly represents all of the zeros, because it may take much longer to initialize the matrix than to solve the linear programming problem. Therefore, either in the case of an inequality constraint representing $A'*X \geq b$, or in the case of an equality constraint representing $A'*X = b$, we can represent $A'$ as a collection of pairs: one element of the pair is the variable, and the other is the coefficient for that variable. For example, if we have the inequality: $V_{173} - V_{84} \geq 3$, we represent $A'$ as $\{(V_{173}, 1), (V_{84}, -1)\}$ and $b$ as 3. This has the added advantage that we can build the constraints without knowing the complete set of variables. Accordingly, Class `Coefficient` is constructed to represent such 2-tuples of the form (variable, coefficient), and Class `RowVector` takes an array of `Coefficient` objects as a parameter to represent the left-hand expression of a constraint. Class `Constraint` then takes a `RowVector` object and a double value in its constructor representing its left-hand expression and right-hand value respectively.
To represent the set of linear constraints in the linear program, Class `ConstraintSet` needs to be defined. Although the linear program solver wants the constraints in the form of matrices, we do not want to use matrices as we build up the constraints, to avoid excessive copying. Instead, we represent the syntactic structure of constraints in the same way as we do for building rows in the circuit in Chapter 2. A `ConstraintSet` object can be a primitive constraint (i.e., an equality constraint or an inequality constraint), or the conjunction of two constraint sets. The design of `ConstraintSet` is shown in Table 4.1. Like Class `Row`, it is made an abstract class instead of an interface to support the definition of three inner subclasses `ConstraintSet.Eq`, `ConstraintSet.InEq`, and `ConstraintSet.And`. This allows its inner classes to implement the `getEqualities` and `getInEqualities` methods specifically.

<table>
<thead>
<tr>
<th>Inner Class Summary</th>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>protected static class</td>
<td>static <code>ConstraintSet equality(RowVector a, double b)</code></td>
</tr>
<tr>
<td>protected static class</td>
<td>static <code>ConstraintSet simpleEq(Variable v1, Variable v2, double d)</code></td>
</tr>
<tr>
<td>protected class</td>
<td>static <code>ConstraintSet inequality(RowVector a, double b)</code></td>
</tr>
<tr>
<td></td>
<td>Static <code>ConstraintSet simpleIneq(Variable v1, Variable v2, double d)</code></td>
</tr>
</tbody>
</table>

`Returns an equality constraint of the form “aX = b”.`

`Returns an equality constraint of the form “v2 - v1 = d”.`

`Returns an inequality constraint of the form “aX ≥ b”.`

`Returns an inequality constraint of the form “v2 - v1 ≥ d”.`

(Table 4.1, continued on next page)
**Table 4.1 Class ConstraintSet**

<table>
<thead>
<tr>
<th>ConstraintSet</th>
<th>and(ConstraintSet that)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns a conjunction of this constraint set with the other specified constraint set.</td>
</tr>
<tr>
<td>abstract Enumeration</td>
<td>getEqualities()</td>
</tr>
<tr>
<td></td>
<td>Returns an enumeration of equality constraints in this constraint set.</td>
</tr>
<tr>
<td>abstract Enumeration</td>
<td>getInEqualities()</td>
</tr>
<tr>
<td></td>
<td>Returns an enumeration of inequality constraints in this constraint set.</td>
</tr>
</tbody>
</table>

### 4.2 Constraint Solving

To get values for these variables from the constraint solver, we need to define a valuation first. A valuation maps variables to values. This mapping allows us to solve the same set of constraints with different optimization vectors, or add more constraints and solve the augmented linear program, and get different values for the variables in both cases. Since *RowVector* and *Valuation* objects both associate *Variable* objects with values, a *RowVector* object containing all the variables in one problem instance is passed as a parameter in the constructor of Class *Valuation*. The methods *setValue* and *eval* are used to set a value to a variable or get the value of a variable in one valuation.

Based on this, *LinearProgram* is designed as an interface that has a *solve* method returning a *Valuation* object. It is made an interface to incorporate two different methods in the solving process.
Interface **LPfactory** and Class **Lpabo** provide interfaces to LPABO\(^4\). Interface **OPfactory** and Class **Opt** provide interfaces to a depth-first-traversal algorithm. By linking this tool to the former or the latter, we are able to get either an optimal solution or a fast, feasible and good solution for the problem. A comparison between them is made in Table 4.2.

<table>
<thead>
<tr>
<th>public interface LPfactory</th>
<th>public interface OPfactory</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Method Summary</strong></td>
<td></td>
</tr>
<tr>
<td>create(ConstraintSet cs,</td>
<td>create(ConstraintSet cs,</td>
</tr>
<tr>
<td>RowVector c1, RowVector</td>
<td>RowVector r)</td>
</tr>
<tr>
<td>c2, RowVector c3, List</td>
<td></td>
</tr>
<tr>
<td>LinearProgram</td>
<td>LinearProgram</td>
</tr>
<tr>
<td>list1, List list2)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public class Lpabo implements LinearProgram</th>
<th>public class Opt implements LinearProgram</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inner Class Summary</strong></td>
<td></td>
</tr>
<tr>
<td>static class Lpabo.Factory</td>
<td>static class Opt.Factory</td>
</tr>
<tr>
<td>Implements LPfactory</td>
<td>implements OPfactory</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Field Summary</strong></td>
<td></td>
</tr>
<tr>
<td>public final static LPfactory = factory</td>
<td>public final static OPfactory = factory</td>
</tr>
<tr>
<td>new Factory()</td>
<td>new Factory()</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Method Summary</strong></td>
<td></td>
</tr>
<tr>
<td>Valuation solve()</td>
<td>Valuation solve()</td>
</tr>
</tbody>
</table>

Table 4.2 Interfaces **LPfactory** and **OPfactory**, Classes **Lpabo** and **Opt**

**LPfactory** and **OPfactory** are two interfaces that both define a `create` method returning a **LinearProgram** object. They only differ with each other in the parameters

\(^4\) LPABO is the interior-point method based linear programming solver developed by Prof. Park, Soondal and the members of Operations Research Laboratory in Dept. of Industrial Engineering at Seoul National University.
required for the *create* method. *Lpabo* and *Opt* are both classes that implement the *LinearProgram* interface, thus they both have a *solve* method that returns a *Valuation* object. Since the two methods need different parameters sent to the solver, these two classes accept different parameters in their constructors. By defining a static field *factory* that creates an instance of a static inner class *Factory*, which, again implements the *LPfactory* or *OPfactory* interface, we are able to pass different parameters from our tool to the two different solving processes. A valuation is obtained by simply invoking *Lpabo.factory.create(..).solve()* or *Opt.factory.create(..).solve()*.

### 4.2.1 LPABO Solver

In my tool I used LPABO 5.72. It supports the MPS input format\(^5\). The first step of the *solve* method is to write the linear program to be solved into a file in the MPS format (e.g., “circuit.dat” in Appendix C). Then, by calling *Runtime.getRuntime().exec(...)* (in java.lang), we get the runtime object associated with the current Java application and execute the specified command and arguments in a separate process. After that we use the *Process.waitFor()* (in java.lang) invocation to block until the linear program solver is done. The output from the solver is saved in a file named “lpabo.out”. An example output file is also given in Appendix C. The last step of the *solve* method is to read the solution from this output file and return a *Valuation* object to the tool for further processing. The output from the LP solver is a possible optimal solution.

However, I met with one problem in the use of LPABO. Solutions for LPABO typically had variables with very large values, even when I had specified the upper and lower bounds for each variable. I had to carefully construct the cost function and include in it more variables than necessary to make sure I get a bounded solution from the solver. This resulted in an abnormally complex cost function in the problem definition, and led to ongoing difficulties. I suspect that the many degrees of freedom in the problem (including invariants under arbitrary shifts) led to the problem. To obtain meaningful solutions, I had to turn to an alternative solving method.

---

\(^5\) MPS input format was introduced by IBM. It is a way of creating inputs for linear and integer programs.
4.2.2 A Depth-first-traversal Algorithm

An alternative approach to find a fast, feasible and good solution for the problem is the use of a depth-first-traversal algorithm, based on the fact that all the constraints take the form of “V₁ - V₀ ≥ 4” and “V₃ - V₂ = 2”. The idea lies in viewing the constraint system as a directed constraint graph, with vertices representing variables and edges representing constraints between the connecting nodes. For example, constraint “V₁ - V₀ ≥ 4” establishes an edge pointing from node(V₁) to node(V₀) labelled with the value 4. Thus for each inequality constraint in the system, a left constraint is added to the variable with coefficient “1” (meaning the value of this variable should be more than that of the pointed variable by at least the edge weight), and at the same time a right constraint is added to the variable with coefficient “-1” (meaning the value of this variable should be less than that of the pointing variable by at least the edge weight). In this way it is easy to solve the constraint system by walking backward from unconstrained variables (i.e., variables with no outgoing edges), first for the left constraint and then for the right constraint. The left constraints produce lower bounds for the values of each variable, and the right constraints produce upper bounds. Averaging the two bounds produces reasonable values. Although this solving process doesn’t take into consideration the cost function in this linear program, it does provide a feasible and good solution in very small amount of time.

Methods addLeftConstraint, addRightConstraint, lo, and hi in Class Variable are defined accordingly to add left or right constraints to this variable and in the end to compute the lower and upper bound values.

Another problem may arise in this methodology, that is, the dealing with equality constraints in the system. In this case the equality constraint can be replaced with an inequality constraint associated with the appropriate substitution. For example, if we have constraints “V₃ - V₂ = 2” and “V₂ - V₀ ≥ 4”, we can replace the equality constraint with the inequality constraint “V₃ - V₀ ≥ 6”, and reconsider the modified system. Methods reduceEqualities, reduce, and reduceInequalities in Class Opt, based on the “union find” algorithm, ensure that all variables connected by equality relations map to the same variable with appropriate constants, and reconstruct the inequality constraint set.
accordingly. When the value of $V_3$ is known, we can get the value of $V_2$ by subtracting the value of $V_3$ by 2 ($V_3 - V_2 = 2$).

Classes *ConstraintEdge, Expr, Equality* and *Inequality* are some helper classes in this solving process.

I used the depth-first-traversal algorithm successfully in my prototype and obtained very satisfying results from the solver.

For details on the implementation of the LP interface, please refer to Appendix A and B.
This chapter talks about the Magic interface. It accepts the geometrical representation of a circuit and a valuation from the constraint solver, maps values to coordinates of rectangles on different layers, and writes to an output file in Magic format.

5.1 Interfaces

There are two classes in the Magic package, MagicOutput and MagicWriter. After showing their structures in Table 5.1 and 5.2, I will explain how they interact with each other to produce the final layout.

<table>
<thead>
<tr>
<th>public class MagicOutput</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method Summary</td>
</tr>
<tr>
<td>static void outputToMagic(Circuit circuit, String fileName, Technology tech, String LPsolver)</td>
</tr>
<tr>
<td>Turns the circuit into its geometrical representation using the specified technology, solves the system using the specified LP solver, and writes the result to a specified file.</td>
</tr>
<tr>
<td>static void writeToFile(String fdeName, GeomCircuit g, Technology t, Valuation v)</td>
</tr>
<tr>
<td>Maps values from a valuation to a geometrical representation of a circuit in a specified technology, and writes the result to a specified file.</td>
</tr>
<tr>
<td>static void writeLayer(MagicWriter writer, String layername, Enumeration rectangles, Valuation, v)</td>
</tr>
<tr>
<td>Maps values from a valuation to coordinates of rectangles on a specific layer, and writes this layer to the output file.</td>
</tr>
</tbody>
</table>

Table 5.1 Class MagicOutput
public class MagicWriter

Constructor Summary

MagicWriter(String filename, String technology)

Creates a buffered character-output stream with the specified file name, and writes the
technology information and timestamp information to the output file.

Method Summary

void beginLayer(String layerName)

Writes the beginning of a new layer into the Magic file.

void addRectangle(int x1, int y1, int x2, int y2)

Writes the four coordinates of a rectangle into the Magic file.

void beginLabels()

Starts writing labels into the Magic file.

void close()

Writes the end of the Magic file and closes the output stream.

Table 5.2 Class MagicWriter

An example Magic file ("FifoStage.mag") is included in Appendix D. By
invoking the static method MagicOutput.outputToMagic(...), we pass a String for the
file name, a Circuit object, a Technology definition and a String for the name of the LP
solver to the Magic interface, and get the geometrical representation of the circuit
together with a Valuation from the solver. The static method MagicOutput.writeToFile
is further invoked. A MagicWriter object is constructed with the creation of a buffered
character-output stream with the specified file name and the appending of the file head,
technology information, and timestamp information. Then for each layer defined in this
technology, it writes the layer name, maps values from the valuation to coordinates of
the rectangles on this layer, and writes every rectangle description into the file. After this
is done, it writes the end of the file and closes the output stream.
6.1 Methodology

I tested my tool using four examples. My goal was to produce layouts comparable to skilled manual design. In my experiments I let my tool generate the component placement for the target circuit, the result of which represented the initial placements and could be sent to the Router package to do the routing. At the routing stage, more constraints might be added when necessary to produce the final layout. So the goal I aim to achieve in the component placement stage is to generate an optimal circuit with the minimum area. By doing so more space and flexibility are reserved for the routing stage to help produce the most efficient final layout.

The first example is a single inverter with $6\lambda$ power and ground width and $8\lambda$ P-transistor and N-transistor width. The second is a row of seven such inverters placed together. The third is an inverter array composed of three such inverter rows stacked together, with sharing power/ground rails between adjacent rows. The fourth is in comparison with a manual layout “FifoStage.mag”, in which there are two power rails and one ground rail ($6\lambda$ wide each), eight P-transistors and ten N-transistors of width $4\lambda$, $6\lambda$, $8\lambda$, $10\lambda$, and $12\lambda$. Appropriate source/drain sharing is achieved based on the interconnections between these terminals.

The experiments were run on a Sun SPARC SUNW, Ultra-5_10 workstation using technology SCMOS SCN3M.35. The Java code was compiled using JDK-1.3. The time necessary for the layout to be generated depends on the dimensions and the complexity of the target circuit, but never exceeds a few seconds for our experiments.

6.2 Results
Figure 6.1, 6.2, and 6.3 show the layouts generated by my tool for the three inverter-related examples. From them we can see that these layouts realize the maximum area reduction and are as dense as manual design with respect to component placement. They are design rule correct (DRC) and satisfy the minimum separation between layers. What’s more, they are elegant in supporting user-defined signal width and transistor width, and dealing with the stacking of smaller cells into a bigger circuit with proper global wire sharing and width adjustment. Figure 6.4 shows the manual layout of “FifoStage.mag” and the layout generated by my tool for the same circuit (both before routing and after routing). Besides the above-mentioned good points, in this example we are able to see appropriate device merging between transistor terminals, which is exactly what the designer employs in her manual layout. Similarly this layout is as dense as possible in the component placement, which leaves much possibility for the routing stage to produce competent automatic layout. The layout after routing is not efficient enough, but it works now, and we are working on the compaction and more efficient routing algorithms. To further illustrate the density of these layouts, Table 6.1 shows the height, width, and area of these cells. All the layouts are as dense as possible after the component placement.
Figure 6.2 Example Layout for Inverter Row Placement

Figure 6.3 Example Layout for Inverter Array Placement
Figure 6.4 Comparison of Generated Layout with Manual Design for FifoStage
Table 6.1 Example Cell Density

The current tool is only a preliminary prototype. A full comparison with other layout methods will require a completed router and more experience with real designs. I expect that designers should be able to produce acceptable layouts with substantially less effort than manual design with some increase in area. I hope that an area penalty of 20-30% should be achievable while dramatically reducing the design time.

At the other end of the spectrum are tools that synthesize layouts from electrical netlists or behavioral descriptions. Typically, such tools make poor use of regular structures that often appear in designs, such as in data-paths, register files, or memory. In my tool, regular structures can be expressed using the structuring and control features of the host language, Java. Thus, it should be easy for the designer to describe good implementations of these structures. I do not foresee my tool offering any advantages for highly unstructured designs such as a random logic to implement an arbitrary finite state machine. Here, synthesis tools excel (because there is no good manual layout). Finding good ways to integrate synthesis techniques into my tool is a promising area for future research.
As covered in this thesis, automatic circuit layout generation involves the deployment and integration of many design automation techniques to produce high quality layouts from an initial transistor level or low level description. This thesis identifies opportunities for circuit optimization across design phases, and describes a fully automatic layout generation tool implemented in Java. The system is flexible enough to handle many process technologies and a wide variety of layout styles. The tool is fully automatic and provides several options to the user to customize the layout. The tool considers performance and generates dense, design rule correct layouts. Experimental results indicate that the area of generated layouts is competitive with manually designed cells for the circuits tried. Run times plus the times needed to write the description of the circuit are dramatically shorter than a human would require to manually create similar layouts, allowing a designer to try more topological variations to obtain an optimal design. Moreover, this tool allows us to target new technologies as they emerge, since only changes in the input technology file are needed.

As most other systems developed in the area of automatic layout generation, this tool also focuses on specific, fundamental problems related to transistor placement, routing, and compaction, and is described mainly to demonstrate its specific innovations. The more general direction for future work, and the promise of greater impact, is to look back on many ignored but practical problems essential to fully automatic layout generation, such as transistor folding\(^6\) to meet cell height requirements and minimum width, well and substrate tie insertion to meet tie coverage requirements in an area efficient way, input/output port placement on a routing grid for compatibility with place and route tools, circuit performance issues, automatic jog insertion to minimize area and

---

\(^6\) Transistor folding is the process of splitting a transistor into multiple transistors of smaller widths connected in parallel. Intelligent transistor folding is crucial to automatic layout generation because it automatically generates area-optimized layouts.

46
wire length to achieve hand-packed density, and more flexible architectures, such as transistor stacking. Considerable further research needs to be done.

The current tool is definitely a prototype, and in many ways, it is an experiment to demonstrate that we are able to generate layouts using very simple techniques. In particular, I provide the API that makes it easy for the designer to describe the design topology. The designer uses the structuring mechanisms of Object-Oriented programming to construct complicated designs using the simple API. Tedious details of physical placement and routing are handled by my tool.

Future research along this line should address how more issues of design and optimization can be incorporated into the simple framework. First, many optimization problems for layout synthesis can be expressed or approximated by linear or convex programming problems. These include, for example, retiming, transistor sizing and performance-driven routing. As many of these problems are NP-complete, formulations based on linear or convex programming are necessarily approximations. Our hope is that these convex formulations will lead to predictable results allowing the designer to find good layouts.

On the other hand, unlike most of the efforts made in the layout synthesis that are dedicated to specific improvements, our tool aims to provide a general high-level solution and free the designer from the need to get familiar with quite a few different optimization tools. We hope that in this way our tool will incorporate all the different optimization tools into a general framework, and by Object-Oriented design have an object view of the circuit and apply the structuring mechanisms. More efforts still need to be made to incorporate different optimization features into our prototype.


[Gamma 95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 
*Design Patterns: Elements of Reusable Object-Oriented Software.* 
Addison-Wesley, 1995.

Generator with Integrated Transistor Folding”. *Proceedings of 

Layout Generator for Two-Dimensional CMOS Cells”. In *the 34th 

[Guruswamy 97] Mohan Guruswamy, Robert L. Maziasz, Daniel Dulitz, Srilata 
Raman, Venkat Chiluvuri, Andrea Fernandez, and Larry G. Jones. 
“CELLERITY: A Fully Automatic Layout Synthesis System for 
Standard Cell Libraries”. In *the 34th ACM/IEEE Design 


[Lakos 97] John Lakos. “Technology Retargeting for IC Layout”. In *the 34th 

[Lea 94] Doug Lea. “Christopher Alexander: An Introduction for Object-
Oriented Designers”. In *Software Engineering Notes*, 1994.

Custom Cell Generation in Physical Synthesis”. In *the 34th 

Layout System”. *Proceedings of the 21st ACM/IEEE Design 

Compact Circuits with CMOS”. *IEEE Journal of Solid-State 


This appendix contains full source code for my layout generation tool. The Circuit package is presented first, followed by the Constraints package and the Magic package. Table A.1 lists the index of all the source files.

<table>
<thead>
<tr>
<th>package Circuit</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>abstract class Circuit</td>
<td>55</td>
</tr>
<tr>
<td>class CircuitInvalidException</td>
<td>56</td>
</tr>
<tr>
<td>class CircuitRow</td>
<td>57</td>
</tr>
<tr>
<td>class DefaultSignal</td>
<td>57</td>
</tr>
<tr>
<td>class GeomCircuit</td>
<td>58</td>
</tr>
<tr>
<td>class GeomRow</td>
<td>60</td>
</tr>
<tr>
<td>class GeomRowProto</td>
<td>61</td>
</tr>
<tr>
<td>class Ground</td>
<td>64</td>
</tr>
<tr>
<td>class Layer</td>
<td>65</td>
</tr>
<tr>
<td>class NRow</td>
<td>65</td>
</tr>
<tr>
<td>class NTransistor</td>
<td>67</td>
</tr>
<tr>
<td>class Node</td>
<td>67</td>
</tr>
<tr>
<td>class PRow</td>
<td>68</td>
</tr>
<tr>
<td>class PTransistor</td>
<td>69</td>
</tr>
<tr>
<td>class Power</td>
<td>70</td>
</tr>
<tr>
<td>class Rectangle</td>
<td>71</td>
</tr>
<tr>
<td>class RoutingRow</td>
<td>71</td>
</tr>
</tbody>
</table>

(Table A.1, continued on next page)
(continuation of Table A.1)

<table>
<thead>
<tr>
<th>Class/Interface</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>abstract class <code>Row</code></td>
<td>72</td>
</tr>
<tr>
<td>class <code>RowNTran</code></td>
<td>75</td>
</tr>
<tr>
<td>class <code>RowPTran</code></td>
<td>76</td>
</tr>
<tr>
<td>abstract class <code>RowTran</code></td>
<td>77</td>
</tr>
<tr>
<td>class <code>Scmos</code></td>
<td>77</td>
</tr>
<tr>
<td>interface <code>Signal</code></td>
<td>78</td>
</tr>
<tr>
<td>abstract class <code>Trow</code></td>
<td>79</td>
</tr>
<tr>
<td>class <code>Technology</code></td>
<td>79</td>
</tr>
<tr>
<td>class <code>Terminal</code></td>
<td>80</td>
</tr>
<tr>
<td>class <code>Transistor</code></td>
<td>81</td>
</tr>
<tr>
<td>package <code>Constraints</code></td>
<td></td>
</tr>
<tr>
<td>class <code>Coefficient</code></td>
<td>82</td>
</tr>
<tr>
<td>class <code>Constraint</code></td>
<td>82</td>
</tr>
<tr>
<td>class <code>ConstraintEdge</code></td>
<td>83</td>
</tr>
<tr>
<td>class <code>ConstraintGraph</code></td>
<td>83</td>
</tr>
<tr>
<td>abstract class <code>ConstraintSet</code></td>
<td>86</td>
</tr>
<tr>
<td>class <code>Equality</code></td>
<td>88</td>
</tr>
<tr>
<td>class <code>Expr</code></td>
<td>89</td>
</tr>
<tr>
<td>class <code>Inequality</code></td>
<td>89</td>
</tr>
<tr>
<td>interface <code>LPfactory</code></td>
<td>90</td>
</tr>
<tr>
<td>interface <code>LinearProgram</code></td>
<td>90</td>
</tr>
<tr>
<td>class <code>Lpabo</code></td>
<td>91</td>
</tr>
<tr>
<td>interface <code>OPfactory</code></td>
<td>93</td>
</tr>
<tr>
<td>class <code>Opt</code></td>
<td>94</td>
</tr>
<tr>
<td>class <code>RowVector</code></td>
<td>96</td>
</tr>
</tbody>
</table>

(Table A.1, continued on next page)
<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valuation</td>
<td>96</td>
</tr>
<tr>
<td>Variable</td>
<td>97</td>
</tr>
<tr>
<td>Magic</td>
<td></td>
</tr>
<tr>
<td>MagicOutput</td>
<td>99</td>
</tr>
<tr>
<td>MagicWriter</td>
<td>100</td>
</tr>
</tbody>
</table>

Table A.1 Index of Layout Generation Tool Source
package Circuit;
import java.util.*;

public abstract class Circuit {
    private boolean placed;

    public Circuit() {
        placed = false;
    }

    public boolean placed() {
        return placed;
    }

    public Circuit setPlaced() {
        if (placed)
            throw new CircuitInvalidException("Circuit Already Placed");
        placed = true;
        return this;
    }

    public abstract Enumeration getRows(boolean bottomFirst);
    public Enumeration getRows() {
        return getRows(true);
    }

    protected class Left extends Circuit {
        Circuit left, right;

        protected Left(Circuit l, Circuit r) {
            super();
            left = l.setPlaced();
            right = r.setPlaced();
        }

        public Enumeration getRows(boolean _bottomFirst) {
            final boolean bottomFirst = _bottomFirst;
            return (new Enumeration() {
                Enumeration e_left = left.getRows(bottomFirst);
                Enumeration e_right = right.getRows(bottomFirst);

                public boolean hasMoreElements() {
                    boolean hm_left = e_left.hasMoreElements();
                    boolean hm_right = e_right.hasMoreElements();
                    if (hm_left != hm_right)
                        throw new CircuitInvalidException("Mismatched Rows");
                    return hm_left;
                }

                public Object nextElement() {
                    Row l = (Row)e_left.nextElement();
                    Row r = (Row)e_right.nextElement();
                    return l.left(r);
                }
            });
        }
    }

    public Circuit left(Circuit that) {
        return new Left(this, that);
    }

    protected class Bottom extends Circuit {
        Circuit bottom, top;

        protected Bottom(Circuit b, Circuit t) {
            super();
            bottom = b.setPlaced();
            top = t.setPlaced();
        }

        public Enumeration getRows(boolean _bottomFirst) {
            final boolean bottomFirst = _bottomFirst;
            return (new Enumeration() {
                Enumeration e1 =
                (bottomFirst ? bottom : top).getRows(bottomFirst);
                Enumeration e2 =
                (bottomFirst ? top : bottom).getRows(bottomFirst);

                Row next = getNext();

                Row getNext() {
                    if (e1.hasMoreElements())
                        return ((Row)e1.nextElement());
                    else if (e2.hasMoreElements())
                        return ((Row)e2.nextElement());
                    else
                        return null;
                }

                public boolean hasMoreElements() {
                    return (next != null);
                }

                public Object nextElement() {
                    Row last = next;
                    for (next = getNext(); next != null) & last.canMerge(next);
                    if (next == null)
                        return last;
                    if (last.type() == Row.Ground)
                        ((Ground)(last)).setWidth(50((Ground)(last)).getWidth() +
                        ((Ground)(next)).getWidth());
                    if (last.type() == Row.Power)
                        ((Power)(last)).setWidth(50((Power)(last)).getWidth() +
                        ((Power)(next)).getWidth());
                    return last;
                }
            });
        }
    }
}
Aug 27 15:46 2003 Circuit.java Page 3

public Circuit below(Circuit that) {
    return new Below(this, that);
}

protected class MirrorX extends Circuit {
    Circuit c;

    protected MirrorX(Circuit _c) {
        c = _c;
    }

    public Enumeration getRows(boolean _bottomFirst) {
        final boolean bottomFirst = _bottomFirst;
        return (new Enumeration() {
            Enumeration e = c.getRows(bottomFirst);

            public boolean hasMoreElements() {
                return e.hasMoreElements();
            }

            public Object nextElement() {
                Row r = (Row)e.nextElement();
                return r.mirrorX();
            }
        });
    }
}

public Circuit mirrorX() {
    return new MirrorX(this);
}

protected class MirrorY extends Circuit {
    Circuit c;

    protected MirrorY(Circuit _c) {
        c = _c;
    }

    public Enumeration getRows(final boolean _bottomFirst) {
        return (c.getRows(!_bottomFirst));
    }

    public Circuit mirrorY() {
        return new MirrorY(this);
    }

    public Enumeration getNets() {
        throw new RuntimeException();
    }
}
Dec 19 11:13 2000 CircuitRow.java Page 1

```java
package Circuit;
import java.util.*;

class CircuitRow extends Circuit {
    private Row r;

    public CircuitRow(Row _r) {
        super();
        r = _r;
    }

    public Enumeration getRows(boolean bottomFirst) {
        return (new Enumeration() {
            boolean hasMore = true;
            public boolean hasMoreElements() {
                return hasMore;
            }
            public Object nextElement() {
                if (!hasMore)
                    throw new NoSuchElementException();
                hasMore = false;
                return r;
            }
        });
    }
}
```
package Circuit;
import java.util.*;
import Constraints.*;

public class GeomCircuit {
    private long num1 = 0;
    private long num2 = 0;
    private List vars1 = new LinkedList();
    private List freq1 = new LinkedList();
    private List vars2 = new LinkedList();
    private List freq2 = new LinkedList();
    private Circuit c;
    private Technology t;
    private ConstraintSet constraints;
    private Map rectangles;

    public GeomCircuit(Circuit _c, Technology _t) {
        c = _c;
        t = _t;
        constraints = null;
        rectangles = new HashMap();
    }

    public void toGeomCircuit() {
        toGeomCircuit(true, true);
    }

    public void toGeomCircuit(boolean bottomFirst, boolean leftFirst) {
        Enumeration e = c.getRows(bottomFirst);
        while (e.hasMoreElements()) {
            Row currow = (Row) e.nextElement();
            if (currow instanceof PROw)
                row = ((PRow) currow).toGeom(t, leftFirst);
            else if (currow instanceof NRw)
                row = ((NRow) currow).toGeom(t, leftFirst);
            else
                currow.toGeom(t, leftFirst);
        }
    }

    public static void main(String[] args) {
        // If one row has no rectangles on one layer, its top limit
        // on this layer == top limit on this layer of the row below it,
        // and its bottom limit on this layer == the bottom limit on this
        // layer of the row above it. As a result, for one layer of one row,
        // the bottom limit might be greater than the top limit.

        if (lastRow == null) {
            constraints = constraints.and(ConstraintSet);
            simpleNeg(lastRow.bottom().top(), lastRow.top().bottom(), 0); |
            if (row.getRectangles().getLayer() == null) |
                constraints = constraints.and(ConstraintSet);
            simpleNeg(lastRow.top().top(), lastRow.top().bottom(), 1); |
            while (12.hasMoreElements()) {
                Layer lay1 = (Layer)(12.nextElement());
                if (bottomFirst)
                    if (t.minSepProbe(lay1, lay2)) |
                        constraints = constraints.and(ConstraintSet);
                    simpleNeg(lastRow.top().top(), lastRow.top().bottom(), |
                        t.minSep(lay1, lay2)); |
                else |
                    if (t.minSepProbe(lay1, lay2)) |
                        constraints = constraints.and(ConstraintSet);
                    simpleNeg(lastRow.top().top(), lastRow.top().bottom(), |
                        t.minSep(lay1, lay2)); |
                } |
            } |
        }
    }
}
Layer metall = Technology.tech().getLayer("metall");
Layer ndc = Technology.tech().getLayer("ndc");
constraints = constraints.and(ConstraintSet.simpleIneq(
    (Variable) row.getRight().get(ndc)),
    (Variable) lastRow.getRight().get(metall)), 0);
constraints = constraints.and(ConstraintSet.simpleIneq(
    (Variable) row.getLeft().get(metall)),
    (Variable) row.getLeft().get(ndc)), 0));
}
if (currow.type() == Row.Ground) &&
    (prerow.type() == Row.PTranRow) {
Layer metall = Technology.tech().getLayer("metall");
Layer ndc = Technology.tech().getLayer("ndc");
constraints = constraints.and(ConstraintSet.simpleIneq(
    (Variable) lastRow.getRight().get(pdc),
    (Variable) row.getRight().get(metall)), 0);
constraints = constraints.and(ConstraintSet.simpleIneq(
    (Variable) row.getLeft().get(metall)),
    (Variable) lastRow.get().get(pdc)), 0));
}
if (currow.type() == Row.PTranRow) &&
    (prerow.type() == Row.Power) {
Layer metall = Technology.tech().getLayer("metall");
Layer pdc = Technology.tech().getLayer("pdc");
constraints = constraints.and(ConstraintSet.simpleIneq(
    (Variable) row.getRight().get(pdc),
    (Variable) lastRow.getRight().get(metall)), 0);
constraints = constraints.and(ConstraintSet.simpleIneq(
    (Variable) row.getLeft().get(metall)),
    (Variable) row.get().get(pdc)), 0));
}
lastRow = row;
prerow = currow;
firstrow = false;
Enumeration enum = t.layers();
while (enum.hasMoreElements()) {
    Layer layer = (Layer) enum.nextElement();
    addObjVars1(lastRow.top(), layer, 1);
}
public Map getRectangles() {
    return rectangles;
}

public ConstraintSet getConstraints() {
    return constraints;
}
public int getObjSize1() {
    return (int) num1;
}
public int getObjSize2() {
    return (int) num2;
}
public List getObjVars1() {
    return vars1;
}
public List getObjVars2() {
    return vars2;
}
public List getFreq1() {
    return freq1;
}
public List getFreq2() {
    return freq2;
}
public void addObjVars1(Variable v, int f) {
    if (!vars1.contains(v)) {
        vars1.add(v);
        num1++;
        int no = f;
        freq1.add(new Integer(no));
    } else {
        int index = vars1.indexOf(v);
        int oldvalue = ((Integer) freq1.get(index)).intValue();
        freq1.set(index, new Integer(oldvalue + f));
    }
}
public void addObjVars2(Variable v, int f) {
    if (!vars2.contains(v)) {
        vars2.add(v);
        num2++;
        int no = f;
        freq2.add(new Integer(no));
    } else {
        int index = vars2.indexOf(v);
        int oldvalue = ((Integer) freq2.get(index)).intValue();
        freq2.set(index, new Integer(oldvalue + f));
    }
}
public RowVector getObjFunction1() {
    Coefficient[] c0 = new Coefficient[{int][num1]};
    for(int i=0; i<num1; i++) {
        int no = {{integer}(freq1.get(i)).intValue();
        c0[i] = new Coefficient{{Variable} (vars1.get(i), {{-1.0}*no}};
    }
    RowVector r = new RowVector(c0);
    return r;
}

public RowVector getObjFunction2() {
    Coefficient[] c0 = new Coefficient[{int][num2]};
    for(int i=0; i<num2; i++) {
        int no = {{integer}(freq2.get(i)).intValue();
        c0[i] = new Coefficient{{Variable} (vars2.get(i), no}};
    }
    RowVector r = new RowVector(c0);
    return r;
}

public RowVector getAllVars() {
    long n = Variable.uniq;
    Coefficient[] c0 = new Coefficient[{int}n];
    for(int i=0; i<n; i++)
        c0[i] = new Coefficient{{Variable} (Variable.variables.elementAt(i), 1.0)};
    RowVector r = new RowVector(c0);
    return r;
}

package Circuit;
import Constraints,*;
import java.util.*;

class GeoRow {
    private List signals;
    private List objVars1, objVars2;
    private List freq1, freq2;
    private ConstraintSet constraints;
    private Map top, bottom, left, right;
    private Map rectangles;

class GeoRow(GeoRowProto gp) {
    signals = gp.getSignals();
    objVars1 = gp.getObjVars1();
    objVars2 = gp.getObjVars2();
    freq1 = gp.getFreq1();
    freq2 = gp.getFreq2();
    rectangles = gp.getRectangles();
    constraints = gp.getConstraints();
    top = gp.getTop();
    bottom = gp.getBottom();
    left = gp.getLeft();
    right = gp.getRight();
    gp.nullify();
}

class Enumeration rectangles(Layer layer) {
    return {Collections.enumeration({List}rectangles.get(layer))};
}

class Map getRectangles() {
    return rectangles;
}

class Enumeration signals() {
    return {Collections.enumeration(signals)};
}

class List getSignals() {
    return signals;
}

class List getObjVars1() {
    return objVars1;
}

class List getFreq1() {
    return freq1;
}

class List getObjVars2() {
    return objVars2;
}

class List getFreq2() {
public ConstraintSet constraints() {
    return constraints;
}

public Variable top(Layer layer) {
    return ((Variable) (top.get(layer)));
}

public Variable bottom(Layer layer) {
    return ((Variable) (bottom.get(layer)));
}

public Variable left(Layer layer) {
    return ((Variable) (left.get(layer)));
}

public Variable right(Layer layer) {
    return ((Variable) (right.get(layer)));
}

public Map getLeft() {
    return left;
}

public Map getRight() {
    return right;
}

package Circuit;
import Constraints.*;
import java.util.*;

public class GeomRowProto {
    private List signals;
    private List objVars1, objVars2;
    private List freq1, freq2;
    private ConstraintSet constraints;
    private Map top, bottom, left, right;
    private Map rectangles;
    private Technology t;

    public void nullify() {
        signals = null;
        constraints = null;
        top = null;
        bottom = null;
        left = null;
        right = null;
        rectangles = null;
        objVars1 = null;
        objVars2 = null;
        freq1 = null;
        freq2 = null;
    }

    public GeomRowProto(Technology t1) {
        this.t = t;
        signals = new LinkedList();
        objVars1 = new LinkedList();
        objVars2 = new LinkedList();
        freq1 = new LinkedList();
        freq2 = new LinkedList();
        rectangles = new HashMap();
        top = new HashMap();
        bottom = new HashMap();
        left = new HashMap();
        right = new HashMap();
        Enumeration e = t1.layers();
        constraints = null;
        while (e.hasMoreElements()) {
            Layer layer = (Layer) e.nextElement();
            Variable x = Variable.array2("x");
            Variable y = Variable.array2("y");
            top.put(layer, x);
            bottom.put(layer, y);
            left.put(layer, x);
            right.put(layer, x);
            addConstraint(x[0], x[1], 0.0);
        }

        public GeomRowProto(GeomRow g1, GeomRow g2, Technology t, boolean leftFirst) {
            this.t = t;
            signals = new LinkedList();
            constraints = null;
            top = null;
            bottom = null;
            left = null;
            right = null;
            rectangles = null;
        }
    }

    public void addRectangle(Rectangle r) {}
signals = new LinkedList<>(g1.getSignals());
objVars1 = new LinkedList<>();
objVars2 = new LinkedList<>();
freq1 = new LinkedList<>();
freq2 = new LinkedList<>();
addSignal(g2.getSignals());
rectangles = new HashMap<>(g1.getRectangles());
addRectangle(g2.getRectangles());
constraints = g1.constraints();
addConstraint(g2.constraints());
if (leftFirst) {
    left = new HashMap<>(g1.getLeft());
    right = new HashMap<>(g1.getRight());
}
else {
    left = new HashMap<>(g2.getLeft());
    right = new HashMap<>(g2.getRight());
}
top = new HashMap<>();
bottom = new HashMap<>();

public void addSignal(Signal s) {
    if (!signals.contains(s))
        signals.add(s);
}

public void addSignal(List s) {
    for (int i = 0; i < s.size(); i++) {
        if (((s.get(i)) instanceof Signal))
            throw new CircuitInvalidOperationException("Not A Signal");
        addSignal(((Signal)(s.get(i))));
    }
}

public void addObjectVars1(Variable v, int f) {
    if (objVars1.contains(v)) {
        objVars1.add(v);
        int num = f;
        freq1.add(new Integer(num));
    } else {
        int index = objVars1.indexOf(v);
        int oldValue = ((Integer)(freq1.get(index))).intValue();
        freq1.set(index, new Integer(oldValue + f));
    }
}

public void addObjectVars1(List v, List f) {
    for (int i = 0; i < v.size(); i++)
        addObjectVars1(((Variable)v.get(i)),
                        ((Integer)f.get(i))).intValue());
}

public void addObjectVars2(Variable v, int f) {
    if (objVars2.contains(v)) {
        objVars2.add(v);
        int num = f;
        freq2.add(new Integer(num));
    } else {
        int index = objVars2.indexOf(v);
        int oldValue = ((Integer)(freq2.get(index))).intValue();
        freq2.set(index, new Integer(oldValue + f));
    }
}

public void addObjectVar2(List v, List f) {
    for (int i = 0; i < v.size(); i++)
        addObjectVar2(((Variable)v.get(i)),
                      ((Integer)f.get(i))).intValue());
}

public List getSignals() {
    return signals;
}

public List getObjVars1() {
    return objVars1;
}

public List getFreq1() {
    return freq1;
}

public List getObjVars2() {
    return objVars2;
}

public List getFreq2() {
    return freq2;
}

public void addRectangle(Rectangle r, Layer layer) {
    if (rectangles.get(layer) == null)
        rectangles.put(layer, new LinkedList<>());
    ((List)(rectangles.get(layer))).add(r);
}

public void addVariable(Variable x0, Variable y0, Variable x1, Variable y1, Layer layer) {
    addRectangle(new Rectangle(x0, y0, x1, y1), layer);
}

public void addRectangle(Map m) {
    Enumeration e = m.elements();
    while (e.hasMoreElements()) {
        Layer layer = (Layer)(e.nextElement());
        if (m.get(layer) != null)
            for (int j = 0; j < ((List)(m.get(layer))).size(); j++) {
                if (rectangles.get(layer) == null)
                    rectangles.put(layer, new LinkedList<>());
            }
        }
    }
}

public void addVariable(Map m) {
    Enumeration e = m.elements();
    while (e.hasMoreElements()) {
        Variable var = (Variable)(e.nextElement());
        if (m.get(var) != null)
            for (int j = 0; j < ((List)(m.get(var))).size(); j++) {
                if (rectangles.get(var) == null)
                    rectangles.put(var, new LinkedList<>());
            }
        }
    }
}
public void addRight(Variable r, Layer layer) {
    right.put(layer, r);
}

public Map getRight() {
    return right;
}

public Variable right(Layer layer) {
    return ((Variable)right.get(layer));
}

public Technology t() {
    return t;
}
package Circuit;
import java.util.*;
import Constraints.*;

public class Ground extends Row implements Signal {
    private Node node;
    private double width;

    public Ground(double _width) {
        super();
        width = _width;
        setType(Row.Ground);
        node = new Node();
        node.addSignal(this);
    }

    public List stuff() {
        return null;
    }

    public double width() {
        return width;
    }

    public void setWidth(double w) {
        width = w;
    }

    public Signal connect(Signal x) {
        return DefaultSignal.connect(this, x);
    }

    public Node node() {
        return node;
    }

    public void setNode(Node nd) {
        node = nd;
    }

    public Enumeration getTerminals(boolean _leftFirst) {
        return null;
    }

    public GeomRow toGeom(Technology t, boolean leftFirst) {
        GeomRowProto gp = new GeomRowProto(t);
        gp.addSignal(this);
        Layer metal1 = Technology.tech().getLayer("metal1");
        Rectangle rect = new Rectangle(Variable(gp.getLeft().get(metal1)),
                                      Variable(gp.getBottom().get(metal1)),
                                      Variable(gp.getRight().get(metal1)),
                                      Variable(gp.getTop().get(metal1)));
        gp.addRectangle(rect, metal1);
        gp.addConstraint(rect.left(), rect.right(),
                         Technology.tech().minWidth(metal1));
        gp.addConstraint(ConstraintSet.simpleEq(rect.bottom(),
                                               rect.top(), width));
        gp.addConstraint(ConstraintSet.simpleIneq(rect.bottom(),
                                                  rect.top(), width));
        gp.addObjVars1(rect.left(), 1);
        gp.addObjVars1(rect.bottom(), 1);
        gp.addObjVars2(rect.right(), 1);
        return new GeomRow(gp);
    }

    public Row toTransRow(boolean leftFirst) {
        return null;
    }
}
package Circuit;

public class Layer {
    private String name;

    public Layer(String name) {
        this.name = name;
    }

    public String name() {
        return name;
    }
}

package Circuit;

import Constraints.*;
import java.util.*;

public class NRow extends TRow {
    public NRow(String s, List _list, List _stuff) {
        super(_s, _list, _stuff);
        setType(TRow.NTrans);
    }

    public GeomRow toGeom(Technology t, boolean leftFirst) {
        GeomRowProto gp = new GeomRowProto(t);
        Enumeration enum = getTerminals(leftFirst);
        boolean firstContact = true;
        boolean firstDate = true;
        Variable mergePoint = null;
        Variable ndc = null;
        Variable ndiff = null;
        Variable rdiff = null;
        Variable rpoly = null;
        Variable bndc = null;
        Variable temp = new Variable('y');
        Character ch = null;
        Variable x = null;
        Variable y = null;
        Layer ndc = Technology.tech1.getLayer("ndc");
        Layer ndiff = Technology.tech1.getLayer("ndiff");
        Layer ntran = Technology.tech1.getLayer("ntran");
        Layer poly = Technology.tech1.getLayer("poly");
        boolean lastC = false;
        ListIterator li = list.listIterator(0);
        while (enum.hasMoreElements()) {
            ch = (Character)(enum.nextElement());
            if (ch.charValue() == '\r') {
                x = Variable.array(2, 'x');
                y = Variable.array(2, 'y');
                double ndcWidth = (Double)(li.next());
                doubleValue();
                gp.addRectangle(x[0], y[0], x[1], y[1], ndc);
                gp.addConstraint(x[0], x[1], Technology.tech1.minWidth(ndc));
                gp.addConstraint(ConstraintSet.simpleEq(y[0], y[1], ndcWidth));
                if (firstContact) {
                    if (lastC) {
                        gp.addConstraint(ConstraintSet.simpleEq(mergePoint, x[0], 0.0));
                    } else {
                        gp.addConstraint(ConstraintSet.simpleEq(mergePoint, x[0], 4.0));
                    }
                    gp.addConstraint(ConstraintSet.simpleEq(bndc, y[0], 0.0));
                    gp.addConstraint(y[1], temp, 0.0);\n                } if (firstContact) {
                    gp.addBottom(y[0], ndc);
                    gp.addLeft(x[0], ndc);
                    gp.addRight(x[0], li);
                    gp.addBottomVars(y[0], li);
                    bndc = y[0];
                    gp.addConstraint(y[1], temp, 0.0);
                    firstContact = false;
                }
            }
        }
    }
}
Aug 13 02:31 2001 NRow.java Page 2

    )
    mergePoint = x[1];
    xndc = x[1];
    lastC = true;
    else if (ch.charValue() == 'O') {
        x = Variable.array(4, 'x');
        y = Variable.array(4, 'y');
        double ndcWidth = (Double)li.next().doubleValue();
        gp.addRectangle(x[0], y[1], x[1], y[2], ndiff);
        gp.addRectangle(x[1], y[1], x[2], y[2], ntran);
        gp.addRectangle(x[1], y[0], x[2], y[1], poly);
        gp.addRectangle(x[0], y[2], x[2], y[3], poly);
        gp.addConstraint(x[0], x[1], Technology.tech().minSep(ndc, ntran));
        gp.addConstraint(x[1], x[2], Technology.tech().minWidth(poly));
        gp.addConstraint(x[2], x[3], Technology.tech().minSep(ndc, ntran));
        gp.addConstraint(ConstraintSet.simpleEq(y[1], y[2], ndcWidth));
        gp.addConstraint(ConstraintSet.simpleEq(y[0], y[1], 0.01));
        gp.addConstraint(y[2], tmp, 0.01);
        mergePoint = x[3];
        ndiff = x[3];
        rpoly = x[2];
        if (firstGate) {
            gp.addBottom(y[1], ntran);
            gp.addBottom(y[0], ndiff);
            gp.addBottom(x[0], ndiff);
            gp.addBottom(y[0], poly);
            if (firstGate = false;)
                lastC = false;
        }
    }
    if (ch.charValue() == 'C') {
        gp.addRight(mergePoint, ndc);
        gp.addRight(ndiff, ndiff);
        gp.addRight(rpoly, poly);
        gp.addRight(rpoly, ntran);
    }
    else if (ch.charValue() == 'G') {
        gp.addRight(mergePoint, ndiff);
        gp.addRight(x[2], ntran);
        gp.addRight(x[4], poly);
        gp.addRight(nndc, ndc);
    }
    }
    gp.addObjVars2(mergePoint, 1);
    gp.addTop(tmp, ndc);
    gp.addTop(tmp, ndiff);
    gp.addTop(tmp, ntran);
package Circuit;

public class NTransistor extends Transistor {
    private int type;
    public NTransistor(double width) {
        super(width);
        type = Transistor.NTrans;
    }
}

package Circuit;
import java.util.*;

public class Node {
    private Vector signals;
    public Node() {
        signals = new Vector();
    }
    public int size() {
        return signals.size();
    }
    public Vector signals() {
        return signals;
    }
    public boolean contains(Signal s) {
        return signals.contains(s);
    }
    public Node addSignal(Signal s) {
        if (!signals.contains(s)) {
            signals.addElement(s);
        }
        return this;
    }
}
package Circuit;
import Constraints.*;
import java.util.*;

public class PRow extends TRow {
    public PRow(String _s, List _list, List _stuff) {
        super(_s, _list, _stuff);
        setType(new PTransRow);
    }

    public PTrans toPTrans(Technology t, boolean leftFirst) {
        Enumeration enum = getTerms(leftFirst);
        boolean firstContact = true;
        Variable mergePoint = null;
        Variable rpd = null;
        Variable rpdiff = null;
        Variable rpoly = null;
        Variable temp = new Variable('y');
        Character ch = null;
        Variable y = null;

        Layer pdc = Technology.tech().getLayer("pdc");
        Layer pdiff = Technology.tech().getLayer("pdiff");
        Layer ptran = Technology.tech().getLayer("ptran");
        Layer poly = Technology.tech().getLayer("poly");
        boolean lastC = false;
        ListIterator li = list.listIterator(0);

        while (enum.hasMoreElements()) {
            ch = (Character) enum.nextElement();
            if (ch.charValue() == 'C') {
                x = Variable.array(2, 'x');
                y = Variable.array(2, 'y');
                double pdcWidth = ((Double) (li.next())) . doubleValue();
                gp.addRectangle(x[0], y[0], x[1], y[1], pdc);
                gp.addConstraint(x[0], x[1], Technology.tech().minWidth(pdc));
                gp.addConstraint(ConstraintSet.simpleEq(y[0], y[1], pdcWidth));
                if (firstContact) {
                    if (!lastC)
                        gp.addConstraint(ConstraintSet.simpleEq(mergePoint, x[0], 0.0));
                    else
                        gp.addConstraint(ConstraintSet.simpleEq(mergePoint, x[0], 0.0));
                    gp.addConstraint(ConstraintSet.simpleEq(mergePoint, x[0], 0.0));
                    gp.addConstraint(ConstraintSet.simpleEq(bpdc, y[0], 0.0));
                    gp.addConstraint(y[1], tmp, 0.0);
                }
            }
        }

        return pTrans;
    }

    public PRow(String _s, List _list, List _stuff) {
        super(_s, _list, _stuff);
    }

    public void setType(PTransRow type) {
        setType = type;
    }

    public static void main(String[] args) {
        PRow row = new PRow("Row", new List(), new List());
        row.setRowType(new PTransRow);
    }
}
Aug 13 02:32 2001 PR:ow.java Page 3

import java.util.*;
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import java.lang.reflect.*;
import java.net.*;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.function.*;
import java.util.stream.*

public class Circuit {
    private int type;
    public PTransistor(double width) {
        super(width);
        type = Transistor.PTran;
    }

    public class Circuit {
        private int type;
        public PTransistor(double width) {
            super(width);
            type = Transistor.PTran;
        }
    }
}

package Circuit;

public class PTransistor extends Transistor {
    private int type;
    public PTransistor(double width) {
        super(width);
        type = Transistor.PTran;
    }

    public class Circuit {
        private int type;
        public PTransistor(double width) {
            super(width);
            type = Transistor.PTran;
        }
    }
}

package Circuit;

public class PTransistor extends Transistor {
    private int type;
    public PTransistor(double width) {
        super(width);
        type = Transistor.PTran;
    }

    public class Circuit {
        private int type;
        public PTransistor(double width) {
            super(width);
            type = Transistor.PTran;
        }
    }

package Circuit;
import java.util.*;
import Constraints.*;

public class Power extends Row implements Signal {
private Signal s;
private Node node;
private double width;

public Power(Signal _s, double _width) {
    super();
    s = _s;
    width = _width;
    setType(Row.Power);
    node = new Node();
    node.addSignal(this);
}

public List stuff() {
    return null;
}

public double width() {
    return width;
}

public void setWidth(double w) {
    width = w;
}

public Signal signal() {
    return s;
}

public Signal connect(Signal x) {
    return DefaultSignal.connect(s, x);
}

public Node node() {
    return node;
}

public void setNode(Node nd) {
    node = nd;
}

public Enumeration getTerminals(boolean _leftFirst) {
    return null;
}

public GeoMRow toGeoM(technology t, boolean leftFirst) {
GeoMRowProto gp = new GeoMRowProto(t);
gp.addSignal(s);
Layer metal1 = technology.tech().getLayer("metal");
Rectangle rect = new Rectangle((Variable)(gp.getLeft()).get(metal1)), (Variable)(gp.getBottom()).get(metal1));
gp.addRectangle(rect, metal1);
gp.addConstraint(rect.left(), rect.right(),
Technology.tech().minWidth(metal1));
/*
gp.addConstraint((ConstraintSet.simpleEq(rect.bottom(),
rect.top(), width));
*/
gp.addConstraint((ConstraintSet.simpleIneq(rect.bottom(),
rect.top(), width));
gp.addObjVar1(rect.left()[1], 1);
gp.addObjVar1(rect.bottom()[1], 1);
gp.addObjVar2(rect.right()[1], 1);
gp.addObjVar2(rect.top()[1], 1);
return (new GeoMRow(gp));
}

public Row toTranRow(boolean leftFirst) {
    return null;
}
package Circuit;
import Constraints.*;

public class Rectangle {
    private Variable left, right, bottom, top;
    public Rectangle(Variable l, Variable b, 
                      Variable r, Variable t) { 
        left = l;
        right = r;
        bottom = b;
        top = t;
    }
    public Variable left() {
        return left;
    }
    public Variable right() {
        return right;
    }
    public Variable bottom() {
        return bottom;
    }
    public Variable top() {
        return top;
    }
    public int left(Valuation v) {
        return (v.eval(left));
    }
    public int right(Valuation v) {
        return (v.eval(right));
    }
    public int bottom(Valuation v) {
        return (v.eval(bottom));
    }
    public int top(Valuation v) {
        return (v.eval(top));
    }
}

Jul 9 13:25 2001 RoutingRow.java Page 1

package Circuit;
import Constraints.*;
import java.util.*;

public class RoutingRow extends Row {
    private Vector globalSignals;
    public RoutingRow() {
        super();
        setType(Row.RouteRow);
        globalSignals = new Vector();
    }
    public List stuff() {
        return null;
    }
    public RoutingRow globalWire(Signal s) {
        if(!globalSignals.contains(s))
            globalSignals.addElement(s);
        return this;
    }
    public Enumeration getTerminals(boolean _leftFirst) {
        return null;
    }
    public GeomRow toGeom(Technology t, boolean _leftFirst) {
        GeomRowProto gp = new GeomRowProto(t);
        for(int i=0; i<globalSignals.size(); i++)
            gp.addSignal((Signal)globalSignals.elementAt(i));
        return (new GeomRow(gp));
    }
    public Row toTranrow(boolean _leftFirst) {
        return null;
    }
}

package Circuit;
import Constraints.*;
import java.util.*;

public abstract class Row {
    private boolean placed;
    private int type;
    protected List list;

    public static final int Power = 0;
    public static final int Ground = 1;
    public static final int PTranRow = 2;
    public static final int NTranRow = 3;
    public static final int RoutRow = 4;
    public static final int Unknown = 5;

    public Row() {
        placed = false;
        list = new LinkedList();
        type = Row.Unknown;
    }

    public List list() {
        return list;
    }

    public void setList(List l) {
        list = l;
    }

    public boolean placed() {
        return placed;
    }

    public Row setPlaced() throws CircuitInvalidException {
        if (placed)
            throw new CircuitInvalidException("Row Already Placed");
        placed = true;
        return this;
    }

    public int type() {
        return type;
    }

    public void setType(int type) {
        this.type = type;
    }

    public boolean canMerge(Row that) {
        return (this.type() == that.type());
    }

    public abstract Enumeration getTerminals(boolean first);
    public Enumeration getTerminals() {
        return (getTerminals(true));
    }
}

 Jul 9 13:36 2001 Row.java Page 1

 Jul 9 13:36 2001 Row.java Page 2

public abstract Collection toGeom(Technology t, boolean leftFirst);
public abstract Row toTransRow(boolean leftFirst);
public Row toTransRow() {
    return (toTransRow(true));
}

public abstract List stuff();
protected class Left extends Row {
    Row left, right;
    List stuff = new LinkedList();

    protected Left(Row l, Row r) {
        super();
        if (l.type() == r.type())
            throw new CircuitInvalidException("Mismatched Rows");
        this.setType(l.type());
        left = l;
        right = r;
    }

    public List stuff() {
        return stuff;
    }

    public Enumeration getTerminals(boolean _leftFirst) {
        final boolean leftFirst = _leftFirst;
        return
        (new Enumeration() {
            Enumeration e1 = (leftFirst ? left : right).getTerminals(leftFirst);
            Enumeration e2 = (leftFirst ? right : left).getTerminals(leftFirst);
            Object next = getNext();
            Object getNext() {
                if (e1.hasMoreElements())
                    return (e1.nextElement());
                else if (e2.hasMoreElements())
                    return (e2.nextElement());
                else
                    return null;
            }
            public boolean hasMoreElements() {
                return (next != null);
            }
            public Object nextElement() {
                Object last = next;
                next = getNext();
                return last;
            }
        });
    }
public class Row {

    private List<Row> left = new ArrayList<>();
    private List<Row> right = new ArrayList<>();

    public void add(Row row) {
        if (row != null) {
            if (right.isEmpty() || row.equals(right.get(right.size() - 1))) {
                right.add(row);
            } else if (left.isEmpty() || row.equals(left.get(left.size() - 1))) {
                left.add(row);
            } else {
                left.add(row);
            }
        }
    }

    public void remove() {
        if (left.isEmpty()) {
            return;
        }
        if (right.isEmpty()) {
            return;
        }
        if (left.contains(left.get(left.size() - 1))) {
            left.remove(left.size() - 1);
        }
        if (right.contains(right.get(right.size() - 1))) {
            right.remove(right.size() - 1);
        }
    }

    public List<Row> getLeft() {
        return left;
    }

    public List<Row> getRight() {
        return right;
    }

    // Other methods...
}

public class TableRow {

    private List<Row> left = new ArrayList<>();
    private List<Row> right = new ArrayList<>();

    public void add(Row row) {
        if (row != null) {
            if (right.isEmpty() || row.equals(right.get(right.size() - 1))) {
                right.add(row);
            } else if (left.isEmpty() || row.equals(left.get(left.size() - 1))) {
                left.add(row);
            } else {
                left.add(row);
            }
        }
    }

    public void remove() {
        if (left.isEmpty()) {
            return;
        }
        if (right.isEmpty()) {
            return;
        }
        if (left.contains(left.get(left.size() - 1))) {
            left.remove(left.size() - 1);
        }
        if (right.contains(right.get(right.size() - 1))) {
            right.remove(right.size() - 1);
        }
    }

    public List<Row> getLeft() {
        return left;
    }

    public List<Row> getRight() {
        return right;
    }

    // Other methods...
}
package Circuit;
import Constraints.*;
import java.util.*;

public class RowNTran extends RowTran {
    public RowNTran(NTranistor tr) {
        super((Transistor)(tr));
        setType(Row.NTranRow);
    }
    private final static RowInput[]巴菲 = new RowInput[32];

    public Row toNRow(boolean leftFirst) {
        List ls = new LinkedList();
        ls.add(0, new Double(tr.width()));
        ls.add(1, new Double(tr.width()));
        setList(ls);
        Enumeration enum = getTerminals(leftFirst);
        while (enum.hasMoreElements()) {
            gp.addSignal(term);
        }
        Layer ndc = Technology.tech().getLayer("ndc");
        Layer ndiff = Technology.tech().getLayer("ndiff");
        Layer ntran = Technology.tech().getLayer("ntran");
        Layer n1 = Technology.tech().getLayer("n1");
        Variable[] x = Variable.array(6, "x");
        Variable[] y = Variable.array(4, "y");
        gp.addRectangle(x[0], y[1], x[1], y[2], ndc);
        gp.addRectangle(x[1], y[2], x[2], y[2], n1);
        gp.addRectangle(x[2], y[2], x[3], y[3], ntran);
        gp.addRectangle(x[3], y[3], x[4], y[2], n1);
        gp.addRectangle(x[4], y[2], x[5], y[2], ndc);
        gp.addConstraint(x[0], x[1], Technology.tech().minWidth(ndc));
        gp.addConstraint(x[1], x[2], Technology.tech().minSep(ndc, ntran));
        gp.addConstraint(x[2], x[3], Technology.tech().minWidth(poly));
        gp.addConstraint(x[3], x[4], Technology.tech().minSep(n1, ntran));
        gp.addConstraint(x[4], x[5], Technology.tech().minWidth(ndc));
        gp.addConstraint(y[0], y[1], Technology.tech().minOverhang(poly, ntran));
        gp.addConstraint(y[1], y[2], Technology.tech().minWidth(n1));
        gp.addConstraint(y[2], y[3], Technology.tech().minOverhang(poly, ntran));
        gp.addTop(y[2], ndc);
        gp.addBottom(y[1], ndc);
        gp.addLeft(x[0], ndc);
        gp.addRight(x[5], ndc);
        gp.addTop(y[2], ntran);
        gp.addBottom(y[1], ntran);
        gp.addLeft(x[2], ntran);
        gp.addRight(x[3], ntran);
        gp.addTop(y[2], ndiff);
        gp.addBottom(y[1], ndiff);
}
Aug 12 14:12 2001 RowPTran.java Page 1

package Circuit;
import Constraints;
import java.util.*;

public class RowPTran extends RowTran {
    public RowPTran(PTranistor_tr) {
        super((Transactions)(t.r));
        setTypes(RowF.PTranRow);
    }

    public Geom toGeom(technology t, boolean leftFirst) {
        GeomRowProto gp = new GeomRowProto(t);
        Enumeration enum = getTerminals(leftFirst);
        while (enum.hasMoreElements()) {
            Terminal term = (Terminal) enum.nextElement();
            gp.addSignals(term);
        }

        Layer pdc = Technology.tech().getLayer("pdc");
        Layer pdiff = Technology.tech().getLayer("pdiff");
        Layer ptran = Technology.tech().getLayer("ptran");
        Layer poly = Technology.tech().getLayer("poly");

        Variable[] x = Variable.array(6, 'x');
        Variable[] y = Variable.array(4, 'y');
        gp.addRectangle(x[0], y[1], x[1], y[2], x[2], y[2], pdiff);
        gp.addRectangle(x[1], y[1], x[2], y[2], x[2], y[2], pdc);
        gp.addRectangle(x[2], y[0], x[3], y[1], y[2], pdc);
        gp.addRectangle(x[2], y[0], x[3], y[1], y[2], pdc);
        gp.addRectangle(x[3], y[1], x[4], y[2], pdiff);
        gp.addRectangle(x[4], y[1], x[5], y[2], pdc);
        gp.addConstraint(x[0], y[1], Technology.tech().minWidth(pdc));
        gp.addConstraint(x[1], y[2], Technology.tech().minSep(pdc, pdc));
        gp.addConstraint(x[2], y[3], Technology.tech().minWidth(pdc));
        gp.addConstraint(x[3], y[4], Technology.tech().minSep(pdc, pdc));
        gp.addConstraint(x[4], y[5], Technology.tech().minWidth(pdc));
        gp.addConstraint(y[0], y[1], Technology.tech().minOverhang(poly, pdc));
        // gp.addConstraint(ConstraintSet.simpleSym[y[1], y[2], tr.width()]);
        gp.addConstraint(ConstraintSet.simpleSym[y[1], y[2], tr.width()]);
        gp.addTop(y[2], pdc);
        gp.addBottom(y[1], pdc);
        gp.addTop(y[2], pdc);
        gp.addBottom(y[1], pdc);
    }

    public Row toRow() {
        return (new RowF.PTranRow());
    }
}

Aug 12 14:12 2001 RowPTran.java Page 2

gp.addLeft(x[1], pdiff);
gp.addRight(x[4], pdiff);
gp.addTop(y[3], poly);
gp.addBottom(y[0], poly);
gp.addLeft(x[2], poly);
gp.addRight(x[3], poly);
gp.addObjVars1(x[0], 1);
gp.addObjVars2(x[5], 1);
gp.addObjVars1(y[0], 1);
gp.addObjVars2(y[3], 1);

return (new PRow("COC", list, stuff));
}
Aug 12 14:28 2001 RowTrans.java Page 1

package Circuit;
import java.util.*;

public abstract class RowTrans extends Row {
    protected Transistor tr;
    protected List stuff = new LinkedList();

    public RowTrans(Transistor tr) {
        this.tr = tr;
    }

    public List stuff() {
        return stuff;
    }

    public Enumeration getProcessedStuff(boolean leftFirst) {
        return getTerminals(leftFirst);
    }

    public Enumeration getTerminals(boolean leftFirst) {
        Vector terminals = new Vector();
        if(leftFirst) {
            if(tr.isMirrored()) {
                terminals.addElement(tr.drain());
                terminals.addElement(tr.gate());
                terminals.addElement(tr.source());
            } else {
                terminals.addElement(tr.source());
                terminals.addElement(tr.gate());
                terminals.addElement(tr.drain());
            }
        } else {
            if(tr.isMirrored()) {
                terminals.addElement(tr.source());
                terminals.addElement(tr.gate());
                terminals.addElement(tr.drain());
            } else {
                terminals.addElement(tr.drain());
                terminals.addElement(tr.gate());
                terminals.addElement(tr.source());
            }
        }
        return (terminals.elements());
    }
}

Apr 29 23:14 2001 Scmos.java Page 1

package Circuit;
import java.util.*;

public class Scmos {
    private static Technology t = null;
    public static Technology tech() {
        if(t == null) {
            Layer poly = new Layer("poly");
            Layer ndiff = new Layer("ndiff");
            Layer pdiff = new Layer("pdiff");
            Layer pc = new Layer("pc");
            Layer ndc = new Layer("ndc");
            Layer mu2 = new Layer("mu2");
            Layer pdc = new Layer("pdc");
            Layer metal1 = new Layer("metal1");
            Layer metal2 = new Layer("metal2");
            Layer tran = new Layer("tran");
            t = new Technology();
            t.addLayer(poly);
            t.addLayer(ndiff);
            t.addLayer(pdiff);
            t.addLayer(pc);
            t.addLayer(ndc);
            t.addLayer(mu2);
            t.addLayer(pdc);
            t.addLayer(metal1);
            t.addLayer(metal2);
            t.addLayer(tran);
            t.addLayer(ptran);
            t.addMinWidth(poly, 2.0);
            t.addMinWidth(ndiff, 3.0);
            t.addMinWidth(pdiff, 3.0);
            t.addMinWidth(metal1, 3.0);
            t.addMinWidth(pdc, 3.0);
            t.addMinWidth(pdc, 4.0);
            t.addMinWidth(mu2, 4.0);
            t.addMinWidth(tran, 3.0);
            t.addMinWidth(ptran, 3.0);
            t.addMinSep(ndiff, ndiff, 3.0);
            t.addMinSep(ndiff, pdiff, 10.0);
            t.addMinSep(pdiff, pdiff, 3.0);
            t.addMinSep(pdiff, ndiff, 10.0);
            t.addMinSep(poly, poly, 2.0);
            t.addMinSep(poly, ndiff, 1.0);
            t.addMinSep(poly, pdiff, 1.0);
            t.addMinSep(ndiff, poly, 1.0);
            t.addMinSep(pdiff, poly, 1.0);
            t.addMinSep(poly, pc, 3.0);
            t.addMinSep(pc, ndiff, 1.0);
            t.addMinSep(pc, pdiff, 1.0);
            t.addMinSep(diff, pc, 1.0);
        }
        return t;
    }
}
Apr 29 23:34 2001 Scmos.java Page 2

t.addInSep(pdff, pc, 1.0);
t.addInSep(ndc, ndc, 4.0);
t.addInSep(pdff, pdff, 4.0);
t.addInSep(dff, dff, 4.0);
t.addInSep(pdff, pdff, 4.0);
t.addInSep(pdff, dff, 4.0);
t.addInSep(dff, pdff, 4.0);
t.addInSep(dff, dff, 4.0);
t.addInSep(pdff, pdff, 4.0);
t.addInSep(dff, dff, 4.0);
t.addInSep(pdff, pdff, 4.0);
t.addInSep(dff, dff, 4.0);
t.addInSep(pdff, pdff, 4.0);
t.addInSep(dff, dff, 4.0);
t.addInSep(pdff, pdff, 4.0);
t.addInSep(dff, dff, 4.0);
t.addInSep(pdff, pdff, 4.0);
t.addInSep(dff, dff, 4.0);
t.addInSep(pdff, pdff, 4.0);
t.addInSep(dff, dff, 4.0);
t.addInSep(pdff, pdff, 4.0);
t.addInSep(dff, dff, 4.0);
t.addInSep(pdff, pdff, 4.0);
t.addInSep(dff, dff, 4.0);
t.addInSep(pdff, pdff, 4.0);
t.addInSep(dff, dff, 4.0);
t.addInSep(pdff, pdff, 4.0);
t.addInSep(dff, dff, 4.0);
t.addInSep(pdff, pdff, 4.0);
t.addInSep(dff, dff, 4.0);
t.addInSep(pdff, pdff, 4.0);
t.addInSep(dff, dff, 4.0);
t.addInSep(pdff, pdff, 4.0);
t.addInSep(dff, dff, 4.0);
t.addInSep(pdff, pdff, 4.0);
t.addInSep(dff, dff, 4.0);
t.addInSep(pdff, pdff, 4.0);
t.addInSep(dff, dff, 4.0);
t.addInSep(pdff, pdff, 4.0);
t.addInSep(dff, dff, 4.0);
t.addInSep(pdff, pdff, 4.0);
t.addInSep(dff, dff, 4.0);
t.addInSep(pdff, pdff, 4.0);
t.addInSep(dff, dff, 4.0);
t.addInSep(pdff, pdff, 4.0);
t.addInSep(dff, dff, 4.0);
t.addInSep(pdff, pdff, 4.0);
t.addInSep(dff, dff, 4.0);
t.addInSep(pdff, pdff, 4.0);
t.addInSep(dff, dff, 4.0);
t.addInSep(pdff, pdff, 4.0);
t.addInSep(dff, dff, 4.0);
t.addInSep(pdff, pdff, 4.0);
t.addInSep(dff, dff, 4.0);

Dec 8 10:48 2000 Signal.java Page 1

package Circuit;

public interface Signal {
    public Signal connect(Signal x);
    public Node node();
    public void setNode(Node nd);
}

    return t;
}
Aug 12 13:54 2001  TRow.java Page 1

package Circuit;
import java.util.*;

public abstract class TRow extends Row {
    private String s;
    private List stuff;

    public TRow(String _s, List _list, List _stuff) {
        super();
        s = _s;
        list = _list;
        stuff = _stuff;
    }

    public List stuff() {
        return stuff;
    }

    public Enumeration getTerminals(boolean _leftFirst) {
        final boolean leftFirst = _leftFirst;
        return
        new Enumeration() {
            int i = 0;

            public boolean hasMoreElements() {
                return (i<s.length());
            }

            public Object nextElement() {
                if(!hasMoreElements())
                    throw new NoSuchElementException();
                else if(leftFirst)
                    return (new Character(s.charAt(i++)));
                else
                    return (new Character(s.charAt(s.length()) - (++i)));
            }
        };
    }

    public Row toTrash(boolean leftFirst) {
        return this;
    }
}

Apr 5 14:46 2001  Technology.java Page 1

package Circuit;
import java.util.*;

public class Technology {
    private List Layers;
    private Map minWidth;
    private Map minSep;
    private Map minOverhang;

    public Technology() {
        layers = new LinkedList();
        minWidth = new HashMap();
        minSep = new HashMap();
        minOverhang = new HashMap();
    }

    public void addLayer(Layer l1) {
        if(!layers.contains(l1))
            layers.add(l1);
    }

    public Layer getLayer(String name) {
        for(int i=0; i<layers.size(); i++)
            if(((Layer)layers.get(i)).getName().compareTo(name) == 0)
                return ((Layer)layers.get(i));
        return null;
    }

    public Enumeration layers() {
        return Collections.enumeration(layers);
    }

    public void addMinWidth(Layer l1, double d) {
        minWidth.put(l1, new Double(d));
    }

    public double minWidth(Layer l1) {
        return ((Double)minWidth.get(l1)).doubleValue();
    }

    public void addMinOverhang(Layer l1, double d) {
        Map minOverhangL1;
        if(!minOverhangL1.containsKey(l1)) {
            minOverhangL1 = new HashMap();
            minOverhang.put(l1, minOverhangL1);
        }
        else
            minOverhangL1 = (Map)minOverhang.get(l1);
        minOverhangL1.put(l2, new Double(d));
    }

    public double minOverhangFetch(Layer l1, Layer l2) {
        Map minOverhangL1 = (Map)minOverhang.get(l1);
        if(minOverhangL1 == null)
            return null;
        return ((Double)(minOverhangL1.get(l2)));
package Circuit;

public class Terminal implements Signal {
    private Transistor belongsTo;
    private int type;
    private Node node;
    public static final int Source = 0;
    public static final int Gate = 1;
    public static final int Drain = 2;
    public Terminal(Transistor t) {
        belongsTo = t;
        node = new Node();
        node.addSignal(this);
    }
    public Transistor belongsTo() {
        return belongsTo;
    }
    public void setType(int type) {
        this.type = type;
    }
    public int type() {
        return type;
    }
    public Signal connect(Signal x) {
        return DefaultSignal.connect(this, x);
    }
    public Node node() {
        return node;
    }
    public void setNode(Node nd) {
        node = nd;
    }
}
package Circuit;

class Transistor {
  public static Terminal source, gate, drain;
  private boolean isMirrored = false;
  private final int pmos = 0;
  private final int npmos = 1;

  public static Terminal source = new Terminal(this);
  public static Terminal gate = new Terminal(this);
  public static Terminal drain = new Terminal(this);

  public Terminal source() {
    return source;
  }

  public Terminal gate() {
    return gate;
  }

  public Terminal drain() {
    return drain;
  }

  public boolean isMirrored() {
    return isMirrored;
  }

  public Terminal thisTerminal() {
    return this;
  }

  public double width() {
    return this.width;
  }
}
package Constraints;

public class Coefficient {
    private Variable v;
    private double c;

    public Variable v() {
        return v;
    }

    public double c() {
        return c;
    }

    public Coefficient(Variable _v, double _c) {
        v = _v;
        c = _c;
    }
}

package Constraints;

public class Constraint {
    private RowVector a;
    private double b;

    public Constraint(RowVector _a, double _b) {
        a = _a;
        b = _b;
    }

    public RowVector getRowVector() {
        return a;
    }

    public double getDouble() {
        return b;
    }
}
package Constraints;

public class ConstraintEdge {
    private Variable v;
    private double d;
    public ConstraintEdge(Variable _v, double _d) {
        v = _v;
        d = _d;
    }
    public Variable v() {
        return v;
    }
    public double d() {
        return d;
    }
}

package Constraints;

import java.util.*;

/**
 * This class represents and solves a system of constraints of the form:
 *
 * 'x1 - x2 >= c'.
 *
 * The representation is as a directed graph. Each constraint variable 'x'
 * is given a corresponding node in the graph. The constraint 'x1 - x2
 * >= c' establishes an edge pointing from node(x1) to node(x2) labelled
 * with the value c.
 */

public class ConstraintGraph {
    private RowVector c;
    private Valuation sol;
    public ConstraintGraph(RowVector _c) {
        c = _c;
        sol = new Valuation(c);
    }

    /**
     * The class represents a node in the graph, which corresponds directly
     * to a Variable.
     */
    protected class Vertex {
        private Variable var;
        private Vector incomingEdges = new Vector();
        private Vector outgoingEdges = new Vector();
        private int dependCount = 0;
        protected Vertex(Variable var) {
            this.var = var;
        }
        protected Variable getVar() {
            return var;
        }
        protected Vector getOutedges() {
            return outgoingEdges;
        }
        protected Vector getInedges() {
            return incomingEdges;
        }
        protected int getDependcount() {
            return dependCount;
        }
        protected void setDependcount(int count) {
            dependCount = count;
        }
    }
}
protected class Edge {
    private Vertex source, dest;
    private int constTerm;
    private Constraint originalConstraint;

    protected Edge(Vertex source, Vertex dest, int constTerm, Constraint c) {
        this.originalConstraint = originalConstraint;
        this.source = source;
        this.dest = dest;
        this.constTerm = constTerm;
    }

    protected Vertex getSource() { return source; }

    protected Vertex getDest() { return dest; }

    protected int getConst() { return constTerm; }

    protected void setConst(int constTerm) { this.constTerm = constTerm; }

    protected Constraint getOriginalConstraint() { return originalConstraint; }
}

/// Maps from Variable to Vertex
private Hashtable vertices = new Hashtable();

/// Look up or create a vertex for the given variable
public Vertex getVertex(Variable var) {
    Vertex vertex = (Vertex)vertices.get(var);
    if(vertex == null) {
        vertex = new Vertex(var);
        vertices.put(var, vertex);
    }
    return vertex;
}

public void addEdge(Variable sourceVar, Variable destVar, int constTerm, Constraint c) {
    Vertex sourceNode = getVertex(sourceVar);
    Vertex destNode = getVertex(destVar);
    boolean alreadyFound = false;

    //Check to see if a '>=' relation already exists between these two Variables.
    Enumeration edges = sourceNode.getOutEdges().elements();
    while(edges.hasMoreElements() && !alreadyFound) {
        Edge edge = (Edge)edges.nextElement();
        if(edge.getDest().equals(destNode)) {
            edge.setConst(Math.max(edge.getConst(), constTerm));
            alreadyFound = true;
        }
    }

    //Add the new edge.
    if(!alreadyFound) {
        Edge e = new Edge(sourceNode, destNode, constTerm, c);
        sourceNode.getOutEdges().addElement(e);
        destNode.getInEdges().addElement(e);
    }
}

/// Returns all known relations involved with the given variable.
public Enumeration getConstraints(Variable var) {
    Vector constraints = new Vector();
    Vertex vertex = (Vertex)vertices.get(var);

    if(vertex != null) {
        Enumeration inEdges = vertex.getInEdges().elements();
        while(inEdges.hasMoreElements()) {
            Edge inEdge = (Edge)inEdges.nextElement();
            constraints.addElement(inEdge.getOriginalConstraint());
        }

        Enumeration outEdges = vertex.getOutEdges().elements();
        while(outEdges.hasMoreElements()) {
            Edge outEdge = (Edge)outEdges.nextElement();
            constraints.addElement(outEdge.getOriginalConstraint());
        }
    }

    return constraints.elements();
}
public void addVertex(Variable var) {
  getVertex(var);
}

public void deleteVertex(Variable var) {
  Vertex vertex = (Vertex)vertices.get(var);
  if (vertex == null)
    return;

  Enumeration inEdges = vertex.getInedges().elements();
  while (inEdges.hasMoreElements()) {
    Edge e = (Edge)inEdges.nextElement();
    e.getSource().getInedges().removeElement(e);
  }

  Enumeration outEdges = vertex.getOutedges().elements();
  while (outEdges.hasMoreElements()) {
    Edge e = (Edge)outEdges.nextElement();
    e.getDest().getInedges().removeElement(e);
  }

  vertices.remove(var);
}

public void solveVertex(Vertex v) {
  Enumeration edges = v.getOutedges().elements();
  int value = 0;

  while (edges.hasMoreElements()) {
    Edge dependEdge = (Edge)edges.nextElement();
    if (dependEdge.getDest().getVar().isSet() == false)
      value = Math.max(value, (sol.eval(dependEdge.
      getDest().getVar()) + dependEdge.getConst()));
  }

  sol.setValue((int) v.getVar().id(), value);
}

public void solveRestVertex(Vertex v) {
  Enumeration edges = v.getOutedges().elements();
  int value = 0;

  while (edges.hasMoreElements()) {
    Edge dependEdge = (Edge)edges.nextElement();
    value = Math.max(value, (sol.eval(dependEdge.
    getDest().getVar()) + dependEdge.getConst()));
  }

  sol.setValue((int) v.getVar().id(), value);
}
Mar 9 21:41 2001 ConstraintGraph.java Page 6

/**
 * Dumps the constraints in the graph.
 */
public String toString(Valuation solution) {
    StringBuffer buf = new StringBuffer();
    Enumeration allVertices = vertices.elements();
    while(allVertices.hasMoreElements()) {
        Vertex vertex = (Vertex)allVertices.nextElement();
        buf.append(dumpOutEdges(vertex, solution));
    }
    return buf.toString();
}

/**
 * Dumps the constraints (edges) pointing out of a given vertex.
 * Return the constraints for the given vertex.
 */
public String dumpOutEdges(Vertex v, Valuation solution) {
    StringBuffer buf = new StringBuffer();
    Enumeration outEdges = v.getOutEdges().elements();
    while(outEdges.hasMoreElements()) {
        Edge edge = (Edge)outEdges.nextElement();
        buf.append(edge.getSource().toString());
        buf.append(edge.getDest().toString());
        buf.append(edge.getVar().toString());
        buf.append(edge.getConst());
    }
    return buf.toString();
}

Feb 5 16:09 2001 ConstraintSet.java Page 1

package Constraints;
import java.util.*;

public abstract class ConstraintSet {
    protected static class Eq extends ConstraintSet {
        RowVector a;
        double b;
        protected Eq(RowVector _a, double _b) {
            a = _a;
            b = _b;
        }

        public Enumeration getEqualities() {
            return
        }
    }

    public Enumeration getInEqualities() {
        return null;
    }

    public static ConstraintSet equality(RowVector a, double b) {
        return new Eq(a, b);
    }

    public static ConstraintSet simpleEq(Variable v1, Variable v2,
            double d) {
        Coefficient[] c = new Coefficient[2];
        c[0] = new Coefficient(v1, -1.0);
        c[1] = new Coefficient(v2, 1.0);
        RowVector r = new RowVector(c);
        return equality(r, d);
    }
}

protected static class InEq extends ConstraintSet {
    RowVector a;
    double b;
    protected InEq(RowVector _a, double _b) {
        a = _a;
        b = _b;
    }
}
Feb 16 09 2001 ConstraintSet.java Page 2

public Enumeration getEqualities() {  
    return null;
}

public Enumeration getInEqualities() {  
    return (new Enumeration() {  
        boolean hasMore = true;
        public boolean hasMoreElements() {  
            return hasMore;
        }  
    });
}

public static ConstraintSet inequality(RowVector a, double b) {  
    return (new InEq(a, b));
}

public static ConstraintSet simpleIneq(Variable v1, Variable v2, double d) {  
    Coefficient[] c = new Coefficient[2];  
    c[0] = new Coefficient(v1, -1.0);  
    c[1] = new Coefficient(v2, 1.0);  
    RowVector r = new RowVector(c);  
    return inequality(r, d);
}

protected class And extends ConstraintSet {  
    ConstraintSet left, right;
    protected And(ConstraintSet l, ConstraintSet r) {  
        super();  
        left = l;  
        right = r;
    }

    public Enumeration getEqualities() {  
        final Enumeration el, e2;
        if(left instanceof Eq)  
            el = ((Eq)(left)).getEqualities();  
        else if(left instanceof InEq)  
            el = ((InEq)(left)).getEqualities();  
        else  
            el = left.getEqualities();

        if(right instanceof Eq)  
            e2 = ((Eq)(right)).getEqualities();  
        else if(right instanceof InEq)  
            e2 = ((InEq)(right)).getEqualities();  
        else  
            e2 = right.getEqualities();

        return (new Enumeration() {  
            Object next = getNext();  
            Object getNext() {  
                if((el != null) && (el.hasMoreElements()))  
                    return (el.nextElement());  
                else if((e2 != null) && (e2.hasMoreElements()))  
                    return (e2.nextElement());  
                else  
                    return null;
            }  
        });
    }

    public boolean hasMoreElements() {  
        return (next != null);
    }

    public Object nextElement() {  
        Object last = next;
        next = getNext();
        return last;
    }
}

public Enumeration getInEqualities() {  
    final Enumeration el, e2;
    if(left instanceof Eq)  
        el = ((Eq)(left)).getInEqualities();  
    else if(left instanceof InEq)  
        el = ((InEq)(left)).getInEqualities();  
    else  
        el = left.getInEqualities();

    if(right instanceof Eq)  
        e2 = ((Eq)(right)).getInEqualities();  
    else if(right instanceof InEq)  
        e2 = ((InEq)(right)).getInEqualities();  
    else  
        e2 = right.getInEqualities();

    return (new Enumeration() {  
        Object next = getNext();  
        Object getNext() {  
            if((el != null) && (el.hasMoreElements()))  
                return (el.nextElement());  
            else  
                return null;
        }  
    });
}
package Constraints;

public class Equality { // v1 -v2 = c
    private Variable v1;
    private Variable v2;
    private int c;

    public Variable v1() {
        return v1;
    }

    public Variable v2() {
        return v2;
    }

    public int c() {
        return c;
    }

    public Equality(Variable _v1, Variable _v2, int _c) {
        v1 = _v1;
        v2 = _v2;
        c = _c;
    }

    public boolean hasMoreElements() {
        return (e1.nextElement());
    } else if ((e2 != null) && (e2.hasMoreElements()))
        return (e2.nextElement());
    else
        return null;

    public Object nextElement() {
        return (next -= null);
    }

    public ConstraintSet and(ConstraintSet that) {
        return (new And(this, that));
    }

    public abstract Enumeration getEqualities();
    public abstract Enumeration getInEqualities();
}
package Constraints;

public class Expr {
    // v + c
    private Variable v;
    private int c;

    public Variable v() {
        return v;
    }

    public int c() {
        return c;
    }

    public Expr(Variable _v, int _c) {
        v = _v;
        c = _c;
    }
}

package Constraints;

public class Inequality {
    // v1 - v2 >= c
    private Variable v1;
    private Variable v2;
    private int c;

    public Variable v1() {
        return v1;
    }

    public Variable v2() {
        return v2;
    }

    public int c() {
        return c;
    }

    public Inequality(Variable _v1, Variable _v2, int _c) {
        v1 = _v1;
        v2 = _v2;
        c = _c;
    }
}
output += spacing.substring(0, 14-output.length());
output += "R" + i + 1;
i++;
output += spacing.substring(0, 33-output.length());
output += "1.0";
ps.println(output);
}
for(int j=0; j<Variable.uniq; j++) {
    String output = " V" + j + 3;
output += spacing.substring(0, 14-output.length());
output += "R" + i + 1;
i++;
output += spacing.substring(0, 33-output.length())-1);
output += "1.0";
ps.println(output);
}
p = ps.println("RHS");
L = 1;
e = cs.getEqualities();
while(!e.hasMoreElements()) {
Constraint con = (Constraint) e.nextElement();
String output = " RHS R" + i + 1;
String endString = Double.toString(con.getDouble());
output += spacing.substring(0, 36-output.length());
output += endString.length() + endString;
p = ps.println(output);
i++;
}
e = cs.getInEqualities();
while(!e.hasMoreElements()) {
Constraint con = (Constraint) e.nextElement();
String output = " RHS R" + i + 1;
String endString = Double.toString(con.getDouble());
output += spacing.substring(0, 36-output.length());
output += endString.length() + endString;
p = ps.println(output);
i++;
}
for(int j=0; j<Variable.uniq; j++) {
String output = " V" + j + 3;
output += spacing.substring(0, 36-output.length()+1);
output += "0.0";
p = ps.println(output);
i++;
}
for(int j=0; j<Variable.uniq; j++) {
String output = " RHS R" + i + 1;
output += spacing.substring(0, 36-output.length()-3)+"-1000.0";
p = ps.println(output);
i++;
}
p = ps.println("ENDDATA");
p = ps.println("ENDDATA");
ps.close();
for.close();
String cmdarray[] = "/Constraints/1pabo572.dat";
Apr 23 11:16 2001  Lpabo.java Page 5

    ) catch(IOException e1) {
      e1.printStackTrace();
      throw new RuntimeException("giving up");
    } catch(InterruptedException e2) {
      e2.printStackTrace();
      throw new RuntimeException("giving up");
    }
Jul 11 10:19 2001 Opt.java Page 1

package Constraints;
import java.io.*;
import java.util.*;
import Circuit.*;

class Opt implements LinearProgram {
    private ConstraintSet cs;
    private RowVector r;
    public Opt(ConstraintSet cs, RowVector r) {
        cs = cs;
        r = r;
    }
}

class Factory implements OPfactory {
    public LinearProgram create(ConstraintSet cs, RowVector r) {
        return new Opt(cs, r);
    }
}

public final static OPfactory factory = new Factory();

public Valuation solve() {
    Map m = reduceEqualities(eq);
    Collection ineq = reduceInequalities(ineq, m);
    Iterator it = ineq.iterator();
    // depth-first-traversal algorithm
    // if c is double, instead of writing if(c == 0)
    // write if(Math.abs(c) < epsilon)
    // public static final double epsilon = 1.0e-12;
    while(it.hasNext()) {
        Inequality ineq = (Inequality)(it.next());
        ineq.v1.addLeftConstraint(ineq.v2, ineq.c);
        ineq.v2.addRightConstraint(ineq.v1, ineq.c);
    }
    Valuation solution = new Valuation(r);
    it = ineq.iterator();
    while(it.hasNext()) {
        Inequality ineq = (Inequality)(it.next());
        if(ineq.v1.isSet()) {
            double low = ineq.v1.lo();
            double high = ineq.v1.hi();
            solution.setValue((int)(ineq.v1.id()), (int)(low+high)/2.0);
        } else {
            if(ineq.v2.isSet()) {
                double low = ineq.v2.lo();
                double high = ineq.v2.hi();
                solution.setValue((int)(ineq.v2.id()), (int)(low+high)/2.0);
            } else {
                Set s = m.keySet();
                boolean complete = false;
                while(!complete) {
                    complete = true;
                    Iterator sit = s.iterator();
                    while(sit.hasNext()) {
                        // do something
                    }
                }
            }
        }
    }
    return solution;
}

Jul 11 10:19 2001 Opt.java Page 2

Variable x = (Variable)(sit.next());
if(x.isSet() == false) {
    if(((Expr)m.get(x)).v1.isSet() == false) {
        complete = false;
        continue;
    }
    int xv = (int)solution.eval(((Expr)m.get(x)).v1) +
            ((Expr)m.get(x)).c1;
    solution.setValue(int(x.id()), xv);
}
return solution;

public Iterator eq() {
    List l = new LinkedList();
    Enumeration eq = cs.getEqualities();
    while(eq.hasMoreElements()) {
        Constraint con = (Constraint)(eq.nextElement());
        RowVector rv = con.getRowVector();
        if(rv.c(0).c() < 0) {
            l.add(new Equality(rv.c(1).v1, rv.c(0).v1), (int)con.getDouble());
        } else {
            l.add(new Equality(rv.c(0).v1, rv.c(1).v1), (int)con.getDouble());
        }
    }
    return l.iterator();
}

public Iterator ineq() {
    List l = new LinkedList();
    Enumeration ineq = cs.getInequalities();
    while(ineq.hasMoreElements()) {
        Constraint con = (Constraint)(ineq.nextElement());
        RowVector rv = con.getRowVector();
        if(rv.c(0).c() < 0) {
            l.add(new Inequality(rv.c(1).v1, rv.c(0).v1), (int)con.getDouble());
        } else {
            l.add(new Inequality(rv.c(0).v1, rv.c(1).v1), (int)con.getDouble());
        }
    }
    return l.iterator();
}

//reduceEqualities() and reduce() are based on the 'union find'
//algorithm described in the C++ text.
public Map reduceEqualities(Iterator eq) {
    Map m = new HashMap();
    while(eq.hasNext()) {
        Equality c = (Equality)(eq.next());
        Expr el1 = reduce(c.v1[1], m);
        Expr el2 = reduce(c.v2[1], m);
        switch(c) {
                c = E1.C1 || E2.C2;
                break;
                c = E1.C1 || E2.C2;
                break;
        }
    }
    return m;
}

```java
if (e1.v().compareTo(e2.v()) == 0) { // same variables
  // there is a cycle in eq, make sure that it is consistent.
  if (c != 0) throw new RuntimeException("inconsistent equalities");
} else
  m.put(e1.v(), new Expr(e2.v()), c));
return m;
}

// reduce|v, m| gives an expression that is equal to v.
// In combination with reduceEqualities, this ensures that all variables
// that are connected by equality relations map to the same variable
// with appropriate constants.
public Expr reduce(Variable v, Map m) {
  Expr e = (Expr)(m.get(v)); // v = e.v + e.c
  if (e == null)
    return new Expr(v, 0); // no map for v:
  v = v + 0;
  else {
    Expr e2 = reduce(e.v(), m); // v = e2.v + e2.c
    if (e2.v() == e.v()) { // this means e2.v() is not a key in m
      // therefore e2 is e.v + 0, no further reductions.
      v = e.v + e.c + 0;
      return e;
    }
    else {
      Expr e3 = new Expr(e2.v(), (e.c() + e2.c()));
      // having further simplified v, we'll put the reduced form
      // into the hashtable to avoid repeating the computation
      // later.
      m.put(v, e3);
      return e3;
    }
  }
}

// reduceInequalities(ineq, m) replaces every variable in an
// inequality of ineq according to the equalities in m.
// obtained using the reduceEqualities method above.
public Collection reduceInequalities(Iterator ineq, Map m) {
  LinkedList list = new LinkedList();
  while (ineq.hasNext()) {
    Inequality ie = (Inequality)(ineq.next());
    Expr e1 = reduce(ie.v1(), m);
    Expr e2 = reduce(ie.v2(), m);
    // as in reduceEqualities: e1.v = e2.v >= ie.c + e2.c = e1.c
    int c = ie.c() + e2.c() - e1.c();
    if (e1.v().compareTo(e2.v()) == 0) { // same variables
      // there is a cycle in ineq, make sure that it is consistent.
      if (c > 0) throw new RuntimeException("inconsistent inequalities");
    }
    else
      list.add(new Inequality(e1.v(), e2.v(), c));
  }
  return list;
}
```
package Constraints;

public class RowVector {
    private Coefficient[] c;
    public Coefficient cli(int i) {
        return c[i];
    }
    public int length() {
        return c.length;
    }
    public RowVector(Coefficient[] c) {
        c = _c;
    }
}

Mar 6 09:57 2001 Valuation.java Page 1

package Constraints;
import java.util.*;

public class Valuation {
    private Map m;
    private RowVector r;
    public Valuation(RowVector r) {
        r = _r;
        m = new HashMap();
        for(int i=0; i<r.length(); i++) {
            Double temp = new Double(r.c[i].c[i]);
            m.put(r.c[i].v(), new Integer(temp.intValue()));
        }
    }
    public RowVector r() {
        return r;
    }
    public void setValue(int i, int value) {
        m.put(r.c[i].v(), new Integer(value));
        if(r.c[i].v().isSet() == false)
            r.c[i].v().set();
    }
    public int eval(Variable v) {
        Integer d = (Integer)m.get(v);
        if(d == null)
            throw new NoSuchElementException(v.toString());
        return (d.intValue());
    }
}
package Constraints;
import java.lang.*;
import java.util.*;

public class Variable implements Comparable {
    public static long uniq = 0;
    public static Vector variables = new Vector();
    private long id;
    private String name;
    private boolean isset = false;
    private List left, right;
    private char key;
    private double lo, hi;
    private boolean loSet, hiSet, busy;
    private static final String keys = "xy";

    public long id() {
        return id;
    }

    public Variable(String a, char k) {
        id = uniq++;
        name = a;
        variables.add(this);
        if(keys.indexOf(a) < 0)
            throw new RuntimeException("variable error");
        key = k;
        loSet = false;
        hiSet = false;
        busy = false;
        left = new LinkedList();
        right = new LinkedList();
    }

    public Variable(char k) {
        this(null, k);
        name = "V"+id;
    }

    public int compareTo(Object obj) {
        Variable that = (Variable)obj;
        if(this.id() < that.id())
            return -1;
        else if(this.id() == that.id())
            return 0;
        else
            return 1;
    }

    public String toString() {
        return name;
    }

    public static Variable[] array(int n, char k) {
        Variable[] v = new Variable[n];
        for(int i=0; i<n; i++)
            v[i] = new Variable(k);
        return v;
    }

    public void set() {
        isset = true;
    }

    public boolean isset() {
        return isset;
    }

    public void addLeftConstraint(Variable l, double c) {
        if(loSet)
            throw new RuntimeException("too late to add left constraint to variable");
        left.add(new ConstraintEdge(l, c));
    }

    public void addRightConstraint(Variable l, double c) {
        if(hiSet)
            throw new RuntimeException("too late to add right constraint to variable");
        right.add(new ConstraintEdge(l, c));
    }

    // return a lower bound for the value of this variable
    public double lo() {
        if(!loSet)
            throw new RuntimeException("cycle of constraints");
        busy = true;
        ListIterator it = left.listIterator(0);
        double v = 0.0;
        while(it.hasNext()) {
            ConstraintEdge e = (ConstraintEdge)it.next();
            double u = e.v().lo() + e.d();
            if(u > v) v = u;
        }
        lo = v;
        loSet = true;
        busy = false;
    }

    // return an upper bound for the value of this variable
    public double hi() {
        if(!hiSet)
            throw new RuntimeException("cycle of constraints");
        busy = true;
        ListIterator it = right.listIterator(0);
        double v = 0.0;
        while(it.hasNext()) {
            ConstraintEdge e = (ConstraintEdge)it.next();
            double u = e.v().hi() - e.d();
            if(v > u) v = u;
        }
    }
}
package Magic;
import java.io.*;
import java.util.*;
import Constraints.*;

public class MagicOutput {
    private static final String TECHNOLOGY = "scmos";

    public static void writeToFile(String fileName, 
        GeomCircuit g, Technology t, Valuation v) throws IOException {
        MagicWriter writer = new MagicWriter(fileName, TECHNOLOGY);
        Enumeration e = t.layers();
        while (e.hasMoreElements()) {
            Layer l = (Layer)e.nextElement();
            if (l.getRectangles() != null) {
                writer.layerWriter(l.name(), Collections.enumeration(
                    (Collection)l.getRectangles().get(l)), v);
            }
            writer.close();
        }
    }

    private static void writeLayer(MagicWriter writer, String layerName, 
        Enumeration rectangles, Valuation v) throws IOException {
        if (rectangles.hasMoreElements()) {
            writer.beginLayer(layerName);
            while (rectangles.hasMoreElements()) {
                Rectangle rect = (Rectangle)rectangles.nextElement();
                writer.addRectangle(rect.leftV, rect.bottomV, 
                    rect.rightV, rect.topV);
            }
        }
    }

    public static void outputToMagicCircuit(String fileName, 
        Technology t, String LpSolver) {
        Geometry geomCircuit = geomCircuit.toGeomCircuit();
        LinearProgram lp;
        if (LpSolver == "lpab")
            lp = lpab.factory.create(geomCircuit.getConstraints(), 
                geomCircuit.getObjectFunction(), 
                geomCircuit.getObjFunction2(), 
                geomCircuit.getVars(), 
                geomCircuit.getPrel1(), 
                geomCircuit.getPrel2());
        else
            lp = opt.factory.create(geomCircuit.getConstraints(), 
                geomCircuit.getObjFunction(), 
                geomCircuit.getObjFunction2(), 
                geomCircuit.getVars());
        Valuation sol = lp.solve();
        try {
            writeToFile(fileName, geomCircuit, t, sol);
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
package Magic;
import java.io.*;

public class MagicWriter {
    private BufferedWriter out;

    MagicWriter(String fileName, String technology) throws IOException {
        if (!fileName.endsWith(".mag"))
            fileName = fileName + ".mag";
        out = new BufferedWriter(new FileWriter(fileName));
        out.write("magic");
        out.newLine();
        if (technology != null) {
            out.write("tech " + technology);
            out.newLine();
        }
        out.write("timestamp " + System.currentTimeMillis()/1000L);
        out.newLine();
    }

    void beginLayer(String layerName) throws IOException {
        out.write("<< " + layerName + " >>");
        out.newLine();
    }

    void addRectangle(int x1, int y1, int x2, int y2) throws IOException {
        out.write("rect " + x1 + " " + y1 + " " + x2 + " " + y2);
        out.newLine();
    }

    void beginLabels() throws IOException {
        out.write("<< labels >>");
        out.newLine();
    }

    void close() throws IOException {
        out.write("<< end >>");
        out.newLine();
        out.close();
    }
}
This appendix includes the four example source tested in Chapter 6. In a similar way, the designer can write a description of a target circuit using the API provided in Appendix A. Table B.1 lists the index of all the example files.

<table>
<thead>
<tr>
<th>class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>class <em>FifoStage</em></td>
<td>102</td>
</tr>
<tr>
<td>class <em>Inv</em></td>
<td>103</td>
</tr>
<tr>
<td>class <em>Invs</em></td>
<td>104</td>
</tr>
<tr>
<td>class <em>newInv</em></td>
<td>105</td>
</tr>
</tbody>
</table>

*Table B.1 Index of Layout Generation Example Source*
Aug 12 15:12 2001 FifoStage.java Page 1

import Circuit.*;
import Constraints.*;
import Magic.*;
import java.util.*;
import java.io.*;

public class FifoStage {
    private Signal inTB, inFB, outTB, outFB;
    private Circuit c;

    public Signal inTB() { return inTB; }
    public Signal inFB() { return inFB; }
    public Signal outTB() { return outTB; }
    public Signal outFB() { return outFB; }
    public Circuit circuit() { return c; }

    public FifoStage(Signal _Vdd) {
        inTB = new DefaultSignal();
        inFB = new DefaultSignal();
        outTB = new DefaultSignal();
        outFB = new DefaultSignal();
        Power Vdd1 = new Power(_Vdd, 6);
        Power Vdd2 = new Power(_Vdd, 6);
        Ground Gnd = new Ground(6);
        RoutingRow rrl = new RoutingRow();
        RoutingRow rr2 = new RoutingRow();
        Transistor[] t = {
            new PTransistor(8),
            new PTransistor(8),
            new PTransistor(8),
        },
        new NTransistor(8),
        new NTransistor(8),
        new NTransistor(8),
        }
        new NTransistor(10),
        new NTransistor(10),
        new NTransistor(8),
        new NTransistor(6),
    }

    public FifoStage() {
        new PTransistor(10),
        new PTransistor(10),
        new PTransistor(10),
    },
Aug 15:12 2001 FifoStage.java Page 3

```java
public static void main(String[] args) {
    DefaultSignal Vdd = new DefaultSignal();
    FifoStage fs = new FifoStage(Vdd);
    Circuit circuit = fs.circuit();
    MagicOutput.outputToMagic(circuit, "FifoStage.mag", Scmos.tech(), "Opt";)
}
```

Aug 15:19 2001 Inv.java Page 1

```java
import Circuit.*;
import Constraints.*;
import Magic.*;
import java.util.*;
import java.io.*;

public class Inv {
    private Signal in, out;
    private Circuit c;
    public Signal in() {
        return in;
    }
    public Signal out() {
        return out;
    }
    public Circuit circuit() {
        return c;
    }

    public Inv(Signal _Vdd) {
        in = new DefaultSignal();
        out = new DefaultSignal();
        Power Vdd = new Power(_Vdd, 6);
        Ground Gnd = new Ground(6);
        RoutingRow rr = new RoutingRow();
        NTypeTransistor nT = new NTypeTransistor();
        PTypeTransistor pT = new PTypeTransistor();
    }
    public Inv(Signal _Vdd) {
        in = new DefaultSignal();
        out = new DefaultSignal();
        Power Vdd = new Power(_Vdd, 6);
        Ground Gnd = new Ground(6);
        RoutingRow rr = new RoutingRow();
        NTypeTransistor nT = new NTypeTransistor();
        PTypeTransistor pT = new PTypeTransistor();
    }
    public static void main(String[] args) {
        DefaultSignal Vdd = new DefaultSignal();
        Inv a = new Inv(Vdd);
        Inv b = new Inv(Vdd);
        Inv c = new Inv(Vdd);
        Inv d = new Inv(Vdd);
        Inv e = new Inv(Vdd);
        Inv f = new Inv(Vdd);
        Inv g = new Inv(Vdd);
        /*
         * Circuit circuit = a.circuit().left(b.circuit()).left(c.circuit());
         * .left(d.circuit()).left(e.circuit()).left(f.circuit()).left(g.circuit());
         */
```
Aug 12 15:19 2001  Inv.java Page 2

```java
Circuit circuit = a.circuit().left(b.circuit()).left(c.circuit()).
    .left(d.circuit()).left(e.circuit()).left(f.circuit());
Circuit circuit = a.circuit().left(b.circuit()).left(c.circuit()).*
    .left(d.circuit()).left(e.circuit()).left(f.circuit());
MagicOutput.outputToMagic(circuit, "Inv.mag", Scmos.tech(), "Opt");
```

---

Aug 12 15:22 2001  Inv.java Page 1

```java
import Circuit.*;
import Constraints.*;
import Magic.*;
import java.util.*;
import java.io.*;

public class Inv {  
    private Signal in, out;
    Circuit c;

    public Signal in() {  
        return in;
    }

    public Signal out() {  
        return out;
    }

    public Circuit circuit() {  
        return c;
    }

    public Inv(Signal _Vdd) {  
        in = new DefaultSignal();
        out = new DefaultSignal();
        Power Vdd = new Power(_Vdd, 6);
        Ground Gnd = new Ground(6);
        RoutingRow rr = new RoutingRow();
        NTransistor nT = new NTransistor(8);
        PTransistor pT = new PTransistor(8);
        RowNTrans row = new RowNTrans(nT);
        RowPTrans row = new RowPTrans(pT);
        nT.gate().connect(pT.gate()).connect(in);
        nT.drain().connect(pT.drain()).connect(out);
        nT.source().connect(Gnd);
        pT.source().connect(Vdd);
        Circuit c1 = new CircuitRow(Vdd);
        Circuit c2 = new CircuitRow(row.toTransRow());
        Circuit c3 = new CircuitRow(rr);
        Circuit c4 = new CircuitRow(row.toTransRow());
        Circuit c5 = new CircuitRow(Gnd);
        c = c5.below(c4).below(c3).below(c2).below(c1);
    }

    public static void main(String[] args) {  
        DefaultSignal Vdd = new DefaultSignal();
        Inv s1 = new Inv(Vdd);
        Inv s2 = new Inv(Vdd);
        Inv s1 = new Inv(Vdd);
        Inv s2 = new Inv(Vdd);
        Inv s1 = new Inv(Vdd);
        Inv s2 = new Inv(Vdd);
        Inv s1 = new Inv(Vdd);
        Inv s2 = new Inv(Vdd);
        Inv s1 = new Inv(Vdd);
        Inv s2 = new Inv(Vdd);
    }
```
Aug 12 15:22 2001 Invds.java Page 2

```java
Invs d2 = new Invs(Vdd);  
Invs e2 = new Invs(Vdd);  
Invs f2 = new Invs(Vdd);  
Invs a3 = new Invs(Vdd);  
Invs b3 = new Invs(Vdd);  
Invs c3 = new Invs(Vdd);  
Invs d3 = new Invs(Vdd);  
Invs e3 = new Invs(Vdd);  
Invs f3 = new Invs(Vdd);  
Invs g3 = new Invs(Vdd);  

Circuit circuit1 = a1.circuit().below(a2.circuit()).mirrorY();  
Circuit circuit2 = b1.circuit().below(b2.circuit()).mirrorY();  
Circuit circuit3 = c1.circuit().below(c2.circuit()).mirrorY();  
Circuit circuit4 = d1.circuit().below(d2.circuit()).mirrorY();  
Circuit circuit5 = e1.circuit().below(e2.circuit()).mirrorY();  
Circuit circuit6 = f1.circuit().below(f2.circuit()).mirrorY();  
Circuit circuit7 = g1.circuit().below(g2.circuit()).mirrorY();  
Circuit circuit = circuit1.left(circuit2.left(circuit3));  
MagicOutput.outputToMagic(circuit, 'Invs.mag', Smos.Smos());  
}
```

Aug 12 15:24 2001 newInv.java Page 1

```java
import Circuit.*;  
import Constraints.*;  
import Magic.*;  
import java.io.*;  

public class newInv {  
  private Signal in, out;  
  private Circuit c;  
  public Signal in() {  
    return in;  
  }  
  public Signal out() {  
    return out;  
  }  
  public Circuit circuit() {  
    return c;  
  }  
  public newInv(Signal in, int) {  
    in = new DefaultSignal();  
    out = new DefaultSignal();  
    Power Vdd = new Power(Vdd, 6);  
    Ground Gnd = new Ground();  
    RoutingRow rr = new RoutingRow();  
    NTransistor nT = new NTransistor();  
    PTransistor pT = new PTransistor();  
    RowPrTrans new = new RowPrTrans();  
    RowPrTrans prow = new RowPrTrans(pT);  
    nT.gate().connect(pT.gate()).connect(in);  
    pT.drain().connect(pT.drain()).connect(out);  
    nT.source().connect(Gnd);  
    pT.source().connect(Vdd);  
    Circuit c1 = new CircuitRow(Vdd);  
    Circuit c2 = new CircuitRow(prow.toTransrow());  
    Circuit c3 = new CircuitRow(rr);  
    Circuit c4 = new CircuitRow([Row][Row].toTransrow());  
    Circuit c5 = new CircuitRow(Gnd);  
    c = c5.below(c4).below(c3).below(c2).below(c1);  
  }  
  public static void main(String[] args) {  
    DefaultSignal Vdd = new DefaultSignal();  
    newInv a = new newInv(Vdd);  
    Circuit circuit = a.circuit();  
    MagicOutput.outputToMagic(circuit, 'newInvs.mag', Smos.Smos());  
  }
```

This appendix contains a sample input data file in the MPS format to the LPABO solver and a sample output file from the LPABO solver. Table C.1 lists the entries for each section of the input or output file.

<table>
<thead>
<tr>
<th>Time</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>circuit.dat (input data file for LPABO)</td>
<td>107</td>
</tr>
<tr>
<td>declaration of rows of equality constraints</td>
<td>107</td>
</tr>
<tr>
<td>declaration of rows of inequality constraints</td>
<td>107</td>
</tr>
<tr>
<td>declaration of the objective function</td>
<td>108</td>
</tr>
<tr>
<td>declaration of the equality/inequality coefficients</td>
<td>108</td>
</tr>
<tr>
<td>declaration of the equality/inequality right-hand side</td>
<td>110</td>
</tr>
<tr>
<td>lpabo.out (output file from LPABO)</td>
<td>112</td>
</tr>
<tr>
<td>variable values</td>
<td>112</td>
</tr>
<tr>
<td>dual variable values</td>
<td>112</td>
</tr>
<tr>
<td>value of the objective function</td>
<td>114</td>
</tr>
</tbody>
</table>

Table C.1 Index of LPABO Input and Output Data Format
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>V17</strong> R15 1.0</td>
<td><strong>V27</strong> R57 1.0</td>
</tr>
<tr>
<td>V20 R16 -1.0</td>
<td>V28 R58 1.0</td>
</tr>
<tr>
<td>V21 R17 1.0</td>
<td>V29 R59 1.0</td>
</tr>
<tr>
<td>V24 R18 -1.0</td>
<td>V30 R60 1.0</td>
</tr>
<tr>
<td>V25 R19 1.0</td>
<td>V31 R61 1.0</td>
</tr>
<tr>
<td>V28 R20 -1.0</td>
<td>V32 R62 1.0</td>
</tr>
<tr>
<td>V29 R21 1.0</td>
<td>V33 R63 1.0</td>
</tr>
<tr>
<td>V32 R22 1.0</td>
<td>V34 R64 1.0</td>
</tr>
<tr>
<td>V33 R23 1.0</td>
<td>V35 R65 1.0</td>
</tr>
<tr>
<td>V36 R24 -1.0</td>
<td>V36 R66 1.0</td>
</tr>
<tr>
<td>V37 R25 1.0</td>
<td>V37 R67 1.0</td>
</tr>
<tr>
<td>V40 R26 -1.0</td>
<td>V38 R68 1.0</td>
</tr>
<tr>
<td>V41 R27 1.0</td>
<td>V39 R69 1.0</td>
</tr>
<tr>
<td>V45 R28 -1.0</td>
<td>V40 R70 1.0</td>
</tr>
<tr>
<td>V46 R29 1.0</td>
<td>V41 R71 1.0</td>
</tr>
<tr>
<td>V48 R30 -1.0</td>
<td>V42 R72 1.0</td>
</tr>
<tr>
<td>V44 R31 1.0</td>
<td>V43 R73 1.0</td>
</tr>
<tr>
<td>V49 R32 -1.0</td>
<td>V44 R74 1.0</td>
</tr>
<tr>
<td>V50 R34 1.0</td>
<td>V45 R75 1.0</td>
</tr>
<tr>
<td>V50 R35 -1.0</td>
<td>V46 R76 1.0</td>
</tr>
<tr>
<td>V51 R36 1.0</td>
<td>V47 R77 1.0</td>
</tr>
<tr>
<td>V52 R37 -1.0</td>
<td>V48 R78 1.0</td>
</tr>
<tr>
<td>V55 R38 1.0</td>
<td>V49 R79 1.0</td>
</tr>
<tr>
<td>V56 R39 -1.0</td>
<td>V50 R80 1.0</td>
</tr>
<tr>
<td>V51 R40 1.0</td>
<td>V51 R81 1.0</td>
</tr>
<tr>
<td>V52 R41 1.0</td>
<td>V52 R82 1.0</td>
</tr>
<tr>
<td>V60 R42 -1.0</td>
<td>V53 R83 1.0</td>
</tr>
<tr>
<td>V44 R43 1.0</td>
<td>V54 R84 1.0</td>
</tr>
<tr>
<td>V49 R44 1.0</td>
<td>V55 R85 1.0</td>
</tr>
<tr>
<td>V50 R45 1.0</td>
<td>V56 R86 1.0</td>
</tr>
<tr>
<td>V6 R46 1.0</td>
<td>V57 R87 1.0</td>
</tr>
<tr>
<td>V8 R47 1.0</td>
<td>V58 R88 1.0</td>
</tr>
<tr>
<td>V9 R48 1.0</td>
<td>V59 R89 1.0</td>
</tr>
<tr>
<td>V9 R49 1.0</td>
<td>V60 R90 1.0</td>
</tr>
<tr>
<td>V10 R50 1.0</td>
<td>V61 R91 1.0</td>
</tr>
<tr>
<td>V11 R51 1.0</td>
<td>V62 R92 1.0</td>
</tr>
<tr>
<td>V12 R52 1.0</td>
<td>V63 R93 1.0</td>
</tr>
<tr>
<td>V13 R53 1.0</td>
<td>V64 R94 1.0</td>
</tr>
<tr>
<td>V14 R54 1.0</td>
<td>V65 R95 1.0</td>
</tr>
<tr>
<td>V15 R55 1.0</td>
<td>V66 R96 1.0</td>
</tr>
<tr>
<td>V16 R56 1.0</td>
<td>V67 R97 1.0</td>
</tr>
<tr>
<td>V18 R57 1.0</td>
<td>V68 R98 1.0</td>
</tr>
<tr>
<td>V19 R58 1.0</td>
<td>V69 R99 1.0</td>
</tr>
<tr>
<td>V20 R59 1.0</td>
<td>V70 R100 1.0</td>
</tr>
<tr>
<td>V21 R60 1.0</td>
<td>V9 R101 1.0</td>
</tr>
<tr>
<td>V22 R61 1.0</td>
<td>V10 R102 1.0</td>
</tr>
<tr>
<td>V23 R62 1.0</td>
<td>V11 R103 1.0</td>
</tr>
<tr>
<td>V24 R63 1.0</td>
<td>V12 R104 1.0</td>
</tr>
<tr>
<td>V25 R64 1.0</td>
<td>V13 R105 1.0</td>
</tr>
<tr>
<td>V26 R65 1.0</td>
<td>V14 R106 1.0</td>
</tr>
<tr>
<td>V27 R66 1.0</td>
<td>V15 R107 1.0</td>
</tr>
<tr>
<td>V28 R67 1.0</td>
<td>V16 R108 1.0</td>
</tr>
<tr>
<td>V29 R68 1.0</td>
<td>V17 R109 1.0</td>
</tr>
<tr>
<td>V30 R69 1.0</td>
<td>V18 R110 1.0</td>
</tr>
<tr>
<td>V31 R70 1.0</td>
<td>V19 R111 1.0</td>
</tr>
<tr>
<td>V32 R71 1.0</td>
<td>V20 R112 1.0</td>
</tr>
<tr>
<td>V33 R72 1.0</td>
<td>V21 R112 1.0</td>
</tr>
<tr>
<td>V34 R73 1.0</td>
<td>V22 R112 1.0</td>
</tr>
<tr>
<td>V35 R74 1.0</td>
<td>V23 R112 1.0</td>
</tr>
<tr>
<td>V36 R75 1.0</td>
<td>V24 R112 1.0</td>
</tr>
<tr>
<td>V37 R76 1.0</td>
<td>V25 R112 1.0</td>
</tr>
<tr>
<td>V38 R77 1.0</td>
<td>V26 R112 1.0</td>
</tr>
</tbody>
</table>
### PR_DATA : input data ###

1 Problem Name : 5.848024e+02
2 Problem Type : Minimizing Problem
3 Output Type : Detail output
4 Number of constraints : 153
5 Number of variables : 62
6 Number of <= type constraints : 0
7 Number of >= type constraints : 143
8 Number of = type constraints : 10
9 Number of Nonzeros in A : 182
10 Density of A : 1.919

<table>
<thead>
<tr>
<th>In</th>
<th>Primal Gap</th>
<th>Dual Gap</th>
<th>Duality GAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.38248</td>
<td>0.89397</td>
<td>1.01036</td>
</tr>
<tr>
<td>2</td>
<td>1.38832</td>
<td>0.049053</td>
<td>1.10228</td>
</tr>
<tr>
<td>3</td>
<td>0.042014</td>
<td>0.00294752</td>
<td>1.43388</td>
</tr>
<tr>
<td>4</td>
<td>6.7413e-05</td>
<td>4.0537e-06</td>
<td>0.00564229</td>
</tr>
<tr>
<td>5</td>
<td>4.7936e-09</td>
<td>2.15956e-10</td>
<td>3.0123e-07</td>
</tr>
<tr>
<td>6</td>
<td>3.5827e-05</td>
<td>1.9599e-05</td>
<td>2.8432e-05</td>
</tr>
<tr>
<td>7</td>
<td>4.60712e-09</td>
<td>6.1296e-09</td>
<td>3.6194e-09</td>
</tr>
</tbody>
</table>

### OPTIMAL SOLUTION ###

<Primal Variables>

<table>
<thead>
<tr>
<th>variable</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>X( V53)</td>
<td>6.718869e+02</td>
</tr>
<tr>
<td>X( V54)</td>
<td>6.718869e+02</td>
</tr>
<tr>
<td>X( V10)</td>
<td>1.00000e+03</td>
</tr>
<tr>
<td>X( V14)</td>
<td>1.00000e+03</td>
</tr>
<tr>
<td>X( V47)</td>
<td>6.718869e+02</td>
</tr>
<tr>
<td>X( V32)</td>
<td>1.00000e+03</td>
</tr>
<tr>
<td>X( V26)</td>
<td>1.00000e+03</td>
</tr>
<tr>
<td>X( V30)</td>
<td>1.00000e+03</td>
</tr>
<tr>
<td>X( V43)</td>
<td>1.00000e+03</td>
</tr>
<tr>
<td>X( V42)</td>
<td>1.00000e+03</td>
</tr>
<tr>
<td>X( V45)</td>
<td>6.771054e+02</td>
</tr>
<tr>
<td>X( V30)</td>
<td>6.821054e+02</td>
</tr>
<tr>
<td>X( V49)</td>
<td>6.811054e+02</td>
</tr>
<tr>
<td>X( V8)</td>
<td>5.848024e+02</td>
</tr>
<tr>
<td>X( V12)</td>
<td>5.848024e+02</td>
</tr>
<tr>
<td>X( V20)</td>
<td>5.848024e+02</td>
</tr>
<tr>
<td>X( V24)</td>
<td>5.848024e+02</td>
</tr>
<tr>
<td>X( V38)</td>
<td>5.848024e+02</td>
</tr>
<tr>
<td>X( V12)</td>
<td>5.848024e+02</td>
</tr>
<tr>
<td>X( V40)</td>
<td>5.848024e+02</td>
</tr>
<tr>
<td>X( V58)</td>
<td>6.891054e+02</td>
</tr>
<tr>
<td>X( V44)</td>
<td>6.818869e+02</td>
</tr>
<tr>
<td>X( V51)</td>
<td>6.841054e+02</td>
</tr>
<tr>
<td>X( V52)</td>
<td>6.851054e+02</td>
</tr>
<tr>
<td>X( V9)</td>
<td>5.848024e+02</td>
</tr>
<tr>
<td>X( V13)</td>
<td>5.848024e+02</td>
</tr>
<tr>
<td>X( V21)</td>
<td>5.848024e+02</td>
</tr>
<tr>
<td>X( V25)</td>
<td>5.848024e+02</td>
</tr>
</tbody>
</table>

<Dual Variables>

<table>
<thead>
<tr>
<th>variable</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y( R1)</td>
<td>2.53613e+00</td>
</tr>
<tr>
<td>Y( R2)</td>
<td>1.00000e+00</td>
</tr>
<tr>
<td>Y( R3)</td>
<td>3.53428e+01</td>
</tr>
<tr>
<td>Y( R4)</td>
<td>4.61931e+01</td>
</tr>
<tr>
<td>Y( R5)</td>
<td>2.00000e+00</td>
</tr>
<tr>
<td>Y( R6)</td>
<td>-2.99911e+00</td>
</tr>
<tr>
<td>Y( R7)</td>
<td>2.46634e+00</td>
</tr>
<tr>
<td>Y( R8)</td>
<td>2.00000e+00</td>
</tr>
<tr>
<td>Y( R9)</td>
<td>2.46634e+00</td>
</tr>
<tr>
<td>Y( R10)</td>
<td>1.00000e+00</td>
</tr>
<tr>
<td>Y( R11)</td>
<td>-0.00000e+00</td>
</tr>
<tr>
<td>Y( R12)</td>
<td>-0.00000e+00</td>
</tr>
<tr>
<td>Y( R13)</td>
<td>1.00000e+00</td>
</tr>
<tr>
<td>Y( R14)</td>
<td>1.00000e+00</td>
</tr>
<tr>
<td>Y( R15)</td>
<td>-0.00000e+00</td>
</tr>
<tr>
<td>Y( R16)</td>
<td>1.00000e+00</td>
</tr>
<tr>
<td>Y( R17)</td>
<td>1.00000e+00</td>
</tr>
</tbody>
</table>
Y(  R130)  0.000000e+00
Y(  R131)  0.000000e+00
Y(  R132)  6.267140e-14
Y(  R133)  4.213090e-14
Y(  R134)  1.000000e+00
Y(  R135)  0.000000e+00
Y(  R136)  7.385261e-09
Y(  R137)  4.797925e-14
Y(  R138)  5.207759e-14
Y(  R139)  4.679963e-14
Y(  R140)  4.666474e-14
Y(  R141)  5.385395e-14
Y(  R142)  4.655021e-14
Y(  R143)  4.533775e-14
Y(  R144)  4.533073e-14
Y(  R145)  4.689182e-14
Y(  R146)  4.769023e-14
Y(  R147)  -3.054329e-21
Y(  R148)  4.641981e-14
Y(  R149)  4.524062e-14
Y(  R150)  4.227386e-14
Y(  R151)  4.636087e-14
Y(  R152)  -7.385171e-09
Y(  R153)  4.343241e-14

The value of the objective function = -6.9219999997e+03

<Time Summary>

Reading input data  =  0.74 sec.
Preprocessing       =  0.00 sec.
Scaling            =  0.00 sec.
Ordering           =  0.00 sec.
Symbolic factorization  =  0.00 sec.
Solving            =  0.82 sec.
Printing solution  =  0.00 sec.
Total time         =  0.76 sec.

LPABO end.
This appendix includes a sample file in the Magic format, the generated "FifoStage.mag".