REST

A Leaf Cell Design System

R. C. Mosteller

Computer Science Department
California Institute of Technology

4317:TR:81

CALIFORNIA INSTITUTE OF TECHNOLOGY

Computer Science Department

REST

A LEAF CELL DESIGN SYSTEM

by

R.C. Mosteller

Silicon Structures Project

Technical Report 4317

Submitted in Partial Fulfillment
of the Requirements for the Degree
of Master of Science
December 1981

This project was sponsored by the Silicon Structures Project.
Copyright © 1981 by California Institute of Technology.

Acknowledgements

The initial development of ideas presented herein came from the meetings of the Geometry Group headed by Ivan Sutherland and conversations with John Williams. The Geometry Group participants were Ivan Sutherland, Ed McGrath, Don Oestreicher, Eric Barton, Telle Whitney and myself. The group's main intent was to explore polygon geometry and desgin rules checking. I would like to express my thanks to all the participants for their help in this project.

I would like to give thanks to John Gray for his patience, assistance and inspiration for this project.

I would like to thank my advisor Jim Kajiya.

Abstract

This thesis describes a leaf cell design system, REST, Richard's Editor for Sticks. REST is intended to be used for the preparation of the lowest level cells in an integrated circuit design. A stick notation is used in the editing process. Given a structured design methodology any design task can be separated into two parts: 1) leaf cell design and 2) composition cell design. This tool addresses the first of these tasks, although it may also be used for general manipulation of stick diagrams. A table driven compaction algorithm is presented. This graph based algorithm uses a weighted affinity factor to reduce total polysilicon and diffusion wire length. A suite of utilities provide functions such as file interface, physical mapping, annotation, etc. consistent with a set of design rules. The system has been implemented in Simula on a DEC 20 computer, and works in conjunction with a limited functional diagramming system. The design rules, models and stick interpretation are table driven and can be changed for various technologies. Currently REST is being used for NMOS technology. A community of users have used the REST system to prepare a number of designs resulting in a substantial reduction of design time. In addition, the system is currently being used at a major computer manufacturer in conjunction with a VLSI design course.

Table of Contents

List of Figures

1. INTRODUCTION

We are now in a new revolution, the microelectronic revolution[Noyce 77]. This microelectronic revolution has provided the means to put 50,000 or more circuit elements on a single chip. As each year goes by, semiconductor manufacturers are able to put an ever increasing number of circuit elements on a single chip as shown by Moore's law in figure 1-1. With the increase in circuit complexity of a VLSI chip, the design time also increases at a rapid rate[Moore 79]. In order to cope with the larger complexity and longer design time, a structured design methodology should be adopted.

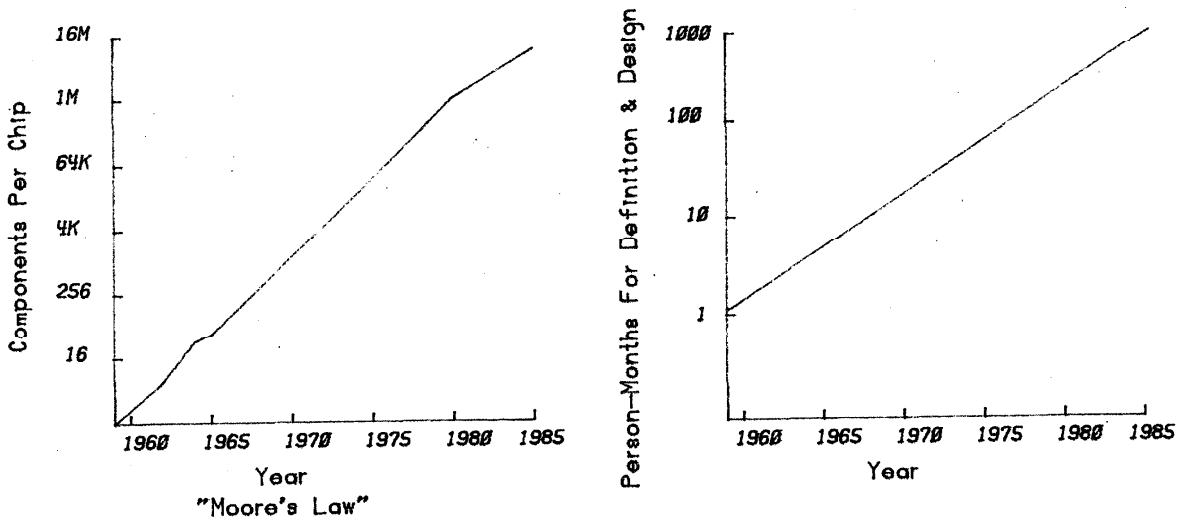


Figure 1-1: Moore's Law

Structured integrated circuit design is similar to the structured design of programs[Gray 80]. The design is partitioned into small manageable pieces called cells which are similar to procedures in programming languages. Each cell is composed of other cells or primitive elements which form a hierarchy of cells akin to the nesting of procedures. The number of elements in a cell is kept small, approximately seven plus or minus two, corresponding to the average person's short term memory limitation[Miller 56].

In structured programming it is desirable to have no global variables and likewise in structured design global wires are undesirable[Wulf 73]. A global wire is one that runs across several cells and is not part of their definition. That is, wires are not laid on top of cells, they must be part of the cell definition. Connection to the cells occur at the perimeter of the cell. This structured approach promotes a regular structure with a consistent wiring strategy. This design style for VLSI systems is presented in Mead and Conway [MEAD 80].

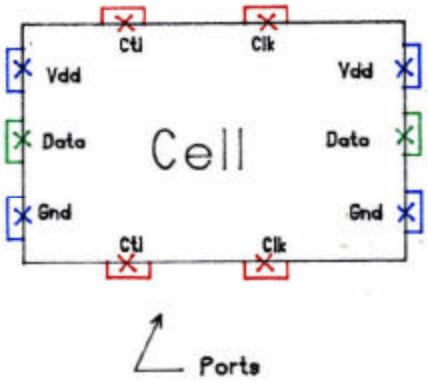


Figure 1-2: Cell Model

There are various shapes for tiling a VLSI chip. Of these shapes there is at least some understanding of packing four sided polygons. A four sided polygon provides four interfaces for connecting cells. The cell model used in this philosophy is a rectangle as shown in figure 1-2. All geometrical data is on the interior of the rectangle and considered to be the property of the cell. The cell contains the structural and physical data to realize the cell. The ports or connectors of the cell are the interface to the outside world. Each port defines a position on the perimeter of the cell where a connection is to be accomplished. These ports protrude from the boundary of the cell like fingers.

The spatial composition of two cells is performed by placing the cells side by side where the ports of each cell correspond by layer and relative location. Thus, cells are connected by abutment. However, there must be a one-to-one correspondence of ports

In structured programming it is desirable to have no global variables and likewise in structured design global wires are undesirable[Wulf 73]. A global wire is one that runs across several cells and is not part of their definition. That is, wires are not laid on top of cells, they must be part of the cell definition. Connection to the cells occur at the perimeter of the cell. This structured approach promotes a regular structure with a consistent wiring strategy. This design style for VLSI systems is presented in Mead and Conway [MEAD 80].

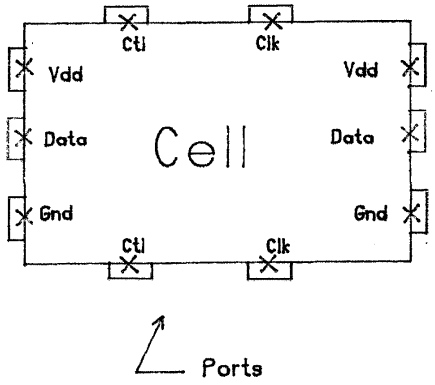


Figure 1-2: Cell Model

There are various shapes for tiling a VLSI chip. Of these shapes there is at least some understanding of packing four sided polygons. A four sided polygon provides four interfaces for connecting cells. The cell model used in this philosophy is a rectangle as shown in figure 1-2. All geometrical data is on the interior of the rectangle and considered to be the property of the cell. The cell contains the structural and physical data to realize the cell. The ports or connectors of the cell are the interface to the outside world. Each port defines a position on the perimeter of the cell where a connection is to be accomplished. These ports protrude from the boundary of the cell like fingers.

The spatial composition of two cells is performed by placing the cells side by side where the ports of each cell correspond by layer and relative location. Thus, cells are connected by abutment. However, there must be a one-to-one correspondence of ports

on the abutment edge. This abutment ensures that there is no global wiring by only allowing connections to be accomplished by abutment. The abutment of cells is applied rigorously throughout the design of a chip.

When the connectors of two abuting cells do not line up properly the cells may either be stretched or a new wiring cell may be inserted between the two cells as shown in figure 1-3. A cell may be stretched along the X or Y axis independently at any port as shown by Rowson[Rowson 80]. This ability for a cell to stretch can be used in the composition process. A wiring cell may be used in lieu of stretching, where desirable.

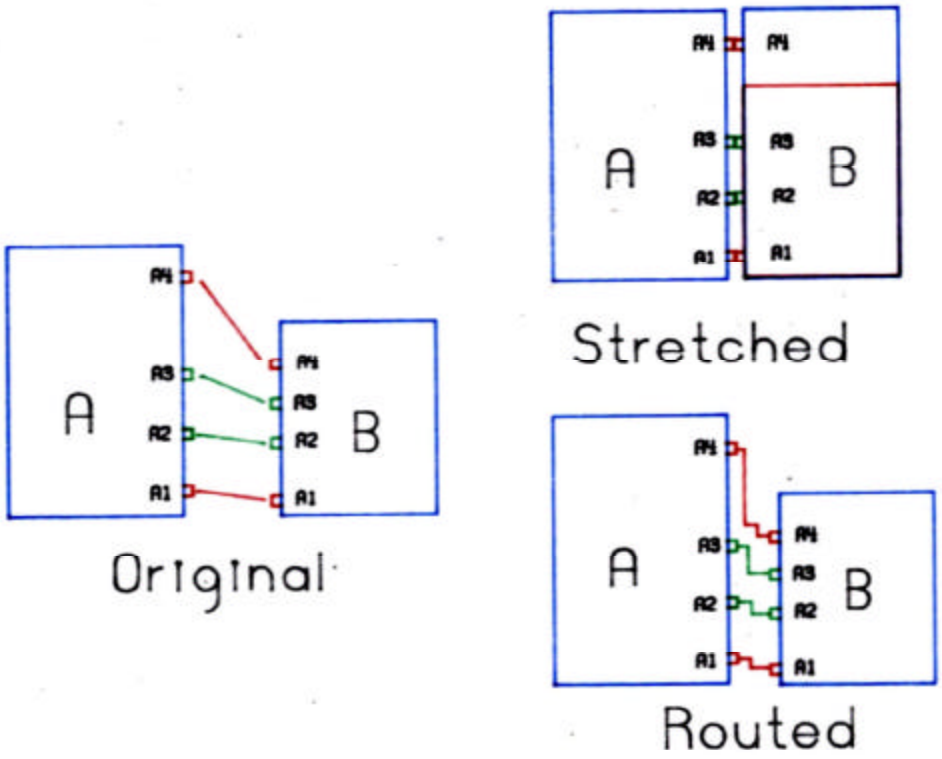


Figure 1-3: Composition of Two Cells

In Figure 1-3 two cells are shown in the composition process. The choice of whether to stretch or to route can either be directed from the user or decided by the

on the abutment edge. This abutment ensures that there is no global wiring by only allowing connections to be accomplished by abutment. The abutment of cells is applied rigorously throughout the design of a chip.

When the connectors of two abuting cells do not line up properly the cells may either be stretched or a new wiring cell may be inserted between the two cells as shown in figure 1-3. A cell may be stretched along the X or Y axis independently at any port as shown by Rowson[Rowson 80]. This ability for a cell to stretch can be used in the composition process. A wiring cell may be used in lieu of stretching, where desirable.

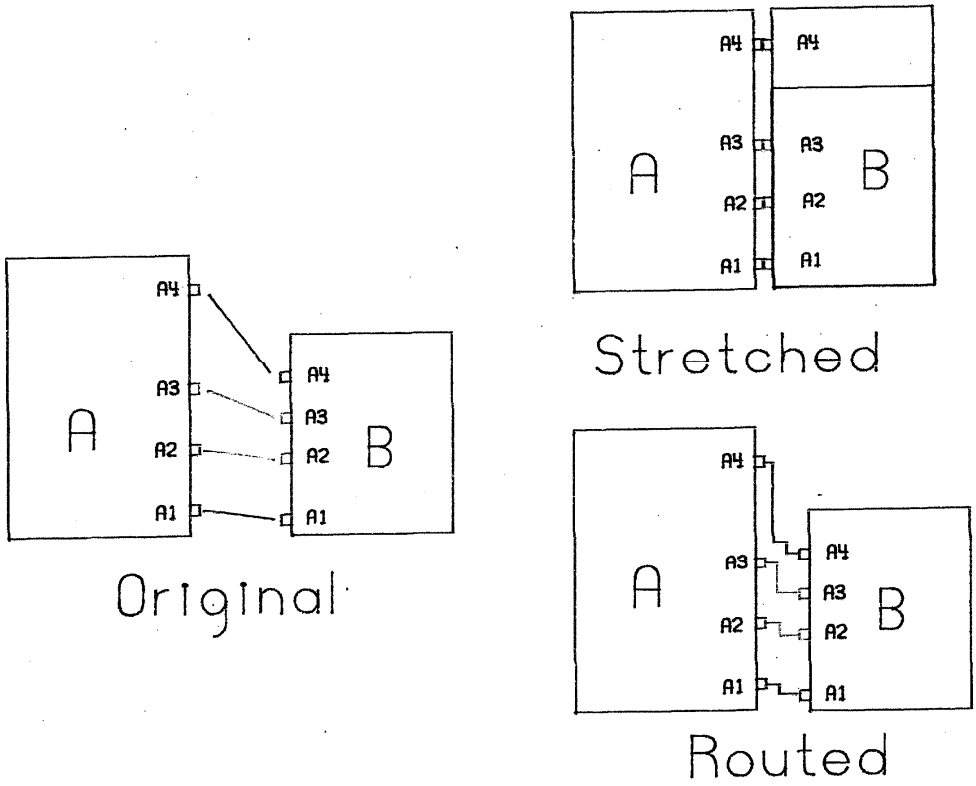


Figure 1-3: Composition of Two Cells

In Figure 1-3 two cells are shown in the composition process. The choice of whether to stretch or to route can either be directed from the user or decided by the

composition tool. An important point about the composition of two cells is that the only necessary information to specify the composition is the sides of abutment[Rowson 80]. The interconnection relationship between the cells is defined with this minimal amount of data. However, there must be a one-to-one mapping of the ports with the two cells.

There will be topological problems for the designer to solve using this model. That is the wiring strategy must be defined during the floor planning phase of the chip design. Not as an after-thought. In general this model insures that the wiring strategy is defined in the planning stages of the design.

1.1 Structured Design

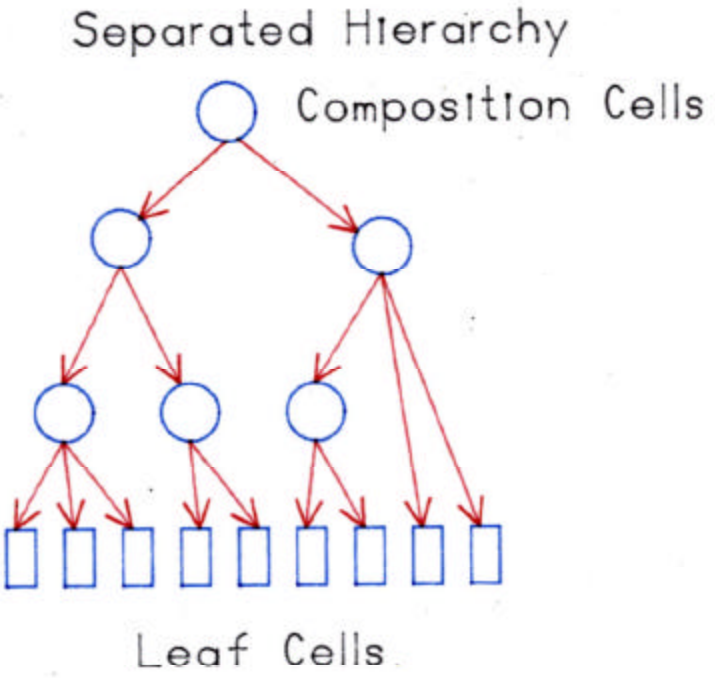


Figure 1-4: Separated Hierarchy

The cells of a structured design are partitioned into two types: 1) composition cells and 2) leaf cells. A composition cell contains either other composition cells or leaf

composition tool. An important point about the composition of two cells is that the only necessary information to specify the composition is the sides of abutment[Rowson 80]. The interconnection relationship between the cells is defined with this minimal amount of data. However, there must be a one-to-one mapping of the ports with the two cells.

There will be topological problems for the designer to solve using this model. That is the wiring strategy must be defined during the floor planning phase of the chip design. Not as an after-thought. In general this model insures that the wiring strategy is defined in the planning stages of the design.

1.1 Structured Design

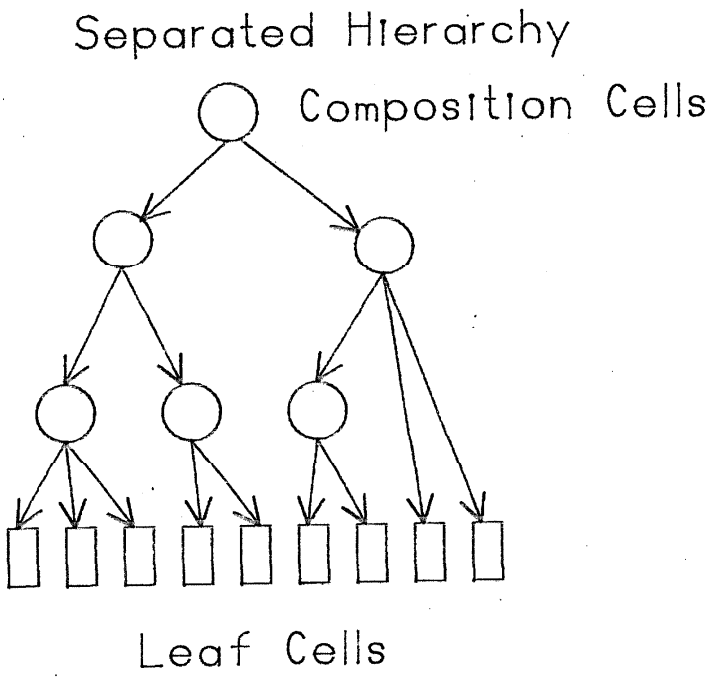


Figure 1-4: Separated Hierarchy

The cells of a structured design are partitioned into two types: 1) composition cells and 2) leaf cells. A composition cell contains either other composition cells or leaf

cells, while a leaf cell contains only wiring and primitive circuit elements. This partitioning of cell types is called a separated hierarchy [Rowson 80]. A graphical representation of a separated hierarchy is shown in figure 1-4. Note that the leaf cells are at the leaves of the tree while the composition cells form the branches. Each cell, being either a composition cell or a leaf cell, contains a small number of elements that are easily grasped by a designer. This is only true if the designer has done his homework on his cell designs by keeping the number of elements within the cells small. It then follows that a designer can comfortably work on a single cell in the hierarchy of his chip.

Following the above principles a separated hierarchy allows large designs to be managed. That is, a designer need only consider his current cell, the cells' interface ports and its interface to other cells which the cell instances[1] and not all the cells in the design. This also provides the ability to partition the design among many designers.

1.2 Methodology and Design Process Flow

It has been shown by various people that transistors are cheap whereas wiring is expensive[Sutherland 77, MEAD 80, Heller 79]. The rough size estimate of a VLSI chip can be calculated from the wires only. Initial design considerations should include the wiring strategy. This philosophy promotes regularity in design with consideration for wiring strategy.

Any design process includes both the initial design of the system followed by design refinement. Each cell of the design may be redesigned several times before completion. Consider floor planning. In the design process several different floor plans will be devised before a final floor plan is reached. Each plan would probably be simulated on a functional simulator and evaluated in terms of area.

The design philosophy and design tools should be amenable to changes in the design.

---

[1] An instance is a call or reference to a cell which is used in this cell.

The cost[2] of each change should be proportional to the degree of the change. An example is a large chip composed of many leaf and composition cells. A change occurs to a leaf cell which adds additional logic to the cell causing an increase in its size but does not increase either the number or the relationship of its ports. The update layout process should only require an execution of the assembler without making changes to the actual composition instructions. Although if a change was desired to the floor plan the composition instructions would need to be changed.

There are therefore two major divisions in a computer aided design system consistent with this design philosophy: a leaf cell design system, and a composition tool for constructing the VLSI chip from leaf cells and composition cells. This thesis in part describes an example of the former, REST. This view of a design system yields figure 1-5. This system is a simple design system with the interfaces between parts being standard text files. These text files provide the intermediate form between the various pieces of the system. With this system, it is not necessary nor desirable to have an exotic data base system. What is required is a flexible file system.

The essential task in leaf cell design is to capture both the topology and the layout for each cell. This is best accomplished by a sticks notation devised by Williams [Williams 77] and described in Mead and Conway [MEAD 80]. The REST system provides the means to process the stick notation and transform this notation to an abstract stick representation, compact the representation, and transform to a layout.

The interface from REST to the other parts of the system is through a simple disk file using a standard form called the Sticks Standard[Trimberger 80]. This standard provides a clean interface between the REST system and any composition tool. The testing of the leaf cells is done through a circuit simulator. This simulator accepts the Sticks Standard.

The task of assembling the chip is performed by a composition tool. The input to the composition tool is both the leaf cells from REST and composition cells which define

---

[2]The cost being the amount of designer effect plus the computer time to a lesser degree.

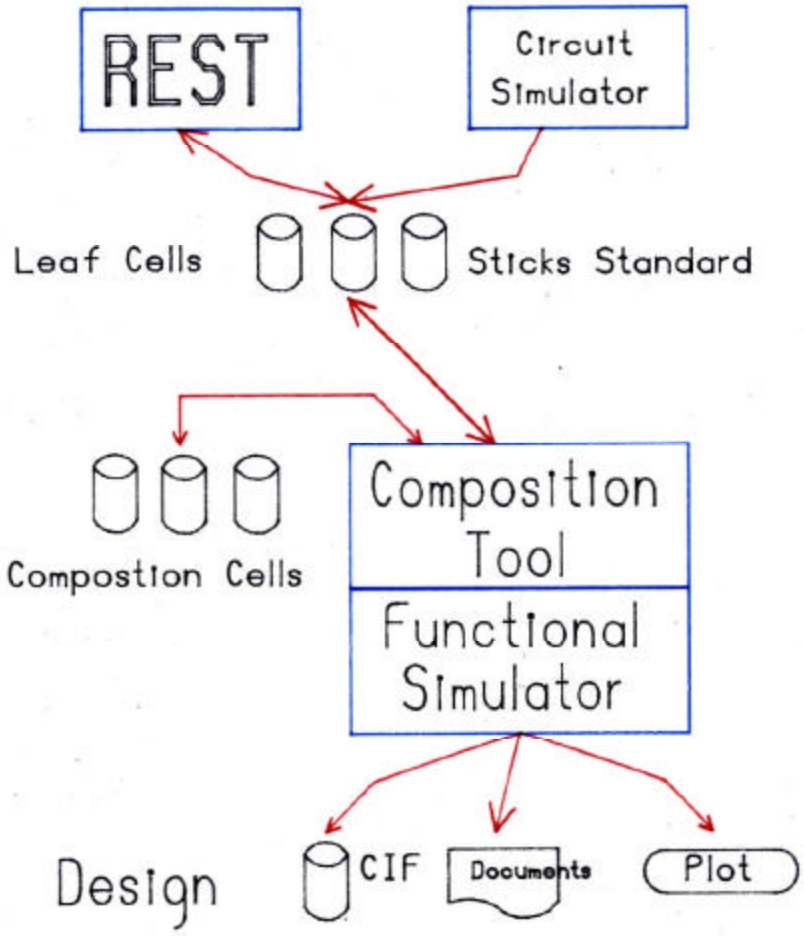


Figure 1-5: Design System

the spatial positioning, structure and behavioral data of the design. The description of the composition is best accomplished in a textual manner with a composition language such as in the SPAM Language[Segal 80]. The main point here is that the spatial positioning, structural and behavioral data is captured through a textual language in a simple manner at one place in the design description.

The textual description of composition in SPAM defines the abutting relationship in a hierarchical manner. The description states which sides of each individual cell abut. The actual interconnection of the port is specified indirectly by a one-to-one mapping

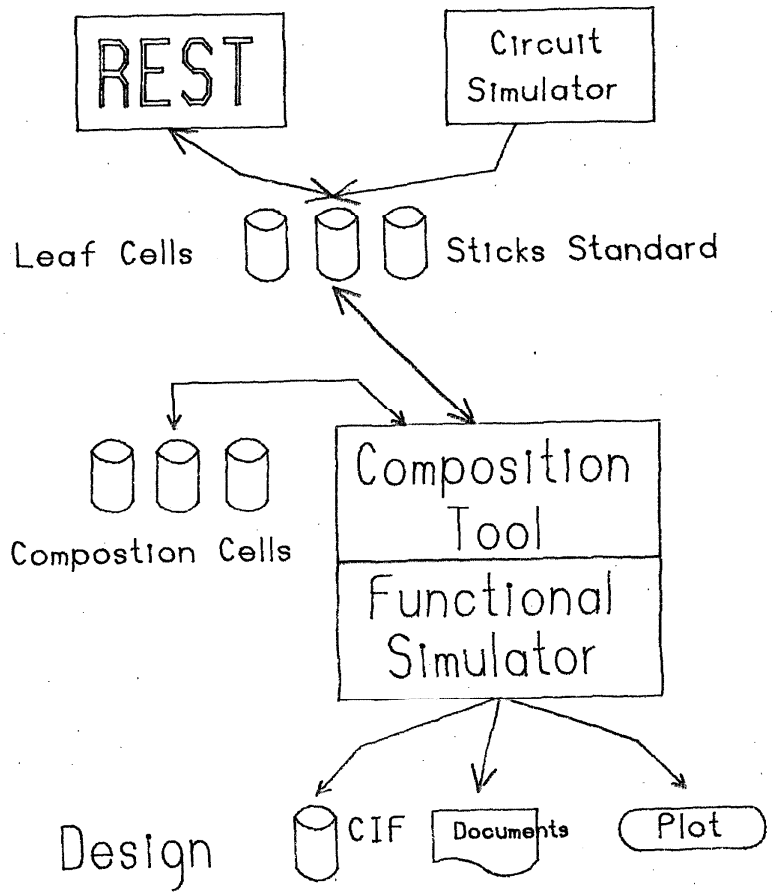


Figure 1-5: Design System

the spatial positioning, structure and behavioral data of the design. The description of the composition is best accomplished in a textual manner with a composition language such as in the SPAM Language[Segal 80]. The main point here is that the spatial positioning, structural and behavioral data is captured through a textual language in a simple manner at one place in the design description.

The textual description of composition in SPAM defines the abutting relationship in a hierarchical manner. The description states which sides of each individual cell abut. The actual interconnection of the port is specified indirectly by a one-to-one mapping

along the side of abutment. The advantage of this is that it is not necessary to specify each individual connection, but only just the abuting edges. A description of a complete chip in SPAM takes very little code. In addition, changes in the design can readily be accomplished with little pain.

The actual composition or tiling of the VLSI chip is performed by aligning the pitches of the cells or by adding wiring cells. Alignment is performed by stretching a cell to fit its environment. The stretching is performed by the composition tool since detailed knowledge of the geometry is not necessary in the stretching process. Compaction could also be used in the alignment process by adding constraints to the leaf cells and using REST to process the cell.

The geometry created in this manner would not need to be design rule checked, although design rule checking would be necessary for program verification. The reason for this is that leaf cells are maintained error-free by the nature of the design process. The composition process only stretches or adds correct routing cells. This system and design philosophy follows the principle of correctness by construction.

The functional debugging of a design would be accomplished with a system like the SPAM system[Segal 80]. In the SPAM system the composition cells also contain the behavioral description in addition to spatial positioning information and structural data. SPAM has a multi-valued functional simulator capable of simulating at any level of the hierarchy.

The output of the composition tool is a design description in text format known as the Caltech Intermediate Form (CIF) [MEAD 80, Sproull 79]. It is also possible to request various plots and some additional documentation. CIF can be processed by various silicon foundry tools to produce mask geometry.

2. OBJECTIVES IN LEAF CELL DESIGN

2.1 Preamble

Leaf cell design involves capturing both the topology and layout. This can be accomplished by using a sticks notation devised by Williams[Williams 77]. A stick diagram is a stylized symbolic layout. This type of notation is between a circuit diagram and an actual layout. It contains all the information in a circuit diagram and relative physical structure of a layout. An example of a shift register cell is shown in figure 2-1. Thin colored lines or wires are used to represent interconnection among various components and ports. Each color represents a wiring mask level.

This view of a one-to-one mapping of stick colors to mask levels is the case for the simple problem as in NMOS. For the complex problem there may be a one-to-many mapping as in the case of CMOS.

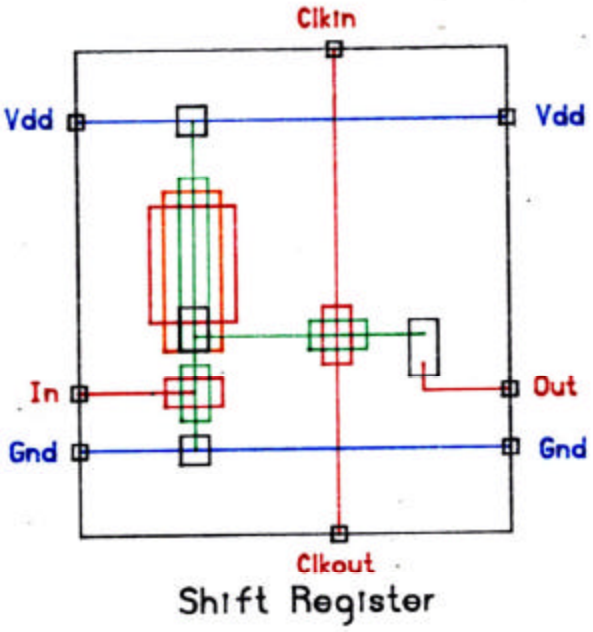


Figure 2-1: Stick Drawing of a Shift Register

In figure 2-1 the blue wire represents metal, the red wire represents polysilicon,

2. OBJECTIVES IN LEAF CELL DESIGN

2.1 Preamble

Leaf cell design involves capturing both the topology and layout. This can be accomplished by using a sticks notation devised by Williams[Williams 77]. A stick diagram is a stylized symbolic layout. This type of notation is between a circuit diagram and an actual layout. It contains all the information in a circuit diagram and relative physical structure of a layout. An example of a shift register cell is shown in figure 2-1. Thin colored lines or wires are used to represent interconnection among various components and ports. Each color represents a wiring mask level.

This view of a one-to-one mapping of stick colors to mask levels is the case for the simple problem as in NMOS. For the complex problem there may be a one-to-many mapping as in the case of CMOS.

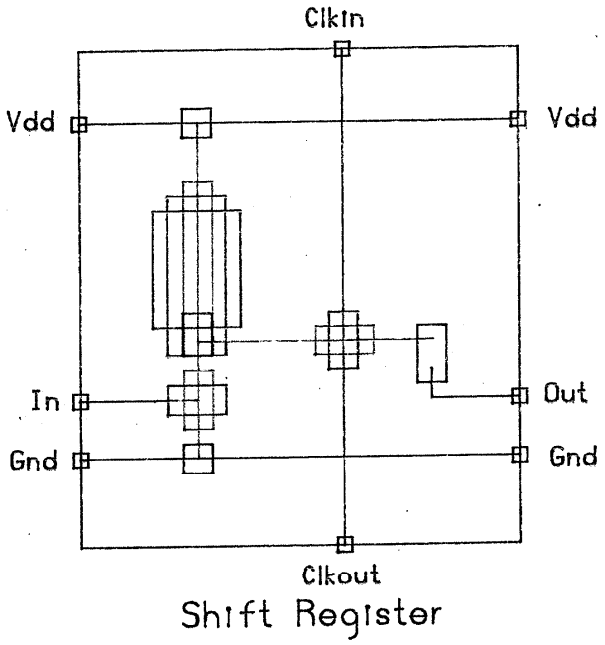


Figure 2-1: Stick Drawing of a Shift Register

In figure 2-1 the blue wire represents metal, the red wire represents polysilicon,

and the green wire represents diffusion. The important point about the sticks drawing is that the various layers are represented by colored lines. These colors provide an easily distinguishable view of mask layer without resorting to dashed lines or other methods. It is important to have the means to use colors in the sticks notation.

A designer need only worry about the circuit description and the topology of a cell in the sticks drawing. The actual physical placement of the wires and components will be effected by the sticks processor. The designer just sketches his circuit on a graphical device. The distances between lines in the sketch is arbitrary. The order of the lines is important. The sticks processor eliminates the tedium of design rules with sophisticated spacing algorithms.

A designer specifies transistors, contacts and connectors symbolically. For example, an NMOS enhancement transistor is specified by a red line crossing a green. A contact is specified by a gray box over the colored lines for the type of contact. Only a minimal amount of data is needed to create a component.

Various people have developed methods for translating a symbolic representation to mask layout. Williams[Williams 80], Dunlop[Dunlop 79], and Hsueh[Hsueh 80] all use graph based methods for the purpose of compaction. These systems process full chips at all levels of the hierarchy, one level at a time. The stick drawing system is used for all cells in the hierarchy. At the higher level cells the system is used to solve the wiring problem. Buses and multi-conductors must be connected as individuals. This is an inappropriate use of sticks.

An interesting problem is automatic jog insertion. A jog is placed in a straight wire so that the wire is allowed to wrap around a piece of geometry. Jogs cost area. There are two bends in a jog wire which take up additional area. When jogs are used it is possible that the cell would be larger than one without jogs. In addition, introducing jogs in a symmetrical cell could destroy the symmetry and increase the size of the cell.

Although Hsueh uses macrocells for components in the drawing process their role is not clearly defined. The stretching of cells to fit an environment is done graphically by the users after compaction. This could be a laborious process if the cells were numerous or the chip was large. Each instance of an individual generic cell would

need to be stretched to its environment. This human interaction would be prohibitively expensive. In addition, if changes are accomplished to the chip the instances would also require human interaction. Changes to the chip should not require additional stretching to each instance. This is the responsibility of the composition tool.

Buchanan [Buchanan 80] pointed out that it is often desirable for the designer to have intimate control over some cells in the design. Most stick systems do not allow this type of control, and if they did it would require bypassing the geometry transformation routines. Buchanan states that the main problem with symbolic layout systems is "to provide the designer with sufficient flexibility without compromising the virtue of the design method".

2.2 Objectives

In a structured design style any design requires the generation of a small number of leaf cells, designed with more than a single constraint in mind. This tool is aimed at this class of cells. There are other design styles where cells are designed once and used in various chip design such as memories, output pads, input pad, and tristate pads. These cells are generally hand laid out with extreme care, design rule checked extensively and physically tested.

The editing process of the stick drawing should have few commands which are powerful and easy to learn. A VLSI designer who is familiar with graphics should need a only few minutes of instruction to start designing cells in REST. It should not be necessary for the designer to spend hours poring over extensive manuals but only a short time reading a simple users guide. Therefore the graphic editor should be a simple drawing system.

In any interactive environment and especially in graphics the response should not exceed approximately 20 seconds. This is because the attention of the designer would be taken from accomplishing the design to waiting for the system to respond. Therefore a fast response is necessary.

The compaction routines should have a response time not greater then a couple of

minutes. In addition there should be no hidden actions, that is the routines should do exactly what the user would expect.

3. THE MODELS

3.1 General

The models in a system directly relate to their usability and extensibility. REST uses two types of models. The first are the abstractions for the sticks, layout, and design rules. The second are REST's internal models.

The REST system follows the principle of isomorphism. The relationship between the various abstract models and the internal representation is isomorphic. This allows fast switching from stick editing to algorithm manipulation. The abstract stick representation is also isomorphic to the physical representation. This provides a consistent transformation from sticks representation to physical representation.

The abstract models in REST are currently set up for NMOS. The models are essentially table driven so that they can be changed for various processes. There are two parts to the tables. The first are a set of classes that define the primitive elements. The elements for NMOS are general contacts, buried contacts, butting contacts, enhancement and depletion transistors, and pullup devices. The second is the tables of valid wire layer, wire wides, affinity weights, and layer separations. For minor processes variations the second table may be modified. To change to another technology both sets of tables would need modification.

REST supports both butting and buried contact. The choice between the two is user control. REST allows switching from butting to buried on an existing cell where the design constraints are managed by REST.

The specific definition of the NMOS models used in REST are in "NMOS Sticks Standard Components"[Kahle 81]. The unit of measure used throughout REST and this document is lambda based on the Mead and Conway design rules [MEAD 80]. This unit allows the cells to be scaled for various process lines.

3.2 Abstract Model

There are four types of abstract models. The first is the stick model which the designer uses to sketch his circuit. The second is the displayed stick representation.

The third is the physical model for the final layout and proof plots. Each model is isomorphic to one another. The fourth is the design rules model for insuring a violation-free cell.

Each model is composed of either boxes or lines. These elements are restricted to being orthogonal. This restriction simplifies the compaction processes. Although this is a limitation it is not severe.

The input stick model is the representation that the user presents to the program to create a component. These models are sketched. The designer need only give a rough approximation of the model. The program will apply a pattern recognition routine to perceive the component.

The design rule model represents pseudo geometry that will be used in the compaction process. The pieces of the model consist of boxes that represent various mask layers or combinations of layers. Each box represents a fundamental design rule. They can easily be changed for various process lines. There is a table that records the set of minimum clearances for each box type.

In addition to representing a layer, the design rules box contains a unique identifier that specifies the net for the box. This is used in the compaction process to define the application of a specific rule.

The models are shown in their normal orientation. They can be rotated in 90 degree increments to point north, south, east, and west.

3.2.1 Wire Model

The basic building block in REST is the wire. A wire represents a connection between two components on a single layer. The wires' width is the default if left unspecified by the user. Buchanan[Buchanan 80] has shown two alternative methods for physical realization of a wire shown in figure 3-1. The first implementation as shown is the curtailed wire which is expanded on both sides of the wires path[3] by

---

[3]Path is a set of points(x,y) that define a start point, intermediate points, and an end point.

one-half its width and not over its endpoints. The second, the inflated wire is inflated from its path by one-half its wire width. This wire is the standard Caltech Intermediate Form(CIF)[MEAD 80, Sproull 79]. Problems with both types have been shown. The curtailed wire leaves holes at the endpoints as shown while the inflated wire overlaps at a tee connection. This overlap does not occur on standard width wires. The problem occurs only for wires with none standard width, that is, wide wires. In general either solution will cause problems.

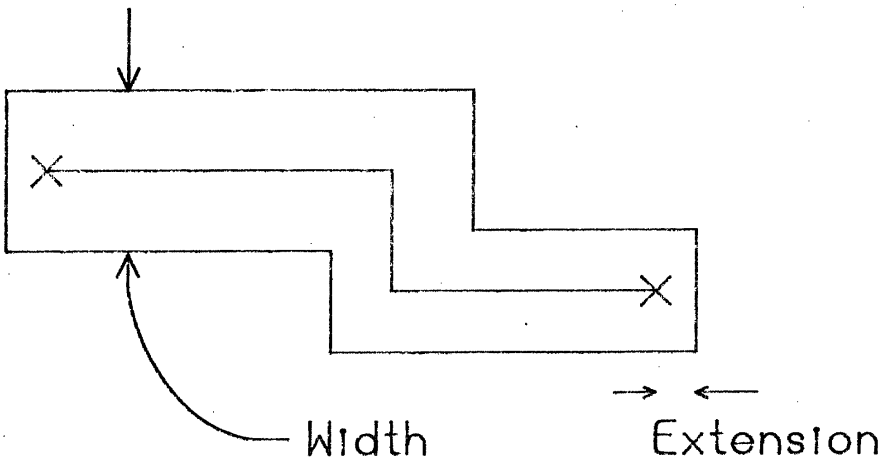


Figure 3-2: Wire Model

The physical realization for a wire in REST is shown in figure 3-2. The wire is inflated perpendicular to its path by the wire width. The ends are extended by the wire extension. The extension for wires is the default standard width. The problems caused by the two prior realization will not occur with this definition. Since the wire is only extended by it's standard width, the overlap on the tee connection will not occur.

The advantage of this model is that the path can be used for the definition of the wire with proper connection to other wires via the intersections of the end-points of the two paths defining the wires without worrying about wire widths. This definition will produce proper wires that will hold with design rules and with bloats

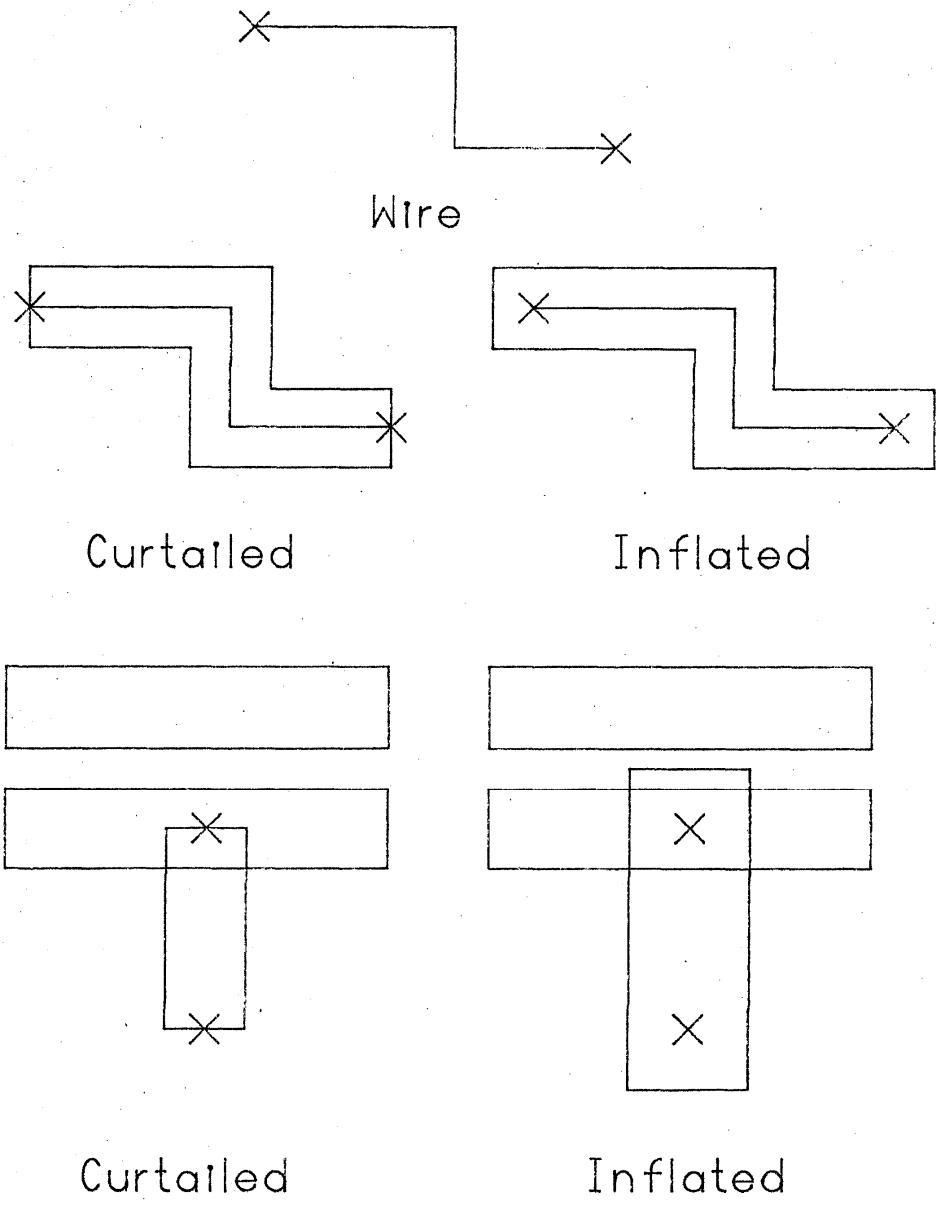


Figure 3-1: The Wire

and shrinks[4].

---

[4]Bloats and shrinks are geometric transformations that are applied to the layout for adjustment to process variations.

3.2.2 Transistors Model

The transistors are created by crossing two different colors. For the enhancement transistor the colors are red and green which represent polysilicon and diffusion respectively. In figure 3-3 the four models are shown. The enhancement transistor has a length and width associated with it. The length and width control the channel size. The models control the default sizes. The defaults sizes may be defined by the user. The normal defaults for the enhancement transistor is a length of two and a width of two. There are four connection points to the enhancement transistor labeled g1, g2 for polysilicon and source and drain for diffusion. These locations are in reference to the center point and extend with the length and width.

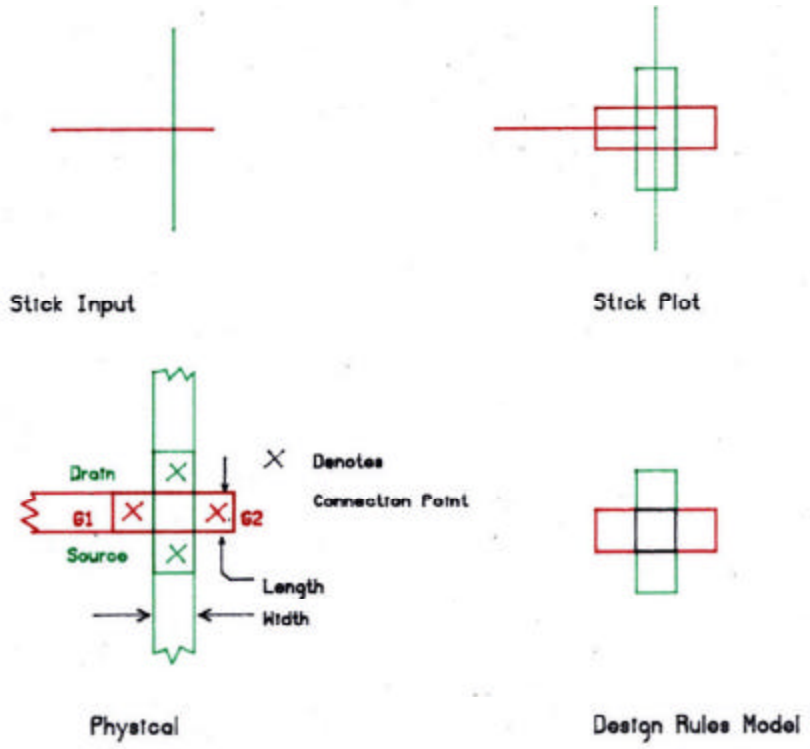


Figure 3-3: Enhancement Transistor

3.2.2 Transistors Model

The transistors are created by crossing two different colors. For the enhancement transistor the colors are red and green which represent polysilicon and diffusion respectively. In figure 3-3 the four models are shown. The enhancement transistor has a length and width associated with it. The length and width control the channel size. The models control the default sizes. The defaults sizes may be defined by the user. The normal defaults for the enhancement transistor is a length of two and a width of two. There are four connection points to the enhancement transistor labeled g1, g2 for polysilicon and source and drain for diffusion. These locations are in reference to the center point and extend with the length and width.

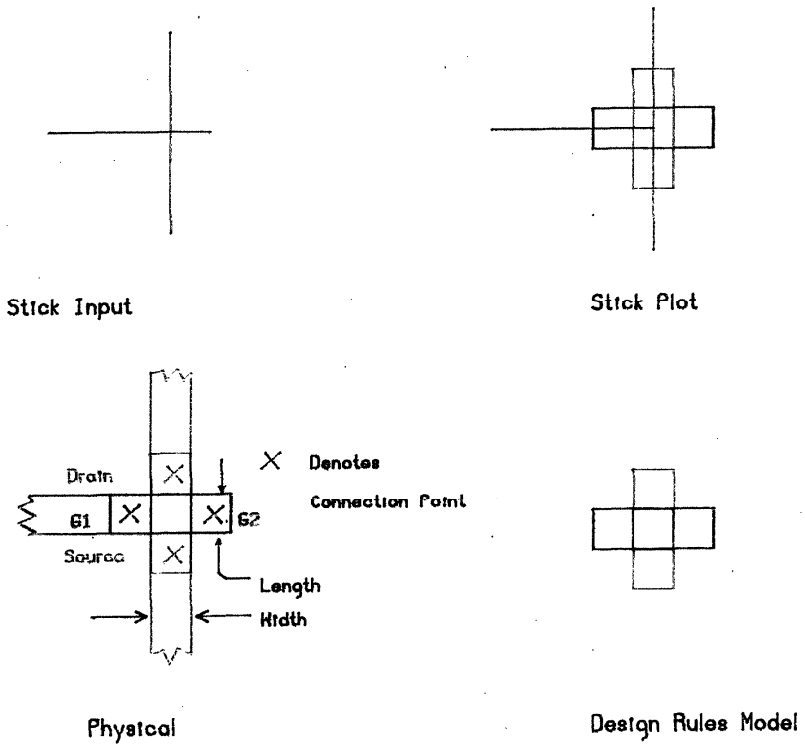


Figure 3-3: Enhancement Transistor

The design rules model for the enhancement transistor has two diffusion boxes above and below the gate as shown in figure 3-3. Each diffusion box is connected to the appropriate net. The polysilicon is a single box where the length is the defined length for the model and the width is the defined width plus the two lambda overlaps on either side of the device. There is a center box of type gate which represents the active area.

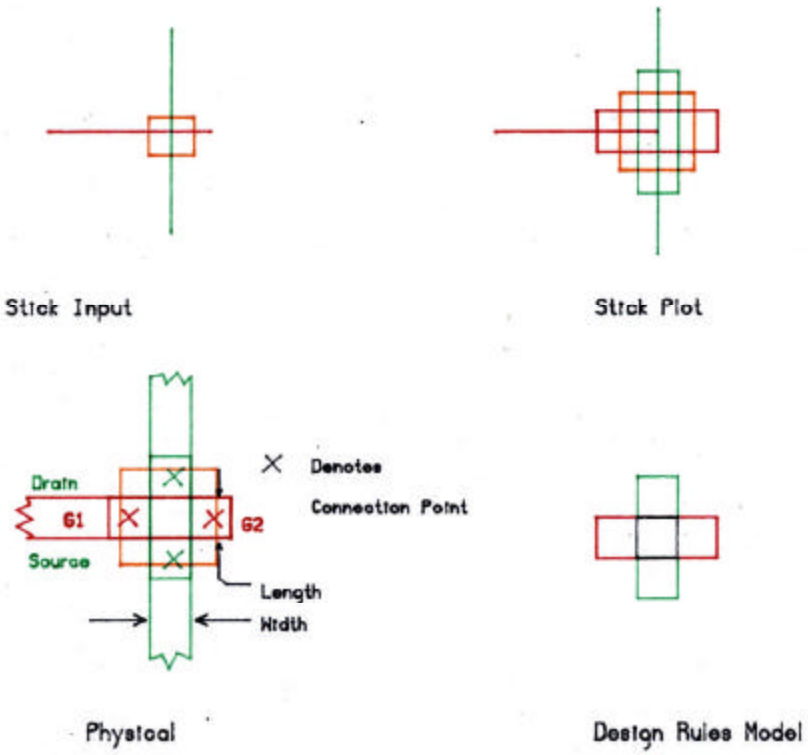


Figure 3-4: Depletion Transistor

The representation for the depletion transistor is the same as the enhancement except for the addition of the implant. The depletion transistor is shown in figure 3-4.

The pullup or resistive type device is shown in figure 3-5. The pullup device is created by placing a red line upon a green with a contact box at one end, and a yellow

The design rules model for the enhancement transistor has two diffusion boxes above and below the gate as shown in figure 3-3. Each diffusion box is connected to the appropriate net. The polysilicon is a single box where the length is the defined length for the model and the width is the defined width plus the two lambda overlaps on either side of the device. There is a center box of type gate which represents the active area.

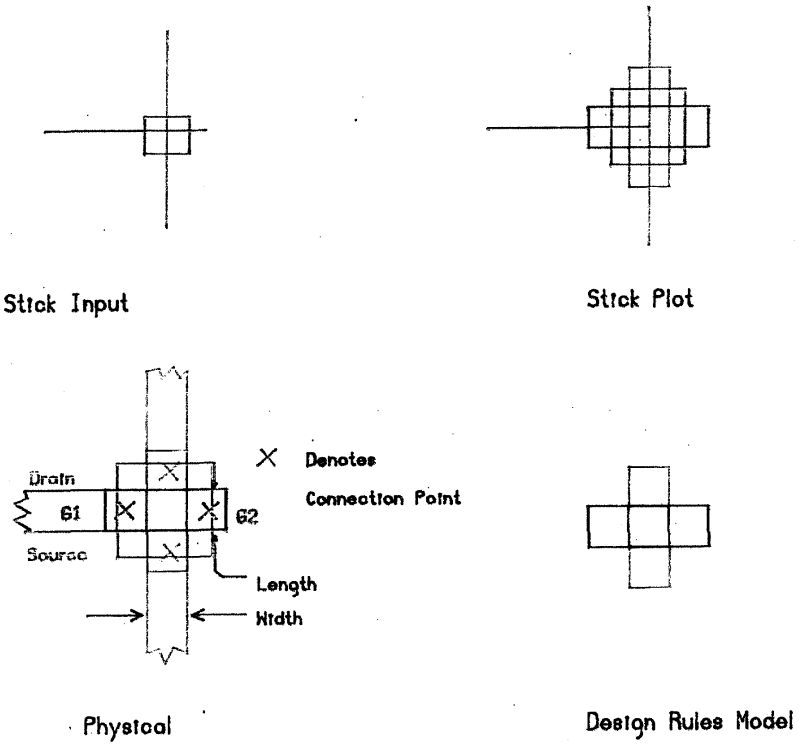


Figure 3-4: Depletion Transistor

The representation for the depletion transistor is the same as the enhancement except for the addition of the implant. The depletion transistor is shown in figure 3-4.

The pullup or resistive type device is shown in figure 3-5. The pullup device is created by placing a red line upon a green with a contact box at one end, and a yellow

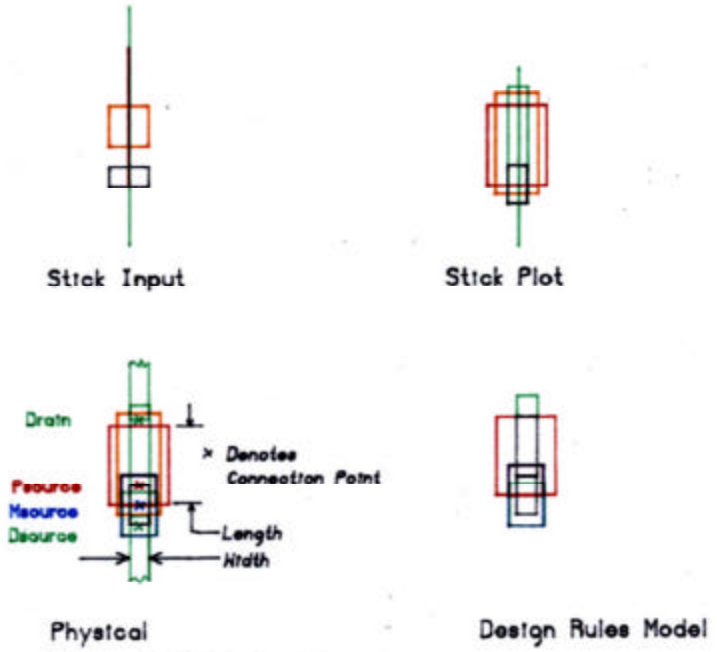


Figure 3-5: Pullup Transistor

box intersecting the red and green lines. The pullup device has a length and width associated with it. As with the prior transistors the user may alter the defaults. The normal defaults are a length of 8 and a width of 2. This default corresponds to a normal inverter with a ratio of 4 to 1. There are four connection points to the pullup device label dsource and drain for diffusion, psource for polysilicon, and msource for metal. These locations are in reference to the center point and extend with the length and width.

The design rules model consists of five boxes: two diffusion boxes (one at the bottom and the other at the top), a gate box which is the active area, a polysilicon box, and a metal box.

3.2.3 Contacts Model

The contacts are shown in figures 3-6 and 3-7. The stick model for the contacts are the same. That is, two colored lines covered by a box. The polysilicon/metal contact

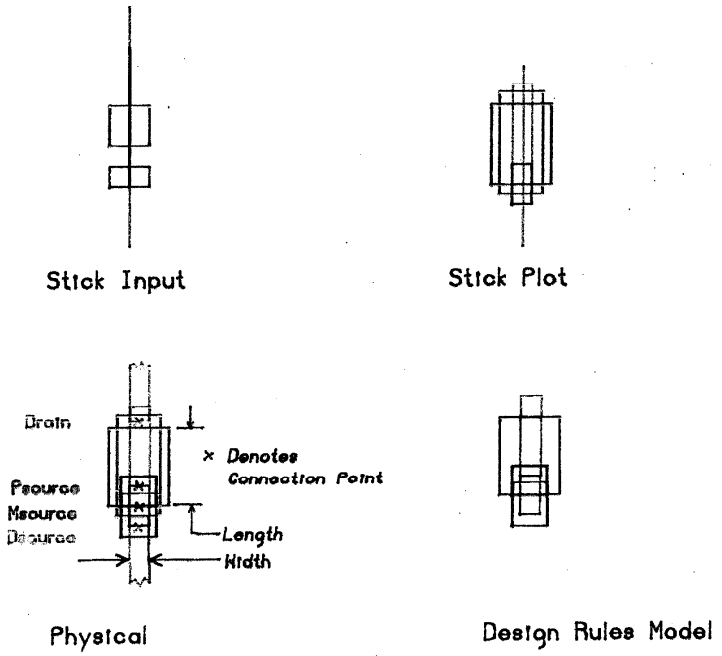


Figure 3-5: Pullup Transistor

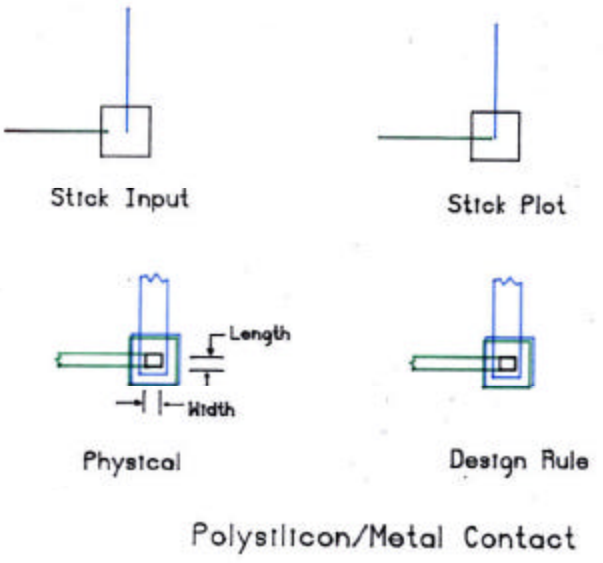
box intersecting the red and green lines. The pullup device has a length and width associated with it. As with the prior transistors the user may alter the defaults. The normal defaults are a length of 8 and a width of 2. This default corresponds to a normal inverter with a ratio of 4 to 1. There are four connection points to the pullup device label dsource and drain for diffusion, psource for polysilicon, and msource for metal. These locations are in reference to the center point and extend with the length and width.

The design rules model consists of five boxes: two diffusion boxes (one at the bottom and the other at the top), a gate box which is the active area, a polysilicon box, and a metal box.

3.2.3 Contacts Model

The contacts are shown in figures 3-6 and 3-7. The stick model for the contacts are the same. That is, two colored lines covered by a box. The polysilicon/metal contact

and the diffusion/metal contact are essentially the same. The width and length define the size of the contact. The cut area is maintained constant. For larger widths and length, the cut is kept constant size and replicated modulo the length and width. The connection point is the center of the contact.



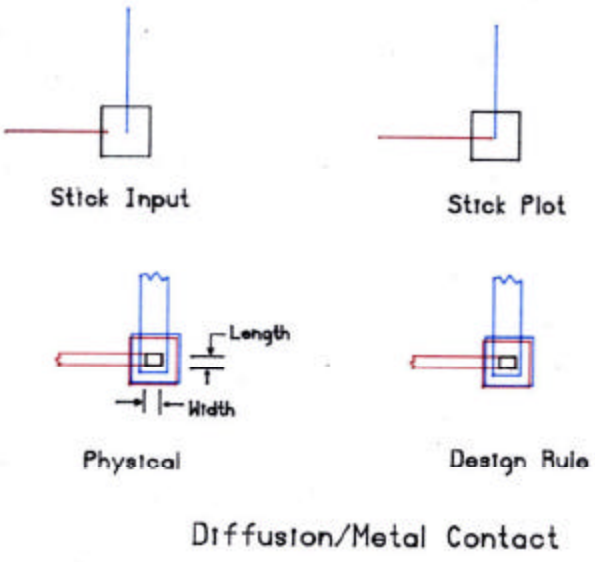


Figure 3-6: Metal Contacts

and the diffusion/metal contact are essentially the same. The width and length define the size of the contact. The cut area is maintained constant. For larger widths and length, the cut is kept constant size and replicated modulo the length and width. The connection point is the center of the contact.

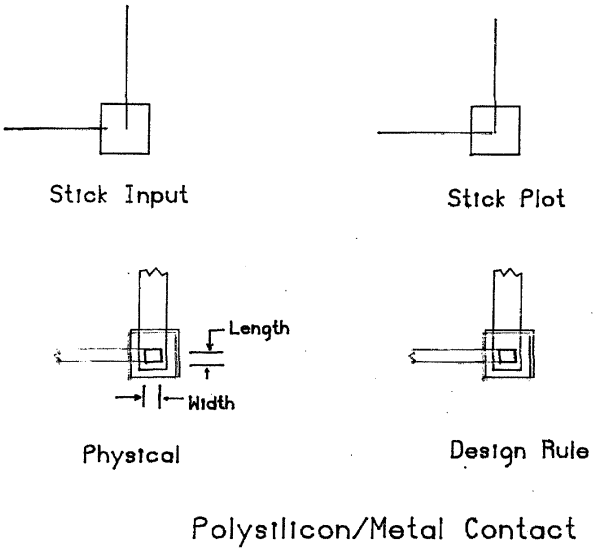


Figure 3-6: Metal Contacts

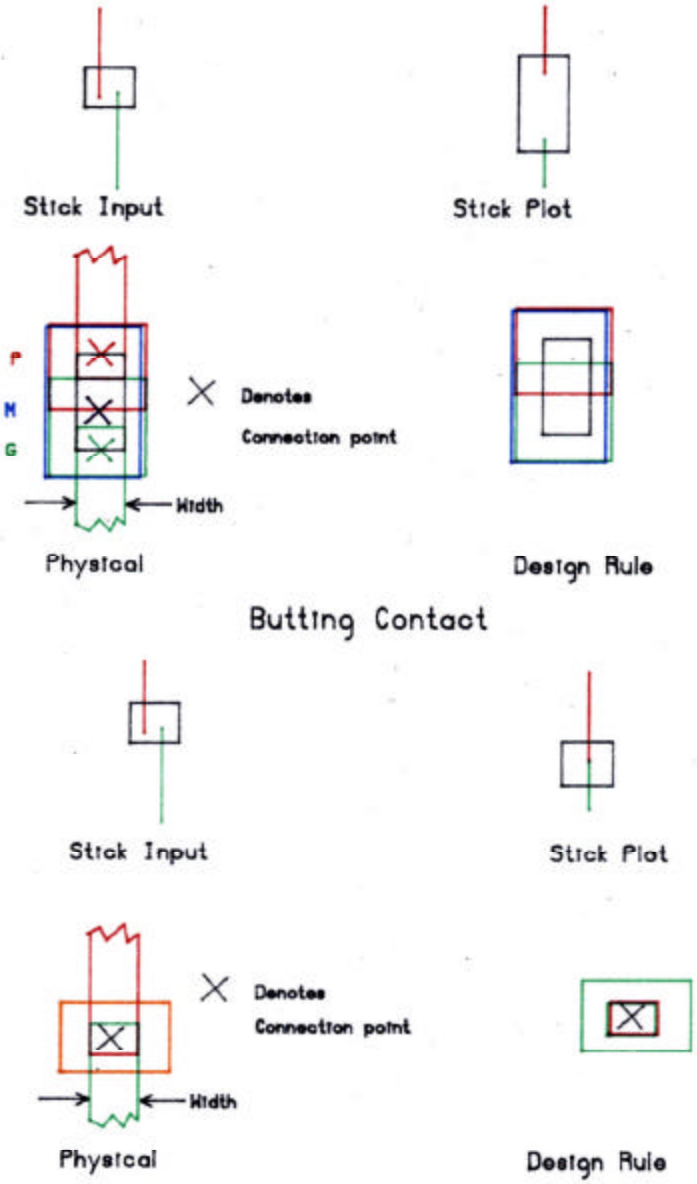


Figure 3-7: Polysilicon to Diffusion Contacts

The butting contact is shown in figure 3-7. The butting only has a width associated with it. The connection points are labeled m for metal, p for polysilicon, and d for diffusion.

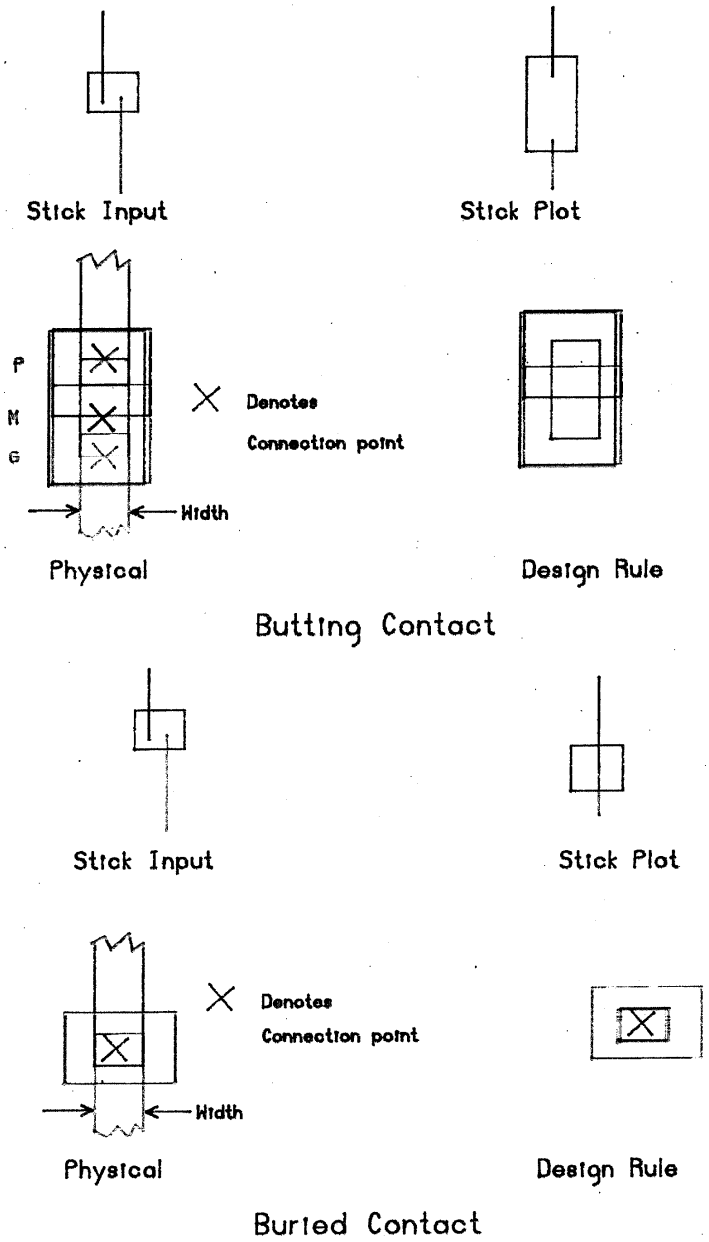


Figure 3-7: Polysilicon to Diffusion Contacts

The butting contact is shown in figure 3-7. The butting only has a width associated with it. The connection points are labeled m for metal, p for polysilicon, and d for diffusion.

3.2.4 Cell Model

The cell is modeled as rectangle with the ports as fingers extending outside of the rectangle[5]. The dimension of the rectangle called the abuting box is defined so that two cells can be aligned on the edges of the boxes without a design rules error, provided the ports are compatible. This spacing is maintained by the compactor.

3.3 Internal Model

The internal models are the data representation for the abstract models. The most basic type is the joint as shown in figure 3-8. A joint is a location with a color. The joint represents a point on the path of a wire. The data attributes of a point are the coordinate x,y, the color, the name, and four pointers. The pointers refer to the top, bottom, left and right segments. All but one of the points of a joint may be nil.

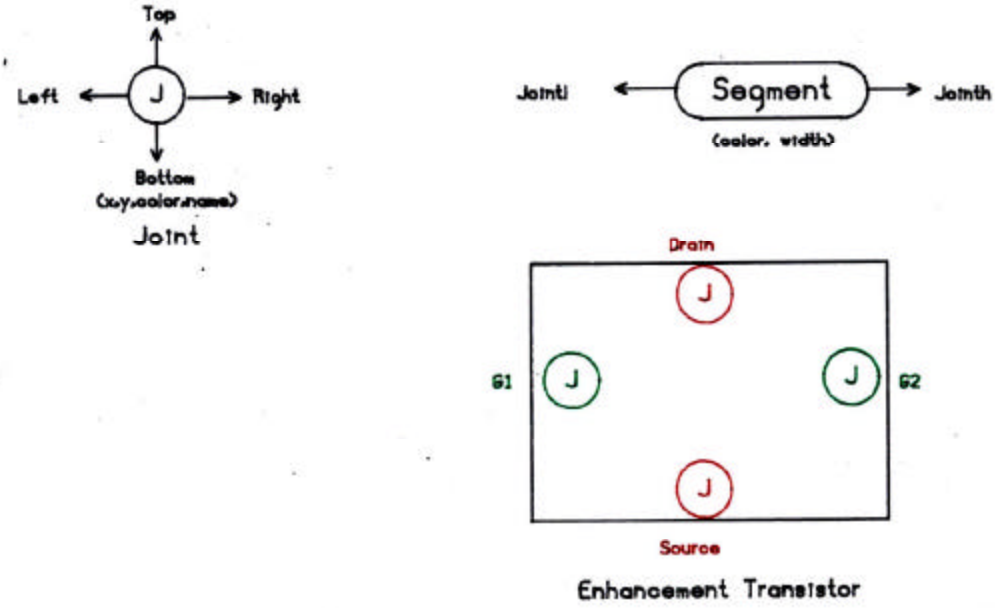


Figure 3-8: Abstract Models

A segment is a fragment of a wire where a bend may occur. Each segment carries its

---

[5] See figure 1-2 in section 1 page 2 for a graphic representation of the cell model.

3.2.4 Cell Model

The cell is modeled as rectangle with the ports as fingers extending outside of the rectangle[5]. The dimension of the rectangle called the abuting box is defined so that two cells can be aligned on the edges of the boxes without a design rules error, provided the ports are compatible. This spacing is maintained by the compactor.

3.3 Internal Model

The internal models are the data representation for the abstract models. The most basic type is the joint as shown in figure 3-8. A joint is a location with a color. The joint represents a point on the path of a wire. The data attributes of a point are the coordinate x,y, the color, the name, and four pointers. The pointers refer to the top, bottom, left and right segments. All but one of the points of a joint may be nil.

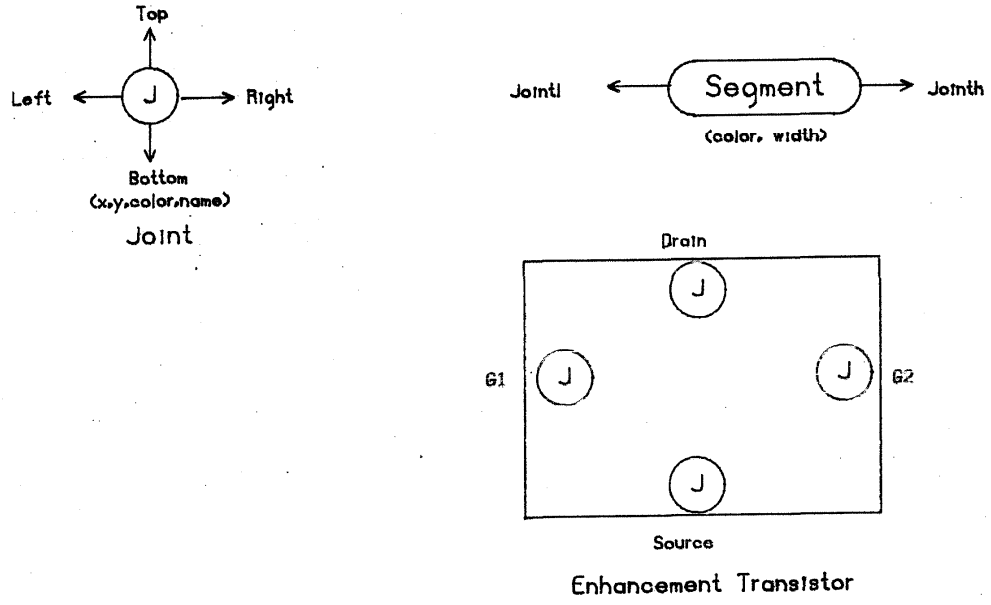


Figure 3-8: Abstract Models

A segment is a fragment of a wire where a bend may occur. Each segment carries its

---

[5]See figure 1-2 in section 1 page 2 for a graphic representation of the cell model.

color and width. A segment has two end pointers jointl and jointh which refer to the terminals of the segment. The jointl always has the lowest coordinate of the two joint references. There are no coordinates associated with the segment. This allows movement of a joint with the attached segments in a easy fashion.

A wire is modeled as a set of joints and segments. Each segment of the wire carries the width of the wire. Two wires of the same color intersect at a common joint. One way to view the relation of segments and joints is a lattice. This lattice is varied for processing in a computational environment.

A component is modeled as a data object with attributes that are a set of joints, a name, and parameters. An example of an enhancement transistor is shown in figure 3-8. The transistor has four references g1, g2, source, and drain. These pointers refer to all the possible connection points of the device. The enhancement transistor also carries a length and width. Displacing any component also moves all referenced joints and in turn moves all attached segments.

Each component is an object with a data part and code part. The data consists of the various pointers and component parameters. The code parts contain the routines to generate the various abstract model types on demand. During the compaction process a request would be effected for the design rule model.

The joints, segments, components, and ports are stored internally as a set of ordered lists. These lists are easily manipulated for various processes.

4. DISTRIBUTION OF TASKS

4.1 General

The topological, structural and geometric data for leaf cells is captured by the REST system. The input is a simple colored sketch of lines and boxes representing a circuit. This colored sketch is digested by the REST process to produce a compacted sticks representation. These two functions lead to a natural partitioning of tasks into two parts: 1) The line and box diagramming system that run in a local color graphics station and 2) the main program that does the major processing. This system is shown in figure 4-1. At the present time, REST permits only orthogonal line drawings.

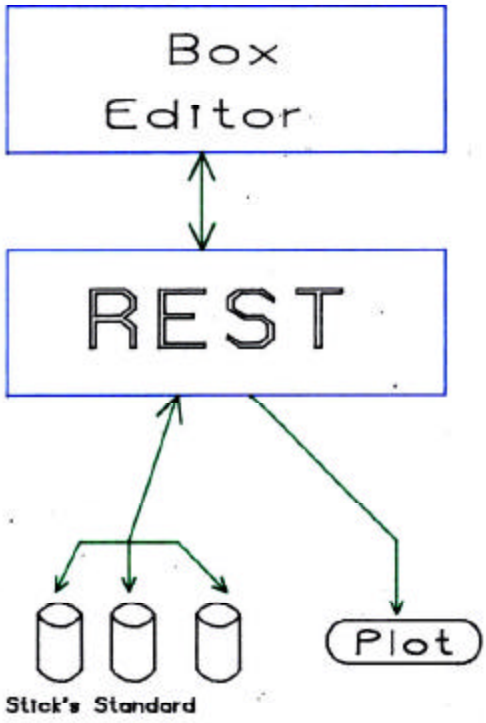


Figure 4-1: Rest System Block Diagram

The advantages of separating the graphical editing function from the main program is twofold. First, graphic editing requires fast computer response in order for the

4. DISTRIBUTION OF TASKS

4.1 General

The topological, structural and geometric data for leaf cells is captured by the REST system. The input is a simple colored sketch of lines and boxes representing a circuit. This colored sketch is digested by the REST process to produce a compacted sticks representation. These two functions lead to a natural partitioning of tasks into two parts: 1) The line and box diagramming system that run in a local color graphics station and 2) the main program that does the major processing. This system is shown in figure 4-1. At the present time, REST permits only orthogonal line drawings.

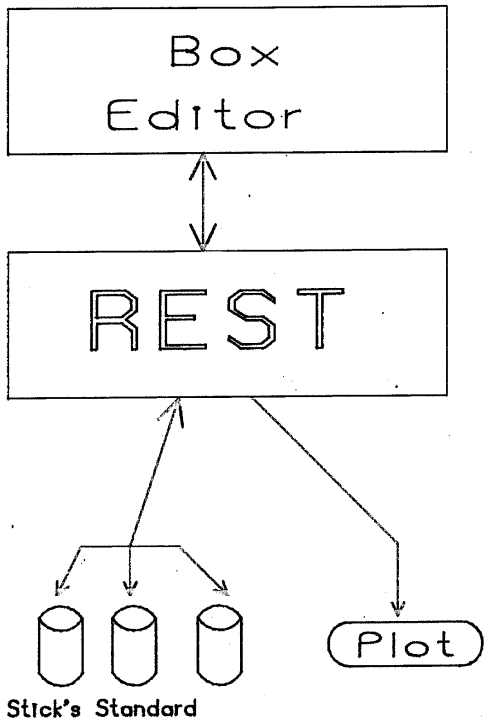


Figure 4-1: Rest System Block Diagram

The advantages of separating the graphical editing function from the main program is twofold. First, graphic editing requires fast computer response in order for the

designer to achieve true interactive drawing capabilities. This fast response is accomplished by using a dedicated local processor for the graphic editing. Second, many different graphic work stations with their own local editors may be used. REST can currently interface two graphic work stations, which are : 1) the Charles terminal and 2) the DEC GIGI. This partitioning is consistent with the objectives.

The diagramming system known as the box editor runs in a color graphics display station called the Charles graphic station[Burke 80, Minter 80]. This station is composed of a video monitor, a DEC LSI-11 processor, a frame buffer, a DEC VT52 terminal, and a Hewlett Packard four pen color plotter. There are 4 bits per pixel on the color monitor which are memory mapped into the address space of the LSI 11. This work station is linked to a host computer via a data communication line at 9600 baud. The graphical entry is via a three button mouse. This system is shown in the hardware organization figure 4-2.

The principles in the diagram system is to limit its scope to simple commands that can be easily learned in a few minutes and to allow the user to just sketch his stick drawing without worrying about design rules or constructing an exact drawing. The geometry shapes are limited to lines and boxes and the commands are limited to create, move and delete.

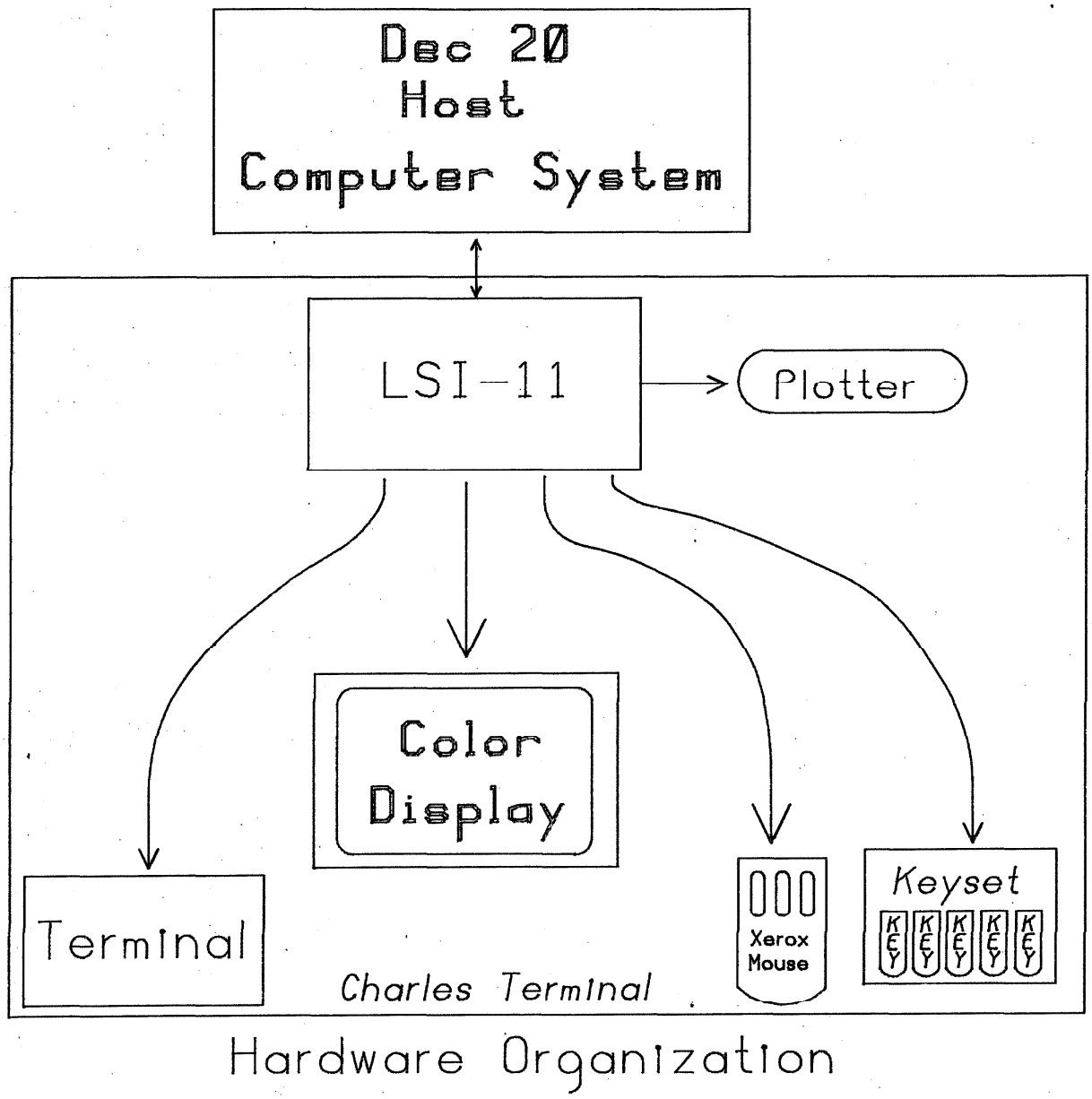


Figure 4-2: Hardware Organization

4.2 Stick Diagram Preparation

The commands to the box editor were kept simple and small in number to provide a good human interface. The box editor has three commands which map directly to the mouse buttons, and two modes. The commands are: 1) line/box add, 2) line/box delete and 3) line/box move. The modes are line and box. Color selection, mode selection, and termination are accomplished via a menu as shown in figure 4-3. This provides an easily remembered set of commands and a reasonable human interface.

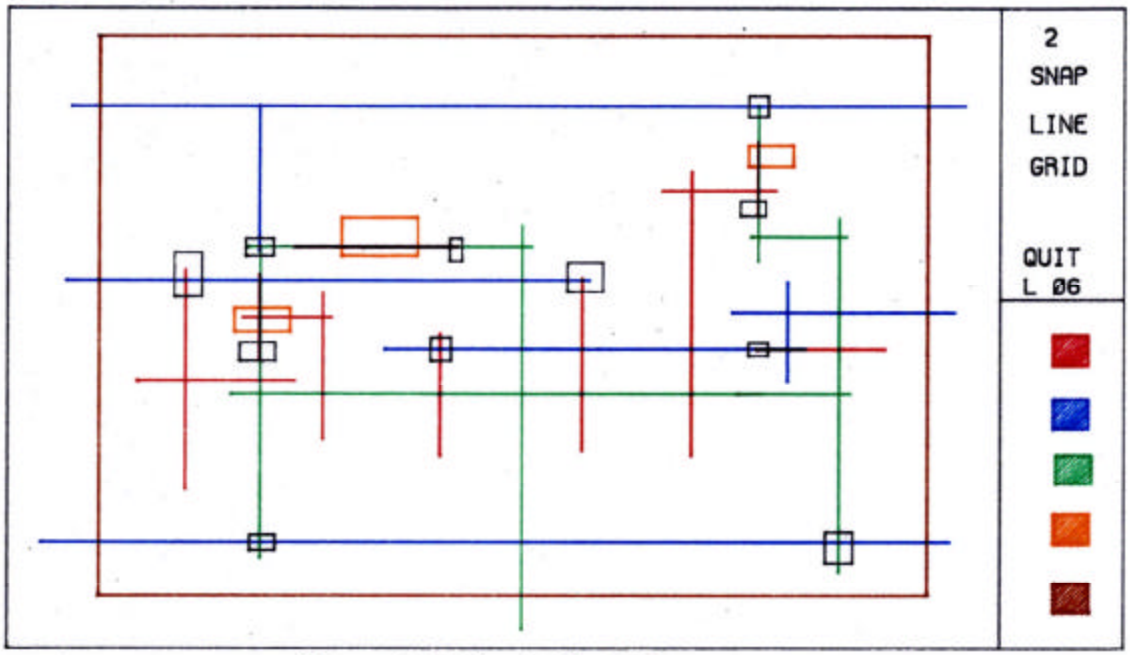


Figure 4-3: Simple Sketch Input

A sample input sketch for REST is shown in figure 4-3. The important characteristic of the sketch is that it is a rough drawing. There is no need for the designer to be neat in the sketch since the REST processor will recognize connection points as any two similarly colored lines which are in close proximity, transistors as red lines crossing green, and contacts as gray boxes over colored lines. A designer generally would sketch out his sticks drawing on the Charles terminal as if he were doing his stick drawing on paper. However, the lines are easier to erase on the screen than they are on paper. Blue lines are recognized as metal, red lines are recognized as polysilicon, and green lines are recognized as diffusion. The meaning of the colors are

4.2 Stick Diagram Preparation

The commands to the box editor were kept simple and small in number to provide a good human interface. The box editor has three commands which map directly to the mouse buttons, and two modes. The commands are: 1) line/box add, 2) line/box delete and 3) line/box move. The modes are line and box. Color selection, mode selection, and termination are accomplished via a menu as shown in figure 4-3. This provides an easily remembered set of commands and a reasonable human interface.

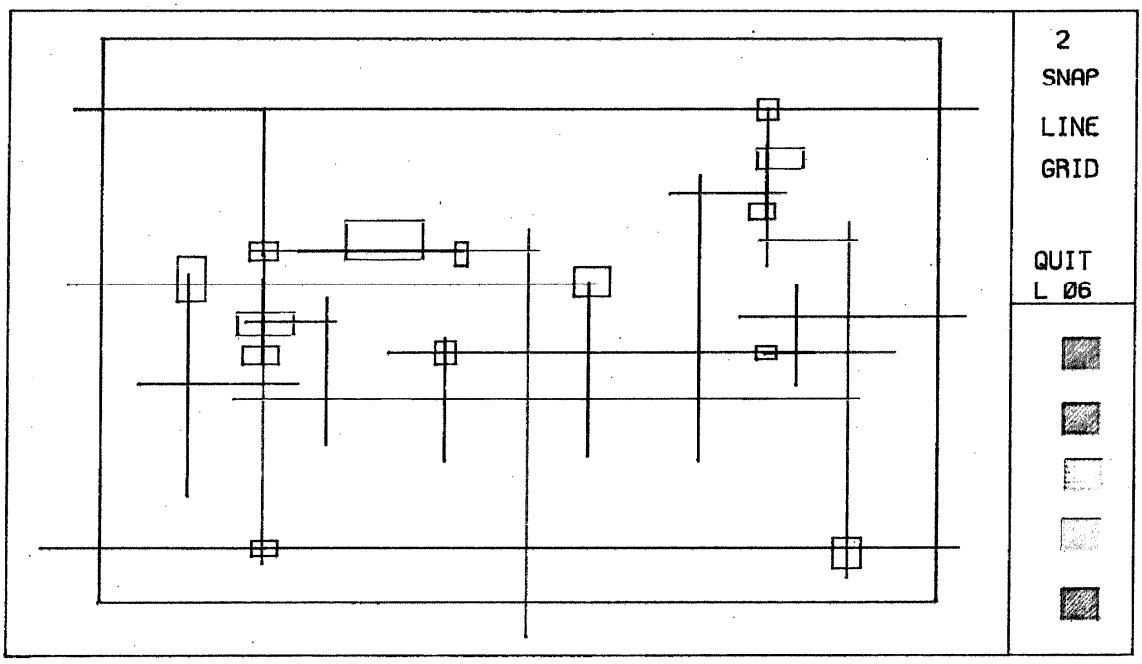


Figure 4-3: Simple Sketch Input

A sample input sketch for REST is shown in figure 4-3. The important characteristic of the sketch is that it is a rough drawing. There is no need for the designer to be neat in the sketch since the REST processor will recognize connection points as any two similarly colored lines which are in close proximity, transistors as red lines crossing green, and contacts as gray boxes over colored lines. A designer generally would sketch out his sticks drawing on the Charles terminal as if he were doing his stick drawing on paper. However, the lines are easier to erase on the screen than they are on paper. Blue lines are recognized as metal, red lines are recognized as polysilicon, and green lines are recognized as diffusion. The meaning of the colors are

consistent with Mead and Conway [MEAD 80]. Internally, these colors are derived from a table so the meaning may be changed easily for color preference or for different technologies.

4.3 Stick Diagram Processing

The interpretation of the sticks drawing is done by the main program of REST which runs on the host computer. The line drawing interpreter analyzes the crude line drawing for connections between lines, simple devices, connection points, and contacts. In figure 4-3, an enhancement device is recognized as red crossing green, a depletion device is recognized as red crossing green with a yellow box on top and a pullup is recognized as a red line on top of a green with a grey box at one end of the red and a yellow box in close proximity. Contacts are recognized as grey boxes over lines. The device ratios and line widths are taken from a user defined table. In addition, the line drawing interpreter correlates the newly interpreted drawing with the previous one, if it exists. This operation is done so that old component names, device ratios, and line widths can be transferred to the new sticks representation for the cell.

The drawing is surrounded by a bounding rectangle called the bounding box. This box defines the scope of the cell. Any line in the drawing that crosses this boundary will be considered a connection point for the cell. Any line or box outside of this region will be discarded. The size of the box is determined by the elements of the cell. The user has the option to increase the size by various controls. The user can only decrease the bounding box by a compaction process.

The recognition of the sketch in figure 4-3 is shown in figure 4-4. This stick cell is shown in the standard sticks diagram form that REST generates. In fact, this drawing was plotted with the attached Hewlett Packard 7221A plotter. In the drawing, lines that were close and of like colors were snapped together. Lines that overlapped were trimmed off. The drawing shows transistors fleshed out as boxes with stylized lines representing the wires. The sizes of the transistors are proportional to their ratios.

The interpreted cell in figure 4-4 is shown compacted in physical form in figure 4-5. Since the sticks representations are isomorphic, it could just as easily have been shown in sticks form. The figure is shown to scale in relation to the uncompacted cell.

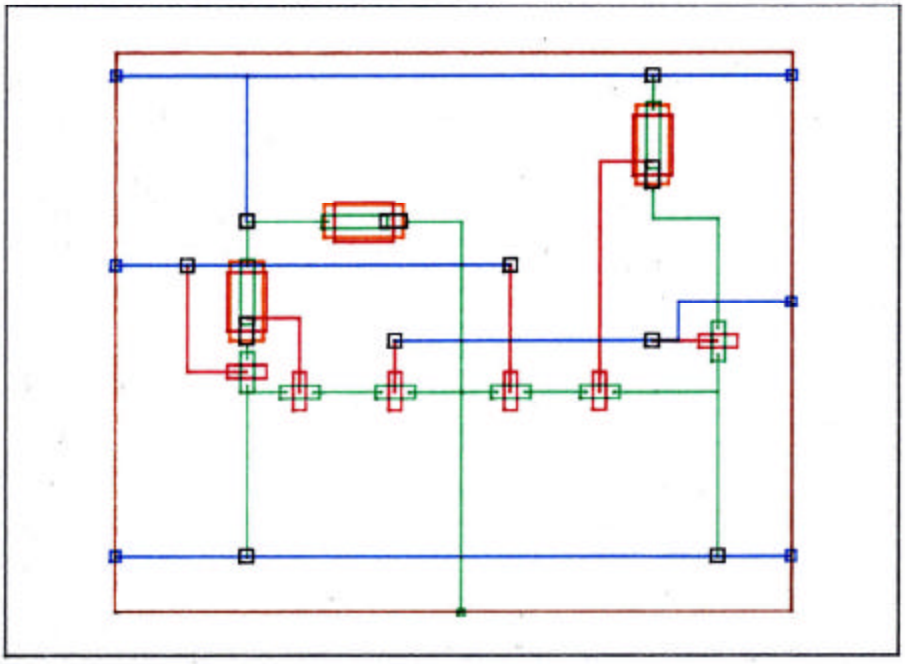


Figure 4-4: Interpreted Cell

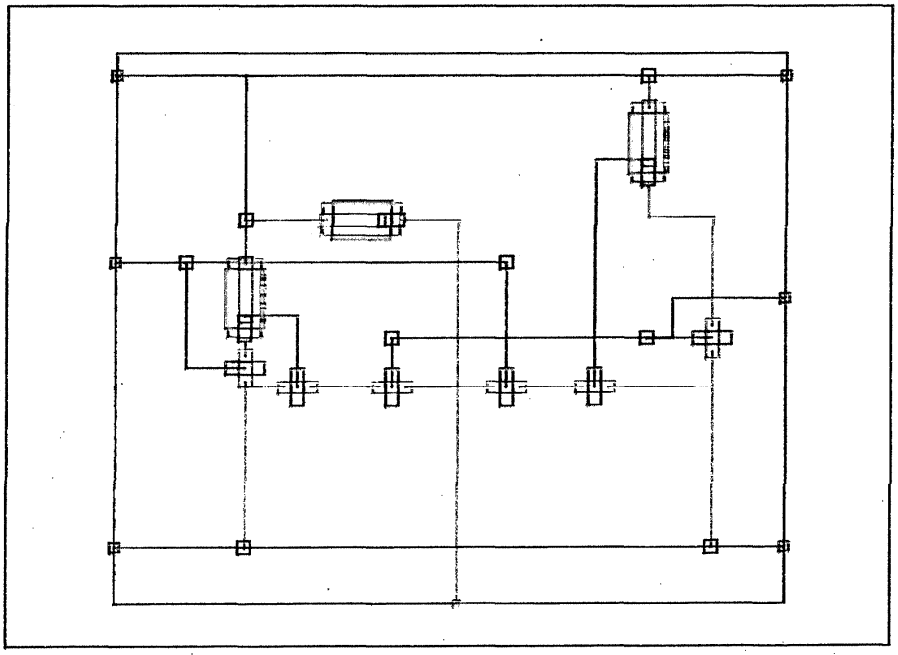


Figure 4-4: Interpreted Cell

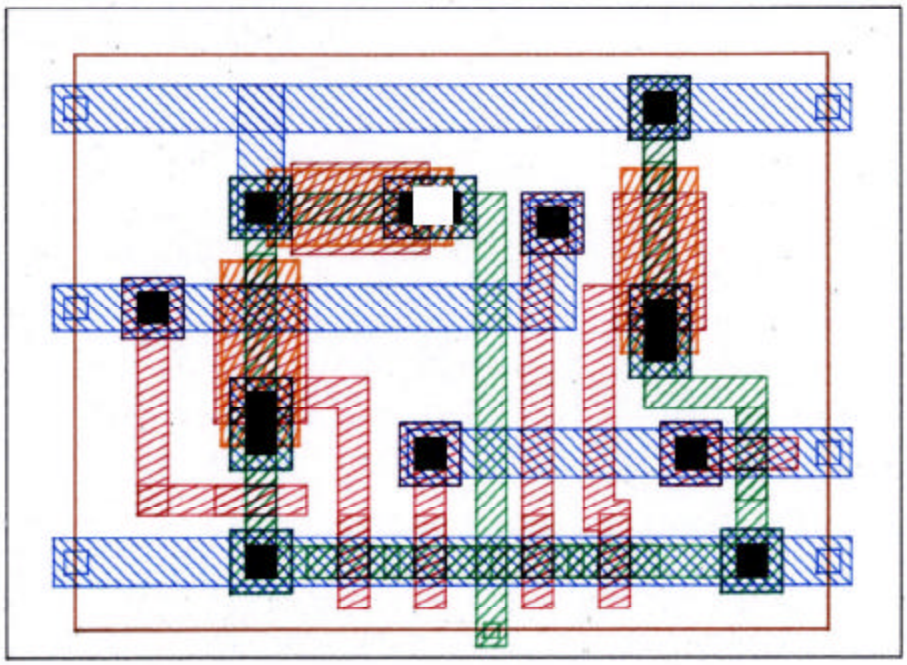


Figure 4-5: Compacted Cell

Compaction in REST is accomplished by compression in the vertical axis followed by compression in the horizontal axis. This order can be user controlled. REST allows a line to jog in either direction at wire intersection points. Topological features are allowed to cross provided they are not constrained by design rules. Constraints defined by the users are also allowed. REST employs a table driven compaction algorithm using a table that can easily be changed for various design rules. Currently REST uses the design rules in Mead and Conway [MEAD 80].

A point that is not well known is that of line jogs. In a few compactors lines that are connected and jog are not allowed to cross but only merge [Williams 80]. This is shown in figure 4-6. In A the line only merges while in B the line is allowed to cross. The reason that lines are only allowed to merge is in the compactor proper which will be discussed in a later section. REST allows lines to cross as in B.

Unlike other graph compaction methods, REST contains a weighted affinity factor in

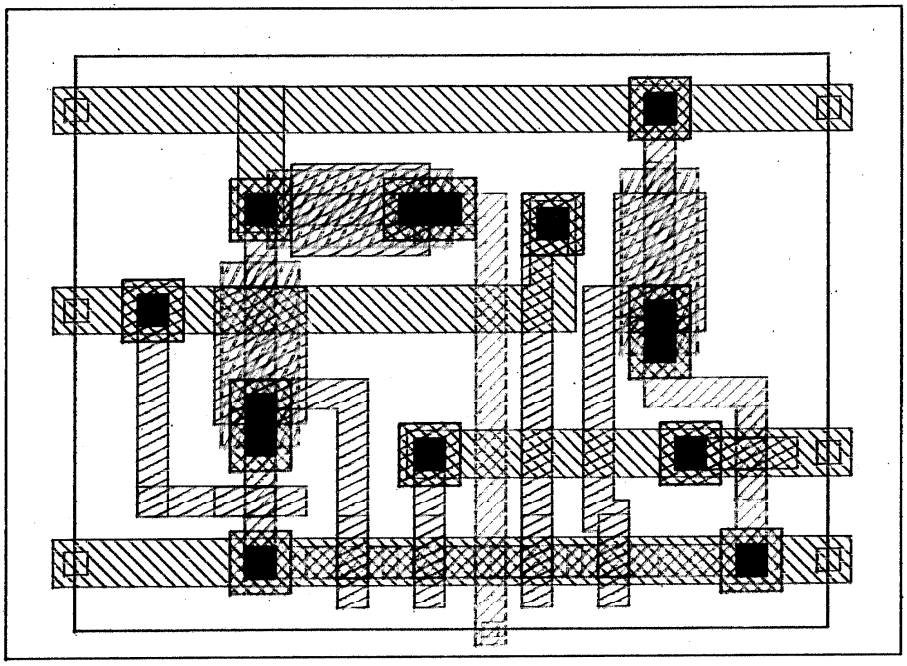


Figure 4-5: Compacted Cell

Compaction in REST is accomplished by compression in the vertical axis followed by compression in the horizontal axis. This order can be user controlled. REST allows a line to jog in either direction at wire intersection points. Topological features are allowed to cross provided they are not constrained by design rules. Constraints defined by the users are also allowed. REST employs a table driven compaction algorithm using a table that can easily be changed for various design rules. Currently REST uses the design rules in Mead and Conway [MEAD 80].

A point that is notwellknownis that of line jogs. In a few compactors lines that are connected and jog are not allowed to cross but only merge[Williams 80]. This is shown in figure 4-6. In A the line only merges while in B the line is allowed to cross. The reason that lines are only allowed to merge is in the compactor proper which will be discussed in a later section. REST allows lines to cross as in B.

Unlike other graph compaction methods, REST contains a weighted affinity factor in

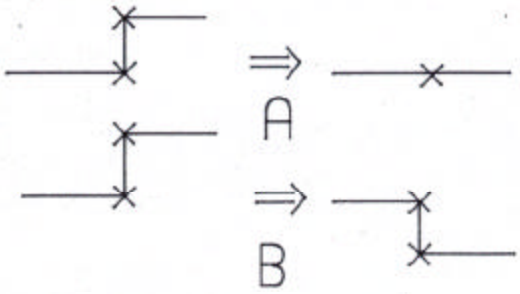


Figure 4-6: Line Jogging

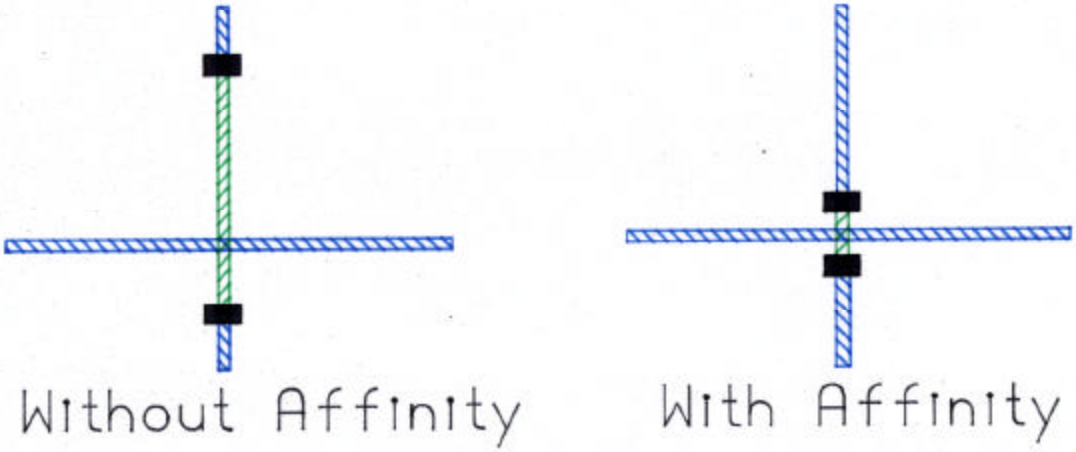


Figure 4-7: Affinity Factor

each branch. This factor is the amount of attractiveness between the two attached nodes. This factor is best explained by an example as shown in figure 4-7. The example cell has a metal wire running through the cell with a second metal line orthogonal to it and a diffusion strap crossing under the first metal wire. If compacted without an affinity factor, the diffusion strap will be elongated toward the edge of compacted cell. That is, the contact that connects the metal wires to the diffusion strap will fall to the edge when it is compacted or in some programs it would be centered between the first metal wire and edge of the cell. In REST, the elongation of the diffusion strap would not occur due to the use of the affinity factor. The diffusion wire would be kept as short as design rules permit. The branch for the diffusion strap

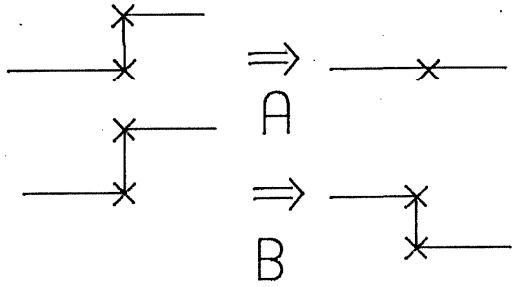


Figure 4-6: Line Jogging

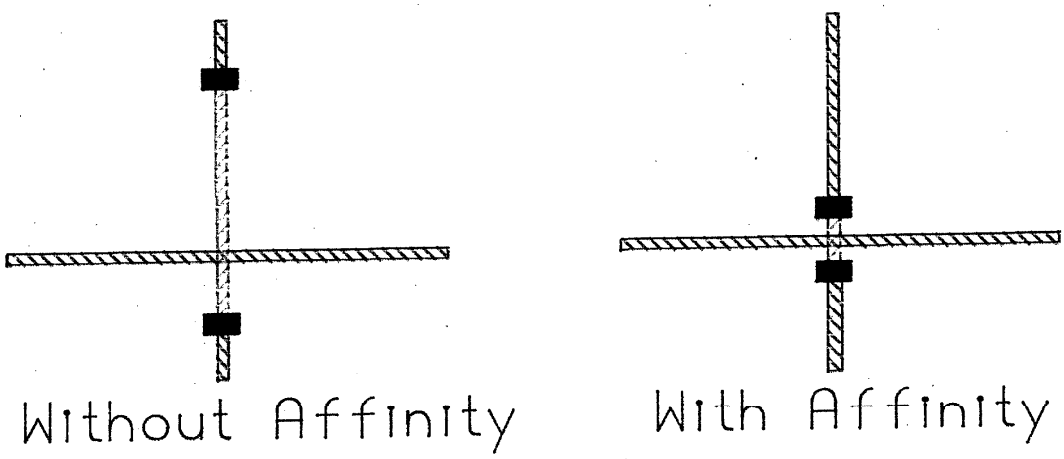


Figure 4-7: Affinity Factor

each branch. This factor is the amount of attractiveness between the two attached nodes. This factor is best explained by an example as shown in figure 4-7. The example cell has a metal wire running through the cell with a second metal line orthogonal to it and a diffusion strap crossing under the first metal wire. If compacted without an affinity factor, the diffusion strap will be elongated toward the edge of compacted cell. That is, the contact that connects the metal wires to the diffusion strap will fall to the edge when it is compacted or in some programs it would be centered between the first metal wire and edge of the cell. In REST, the elongation of the diffusion strap would not occur due to the use of the affinity factor. The diffusion wire would be kept as short as design rules permit. The branch for the diffusion strap

would have a higher affinity than the metal line from the contact to the edge. The affinity in REST is table driven based on line type and line width. That is, the polysilicon and diffusion lines have greater affinity than metal. This factor can be altered based on design rules. The affinity factor controls the trade off in line length between the metal, diffusion and polysilicon wires.

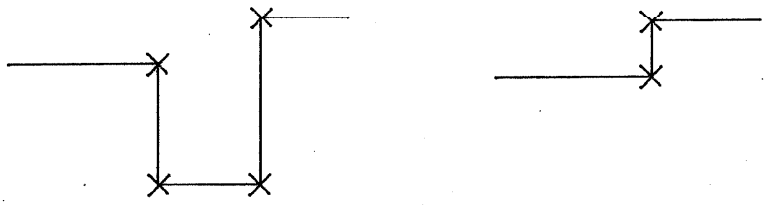


Figure 4-8: Loops

Loops in wires will be removed in REST in the compaction process as a direct result of the affinity factor. A loop is shown in figure 4-8.

The designer has complete control over the compaction process. The compactor only compresses the cell. Topological changes of related objects are not performed. Rotation of elements and jog insertion are effected by the designer. The designer can designate the direction of compaction, insert jog points, and add constraints.

Generally after the initial compaction the cell is not as small as it could be. This is because the compactor does not consider overall scope. This is the responsibility of the designer, who does it best. A designer would generally design his cell and then perform several compaction and edit sessions. The initial set of compaction sessions will produce a reasonable amount of reduction. Further reduction of the cell will require much more additional time for only a little decrease in size. It has been found that after several editings and compactions the cell is at a reasonable density.

The user has several ways to control the compaction. First the designer can direct the axis of compaction and second direct the side which to compress. It has been found

that just by expanding the cell and trying the two alternatives, the cell can be considerably reduced.

Another way the user can improve the compactness of his cell is by introducing jog points. A jog point is a position on a wire where the wire is allowed to bend. This allows unused space to be filled. Jog points cost area and should be used with caution.

The designer can impose constraints on a cell. The constraint is in a general form of: component1+value <= component2 <=component1+value. REST provides an easy means to set the constraints. Please see the users guide for specifics. Constraints allow tailoring a cell for a specific environment. An example would be setting all the ports on the left equal to the ports on the right. This allows the example cell to be easily stepped across in a row.

When additions or changes to a cell are to be effected, the designer has several modes of expansion that can be employed. The cell can be expanded in general along either of the two axis. This general expansion separates each group by an additional amount specified by the designer. The abutment box can be stretched or pulled out at the left, right, top, or bottom. This stretching will also lengthen any wires that are attached to ports on the side that is stretched. In addition the designer can identify a point around which expansion will take place as shown in figure 4-9. The amount of expansion is defined by the user. This point type expansion is very helpful for cells that have been compressed and when additional logic needs to be added.

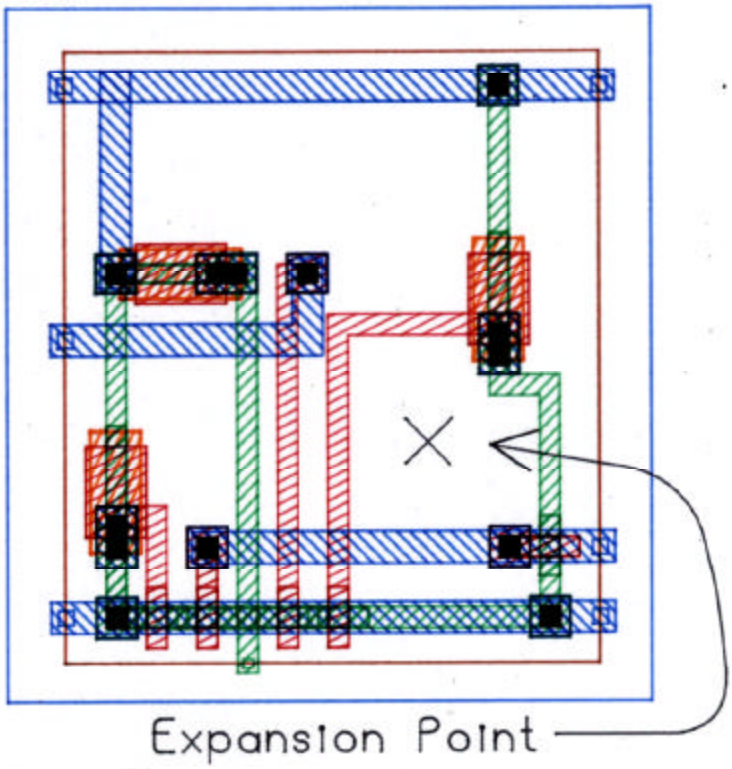


Figure 4-9: Expanded Cell

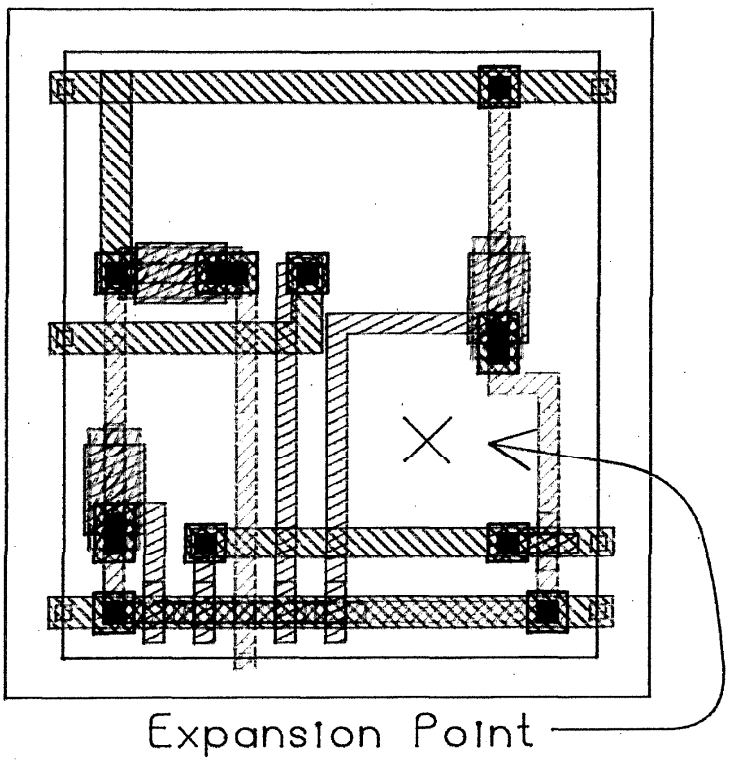


Figure 4-9: Expanded Cell

5. SOFTWARE ARCHITECTURE

REST is implemented in the SIMULA programming language. SIMULA is an object oriented language based on ALGOL60 [Birtwistle 73]. An object is a block capable of containing data and code. The object is the fundamental data structure mechanism in SIMULA. The object is the data structure mechanism that REST uses for the models previously described. The following discussion will be comprehensible with some familiarity of SIMULA.

A block diagram of REST is shown in figure 5-1. The bottom boxes contain the SIMULA base. This is the library of general routines that REST uses. The second major box is the standard data routines and the graphic package consisting of things, views and display. The third group is the REST modules consisting of uvec the data structures, stuck which is the line drawing interpreter and graphic interface, compac the compactor and file interfaces, parts the utilities and command parser and REST which is the shell for the program.

The module things contains several primitive data structures that are used throughout REST. They include point, vector and rectangles. A vector is a object that is a set of things which is capable of sorting itself. All objects in REST are subclassed by an object thing so that references can be used in vectors.

The SSP graphic package is shown in a box: Display, and Views[Wipfli 78]. The graphic package contains an object called device that can be created for the Charles terminal or the hp plotter. The object device is defined in the display module. In module views there is an object window which defines the viewport on the specific device. A users rectangle is supplied to the window to define the users viewport, and a virtual rectangle is supplied to the window to define its position on the device.

The module UVEC contains the data structure definitions and the design rules tables. The first set of objects in uvec are the lines and boxes for the line drawing interpreter. The next set of objects are the joints, contacts, connectors,transistors, and segments. The design rules are initialized in a table at the end of this module. The models or design rules can easily be changed in uvec.

The module STRUCK contains the line drawing interpreter, graphic interfaces, and

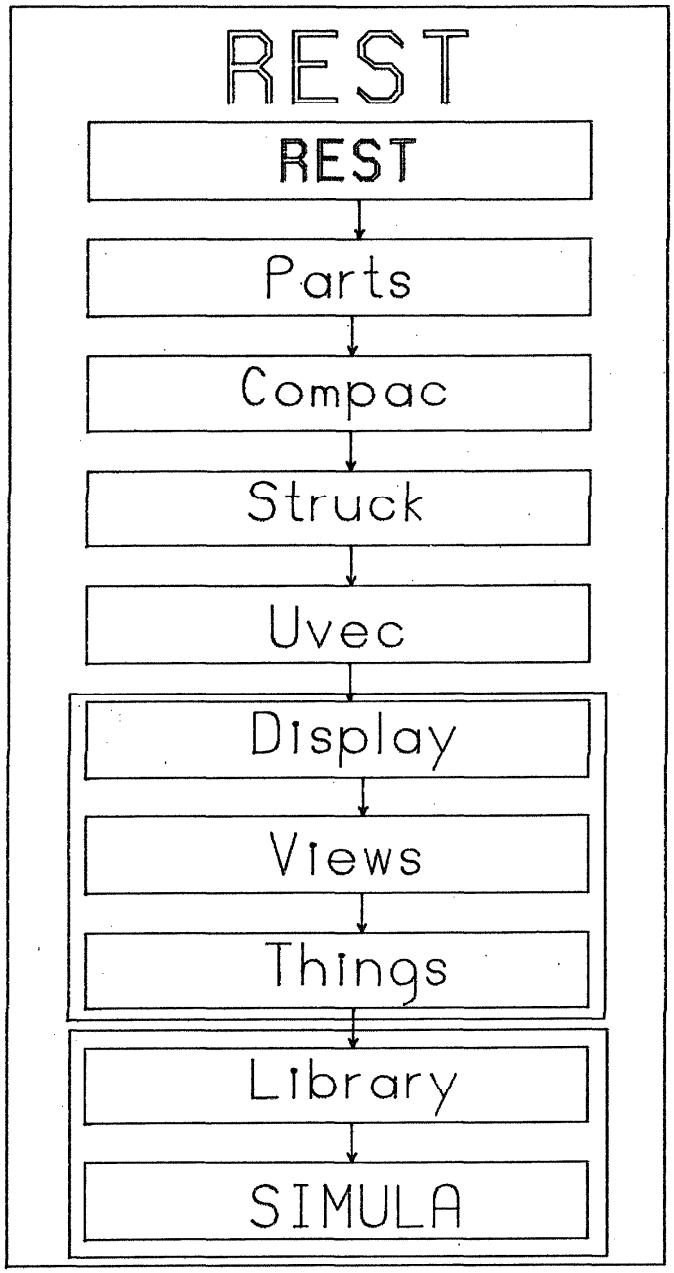


Figure 5-1: Software Blocks

base object for the stick cell. The procedure processlines is the line drawing interpreter. At the base of this procedure various internal procedures are called to correspond to the description in the algorithm section. These procedures are easily modifiable for other device or colored line recognition. At the base of module stuck are the plot procedures.

The module COMPAC contains the compaction and expansion routines. The object boxo defines the boxes used in the compaction algorithm. The features and Branches are also defined here. The procedure that effects the compaction is called packfeatures. This procedures calls definesfeatures for defining the features,and definebranches for defining the branches. The graph solving is accomplished by the procedure itself. The reading in of a stick file is accomplished by procedure getfile and write to disk by sendtofile. There are various other procedures for trimming, absorbing, constraint checking and data structure clean-up.

The module PARTS contains the command processor and utilities. The utilities consist of procedures: printstatus, setlinewidth, cutsegment, setcursor and cursorconstraint. There is an additional set of routines in the command processor for setting device lengths and width, constraints, and general parameters. The command processes is a standard parser with a feature that allows keywords to be a subset of their full name.

The module REST is the shell for the whole program. The main reason for the shell is that the main program can be loaded into the DEC 20 high core segment area.

6. ALGORITHMS

6.1 Analyzing

The analysis process digests the lines and boxes from the box editor. This function is to pattern recognize wire intersections, contacts, transistors and ports. This is the function of the line drawing interpreter. The raw input data from the box editor is an unordered set of colored lines and colored boxes. REST writes this data to a disk file so that recovery can be possible. The unordered set is partitioned into two sets: LINES and BOXES. The names used herein are merely for notation and clarity. The names may not correspond to the identifiers in the program. The algorithms in REST are set oriented and do operations on ordered sets of lists.

Each member of the LINES and BOXES is checked for proper color and checked to see if it is outside of the bounding box. Appropriate error messages are given for members that are in error and the members in error are deleted from the sets. The LINES and BOXES are now ordered by the lowest color,then by X, and by Y. The algorithm used for the sort is the quicksort[Sedgewick 77].

The first major process is to analyze the set of lines. During this process the joints and segments are created from the set of LINES. A line element has attributes of color, low X, Y, high X, Y and two terminal joint references. That is L(Color,LOW(X,Y),HIGH(X,Y),J1,J2). A simplified program follows.

The following program steps through each member of the LINES and compares it to a window set, ACTIVEBODY. The temporary set contains lines that are in a moving window that traverses up the cell. Lines in the temporary set that are below the testing member are culled. Lines in the temporary set that intersect with like colors lines are merged if parallel or split at the point of intersection.

```
create list ACTIVEBODY
For each member m of the LINES do
begin
      For each member n in ACTIVEBODY do
      begin
            if n.high.y is less then m.low.y then
            begin
                  if n.low is nil then n.low := create joint
                  if n.high is nil then n.high := create joint
                  create segment
                  add n.low,n.high to created segment
                  add to SEGMENTS, INSTANCES
                  delete n from ACTIVEBODY
            end else
            if n intersects m and they have like colors then
            begin
                  if n parallel m then
                  begin
                        merge and create appropriate joints,
                        add joints  to m,n
                  end else
                  begin
                        put into lines k,l if necessary
                        add k,l pieces to ACTIVEBODY,
                        create appropriate joints
                  end
            end
      append m to ACTIVEBODY
      end
end
delete LINES
For each member n in ACTIVEBODY do
begin
      create joints, segments add to SEGMENTS, INSTANCES
      delete n from ACTIVEBODY
end
sort INSTANCES, SEGMENTS
```

The result of this process is the creation of a proper set of joints and segments from the raw input colored lines. These are stored in ordered sets, INSTANCES for the joints and SEGMENTS for the segments. These newly created segments are set to the default width for their specific layer. The sets INSTANCES and SEGMENTS are now sorted. The set of LINES is then discarded.

The next step is to process the contacts. This is done similar to the lines by creating a temporary set for holding the boxes, and a set for the intermediate segments. The following simplified program shows the algorithm.

```
create list BOXLIST, SEGLIST, SEGS
SEGS :- ordered copy of SEGMENTS
For each member b of the BOXES do
begin
      while s:-head(SEGS) low.y <= b.high.y do
      add to s SEGLIST delete s from SEGS
      For each member c in BOXLIST do
      begin
            if c.high.y less b.low.y then
            begin
                  if c.contact then append to instances
                  delete c from BOXLIST
            end else
            if b intersects c then merge
      end
      For each member s in SEGLIST do
      begin
            if s.high.y less b.low.y then
            delete s from SEGLIST
            else
            if s intersects b  then
            begin
                  split s if necessary add split(s) to SEGLIST,SEGMENTS
                  b.contact:=create contact
                  insert joints in b.contact
            end
            append b to BOXLIST
      end
      For each member c in BOXLIST do
      begin
      if c.contact then append to instances
      delete c from BOXLIST
end
```

The box analyzing algorithm similar to the line algorithm has a moving widow that traverses up the cell. The routine steps through the list of ordered BOXES processing one box at a time. The stepping box is M in the routine. The routine compares box M to the boxes and the segments in this window. Segments that intersect boxes are split into two with an intervening joint. The joint of intersection is added to the box. Any segment or box that is below box M is deleted from the window. This process creates the contact components and adds them to the set of INSTANCES.

The next process is to find all the transistors. This is accomplished by using a moving window as above comparing the red segments to the green segments. The choice of color is table driven and can easily be changed for various models. The recognition of resistive type devices is also effected here by looking for a red line colinear with a green with a contact at one end. In the following program the creation

of a specific transistor is accomplished by routine, create-transistor. The recognized transistors are then added to the set of INSTANCES. The newly created transistors are set to the default value for their specific type.

```
create list POLYLIST, DIFFUSIONLIST
SEGS :- ordered copy of SEGMENTS
For each member s of the SEGMENTS do
begin
      For each member k in POLYLIST do
      begin
            if s.high.y less k.low.y then
            delete K from POLYLIST
      end
      For each member k in DIFFUSIONLIST do
      begin
            if s.high.y less k.low.y then
            delete K from DIFFUSIONLIST
      end
      if s.color is polysilicon then
      begin
            For each member k in DIFFUSIONLIST do
            if s.intersects(k) then
            begin
                  l:=create-transistor(s,k)
                  if not l is nil then add l to instances
            end
      end else
      if s.color is diffusion then
      begin
            For each member k in POLYLIST do
            if k.intersects(s) then
            begin
                  l:=create-transistor(k,s)
                  if not l is nil then add l to instances
            end
      end
end
```

The process of transistor recognition could easily be enhanced to accept various other forms. The required change would be to modify routine create-transistor. This routine is small and easily extended. For example, a transistor with a path could be recognized. It would require model adaptation for path type transistors. An example of a path transistor is shown in figure 6-1.

To be able to recognize other combinations of layers for transistors would require no changes to the recognizer routine. It would require changes to the tables. This would be used for other MOS logic family. For instance CMOS has two transistor types which use two different diffusions.

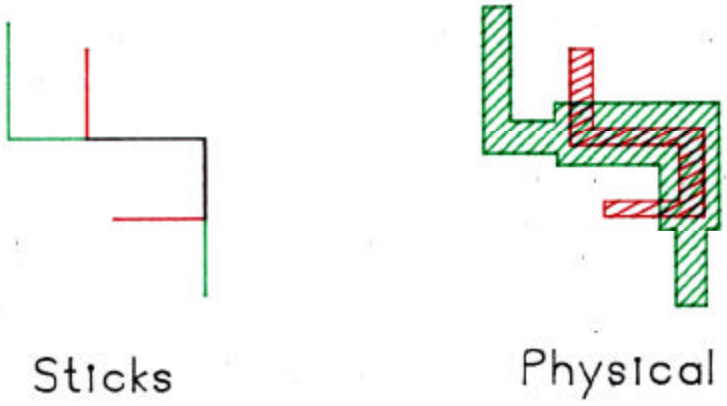


Figure 6-1: Transistor Model Recognition

The connectors are now determined from joints that are coincidence with the edge of the cell. These are stored in their appropriate connector side set: LEFT, RIGHT, TOP or BOTTOM.

The final process is to correlate the newly created segments, and instances with the current cell if it exists. This is done in a similar process as above with a moving window. No sorts are required for this process since the old and new data are in sorted order. For each new segment that has a corresponding old segment, the wire width is copied to the new segment. The data for each corresponding instance is also copied. This includes length, widths for transistors and contacts, and instance names. The constraints are also updated to reflect the new data.

The computation complexity of this process is order n log n. This figure comes from the computational time to sort the lists. A quicksort algorithm is used in this process[Sedgewick 77]. The number of items in the window list at any one time is about ten.

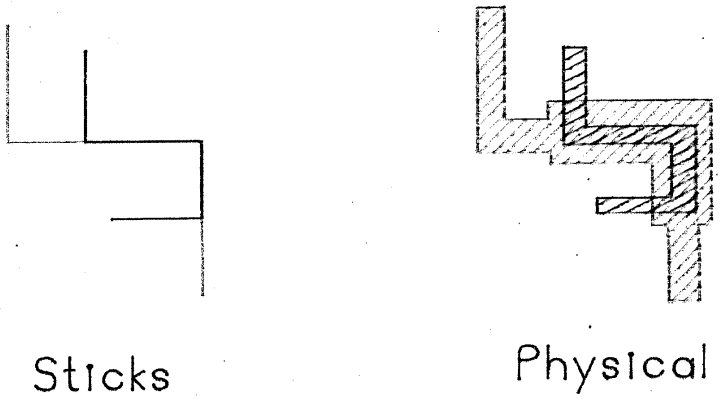


Figure 6-1: Transistor Model Recognition

The connectors are now determined from joints that are coincidence with the edge of the cell. These are stored in their appropriate connector side set: LEFT, RIGHT, TOP or BOTTOM.

The final process is to correlate the newly created segments, and instances with the current cell if it exists. This is done in a similar process as above with a moving window. No sorts are required for this process since the old and new data are in sorted order. For each new segment that has a corresponding old segment, the wire width is copied to the new segment. The data for each corresponding instance is also copied. This includes length, widths for transistors and contacts, and instance names. The constraints are also updated to reflect the new data.

The computation complexity of this process is order n log n. This figure comes from the computational time to sort the lists. A quicksort algorithm is used in this process[Sedgewick 77]. The number of items in the window list at any one time is about ten.

6.2 Compaction

A graph based compaction algorithm is used in REST. A pictorial representation of the graph for a shift register cell is shown in figure 6-2. This example is for vertical compaction. Each node in the graph represents a feature. A feature is a set of primitive elements that moves as a unit. Node 1 and 7 in figure 6-2 represent the bottom and top edge of the cell. Node 2 is the metal wire with an attached connector. Each additional node represents a feature in the graph. The node number in the graph has a corresponding number in the sticks representation next to the nodes feature. The determination of the feature is a simple mapping from REST internal sticks data structure.

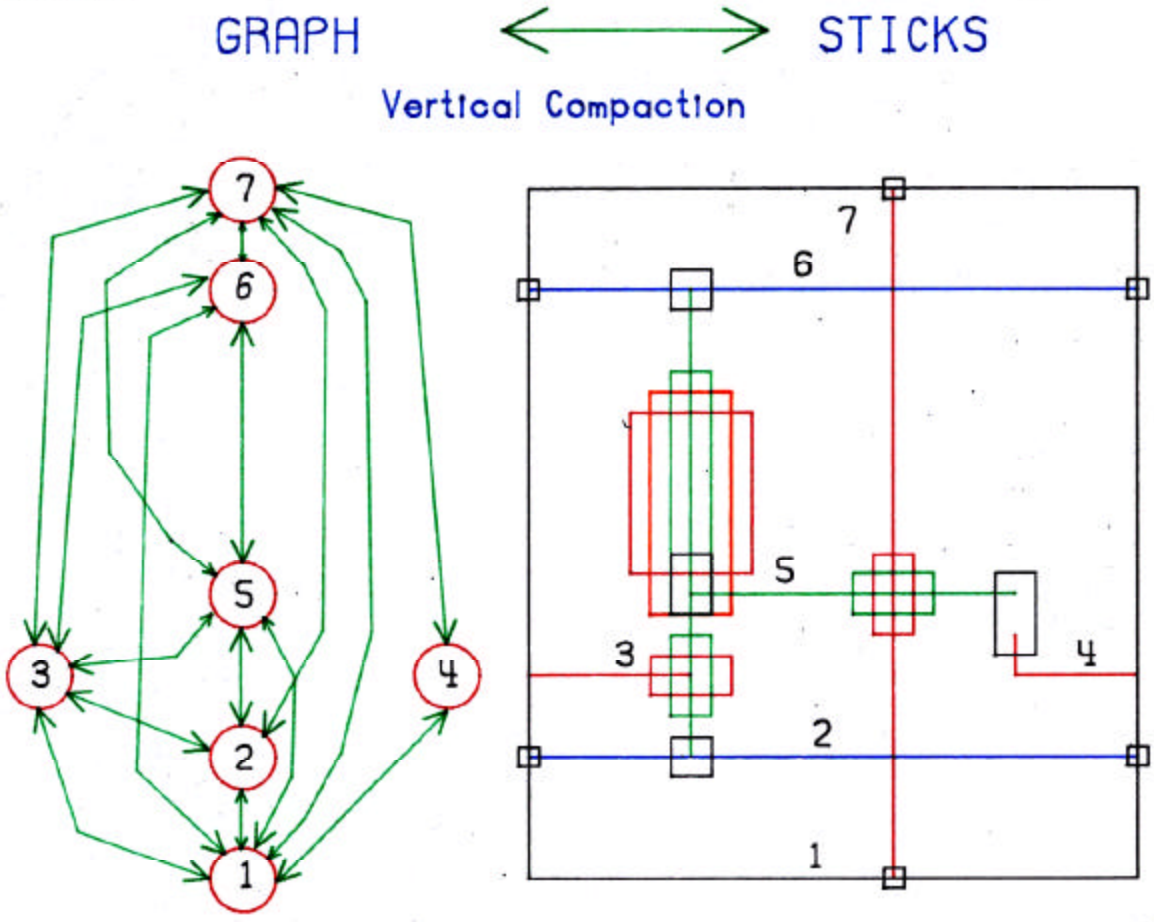


Figure 6-2: Compaction Graph

6.2 Compaction

A graph based compaction algorithm is used in REST. A pictorial representation of the graph for a shift register cell is shown in figure 6-2. This example is for vertical compaction. Each node in the graph represents a feature. A feature is a set of primitive elements that moves as a unit. Node 1 and 7 in figure 6-2 represent the bottom and top edge of the cell. Node 2 is the metal wire with an attached connector. Each additional node represents a feature in the graph. The node number in the graph has a corresponding number in the sticks representation next to the nodes feature. The determination of the feature is a simple mapping from REST internal sticks data structure.

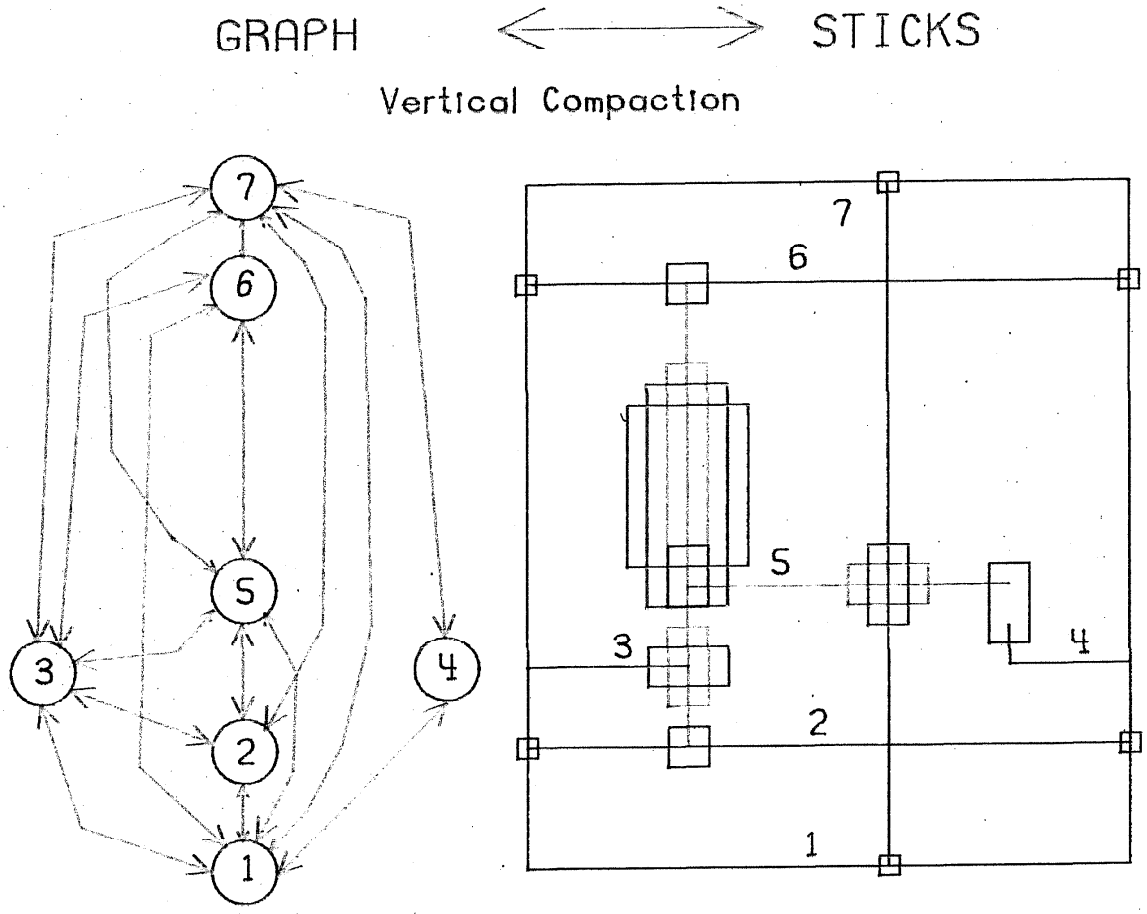


Figure 6-2: Compaction Graph

Consider the feature as tuple F(val,base,BK,E,BX,left,right) where VAL is the new location for the feature, BASE is the current location, {BK} is the set of all branches that apply to this feature, {E} is the features set of elements, that is segments and instances, and {BX} is the set of design rule models boxes. The LEFT and RIGHT values are the extreme end points for the feature. For vertical compaction theses points are x values while for horizontal compaction they are y values.

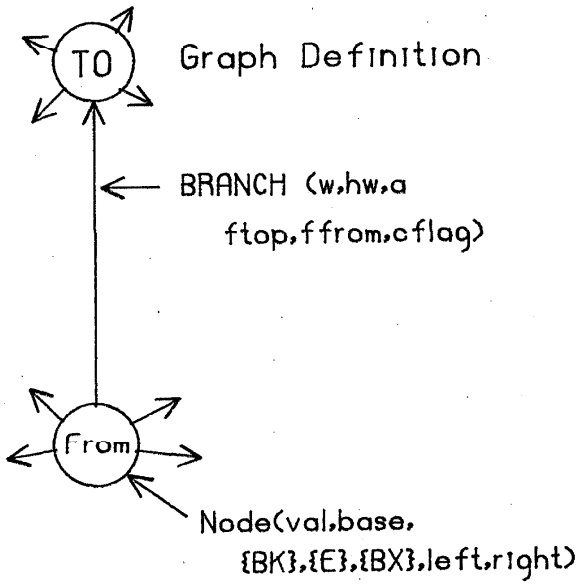


Figure 6-3: Branch Model

The branches of the graph represent the design rules between any two features and the affinity factor. Figure 6-3 shows the relationship for a branch. A branch is a directed arc in the compaction graph. Each branch contains the minimum separation between its two nodes.

Consider the branch as a tuple B(w,hw,a,ftop,ffromp,cflag). The elements of the branch are w for the minimum separation of the two adjoining feature nodes pointed to by, ftop the feature-to-pointer and ffromp the feature-from-pointer. The value hw is the maximum separation of the two features. The value a is the affinity factor. The cflag denotes this branch was defined from a user constraint. The cflag is used to detect

user defined loops in the graph so that error messages to the user will be meaningful.

The generation of the features comes naturally from the REST data structure. This is accomplished by collecting all joints and components with paths that have common left and right pointers. The following program illustrates the algorithm.

```
TEMPINST:-copy INSTANCES
create list FEATURES
while C:-head(TEMPLIST) do
begin
      create new feature F
      if c is component then
      begin
            delete c from templist
            append c to F
            for each joint j of c do
            begin
                  append j to F
                  delete j from TEMPLIST
                  traceleft(j,F);traceright(j,F)
            end
      end else
      begin
            append c to F
            delete c from TEMPLIST
            traceleft(c qualification joint,F);
            traceright(c qualification joint,F)
      end
      create new feature F
end
```

The above algorithm essentially visits each component, joint, and segment once. Each individual feature is constructed from tracing all joints and segments through the left and right pointer with procedure trace. This algorithm is for vertical compaction. For horizontal compaction the top and bottom pointers of the joints would be followed.

After this process the base value for each feature is set to the value of the lowest element. In addition the left and right values are also set. The set of boxes {BX} will be defined at a later point in time.

The most computational demanding of all the compaction process is determining the branches. That is specificing the minimum separation distances from each feature to every other feature. The actual minimum separation for each layer pair is the design rules tables. In general this would require each design rule box of each feature to be compared to every other feature. However with the left and right values it is only

necessary to compare the design rule boxes of the features when the left and right values intersect within a fuzz value. Even with this optimization the time could be large. This is because the sum total of all the design rule boxes of the features that do intersect is large.

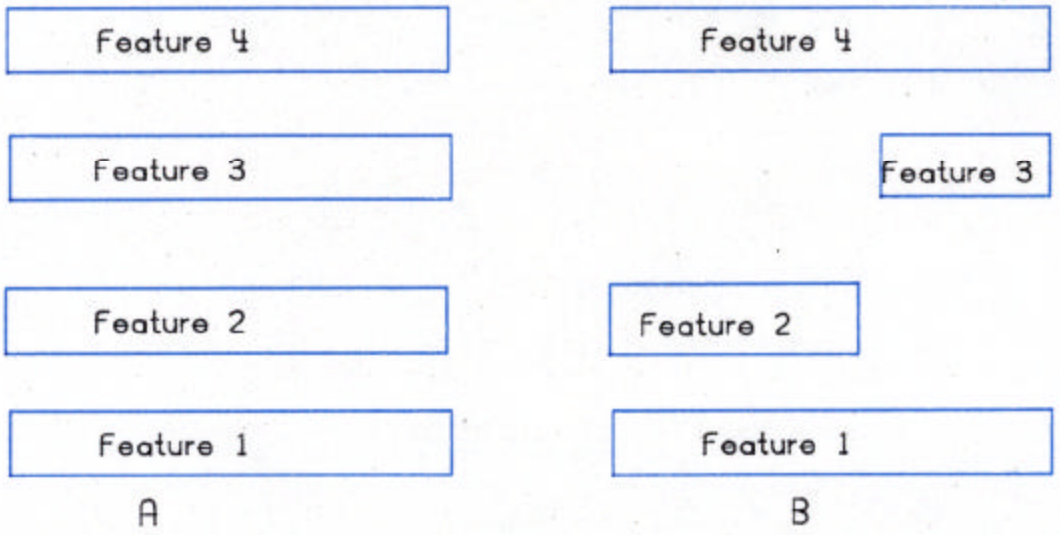


Figure 6-4: Design Rule Box Shadowing

To solve this problem REST employs a shadowing technique. This process cuts down the amount of necessary geometry that needs to be compared. An example is shown in figure 6-4 with metal boxes that need to be compared. Feature 1 in part A is to be compared to features 2,3 and 4. In A feature 1 only needs to be compared to feature 2 because feature 1 is completely shadowed by 2. That reduces the comparison for this example from 3 to 1. However in part B Feature 1 is not covered by feature 2 nor feature 3 so that three comparisons are necessary.

REST uses the method of reducing the design rules boxes by the amount of coverage. If a design rules box is completely covered it is deleted. If the box is partially covered

necessary to compare the design rule boxes of the features when the left and right values intersect within a fuzz value. Even with this optimization the time could be large. This is because the sum total of all the design rule boxes of the features that do intersect is large.

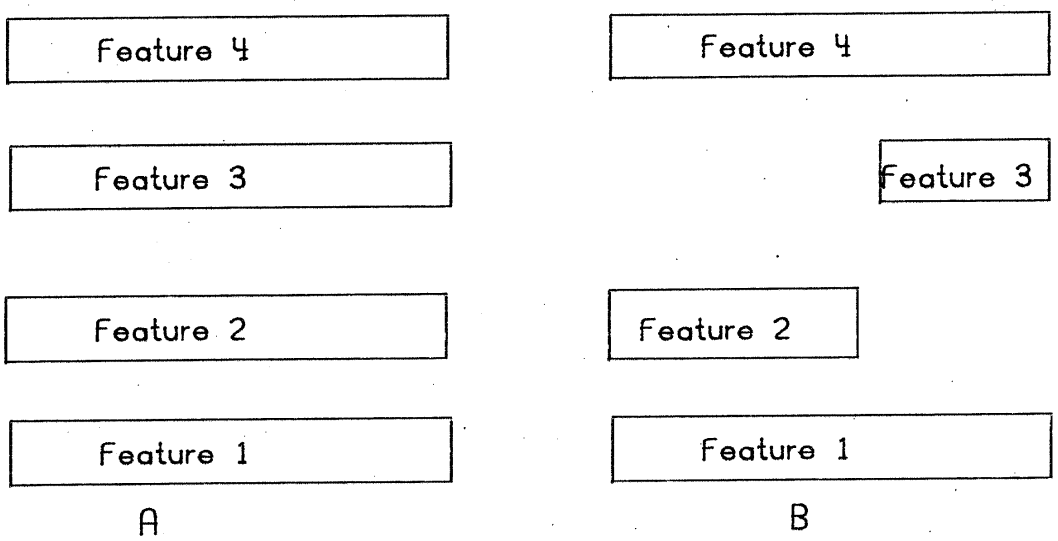


Figure 6-4: Design Rule Box Shadowing

To solve this problem REST employs a shadowing technique. This process cuts down the amount of necessary geometry that needs to be compared. An example is shown in figure 6-4 with metal boxes that need to be compared. Feature 1 in part A is to be compared to features 2,3 and 4. In A feature 1 only needs to be compared to feature 2 because feature 1 is completely shadowed by 2. That reduces the comparison for this example from 3 to 1. However in part B Feature 1 is not covered by feature 2 nor feature 3 so that three comparisons are necessary.

REST uses the method of reducing the design rules boxes by the amount of coverage. If a design rules box is completely covered it is deleted. If the box is partially covered

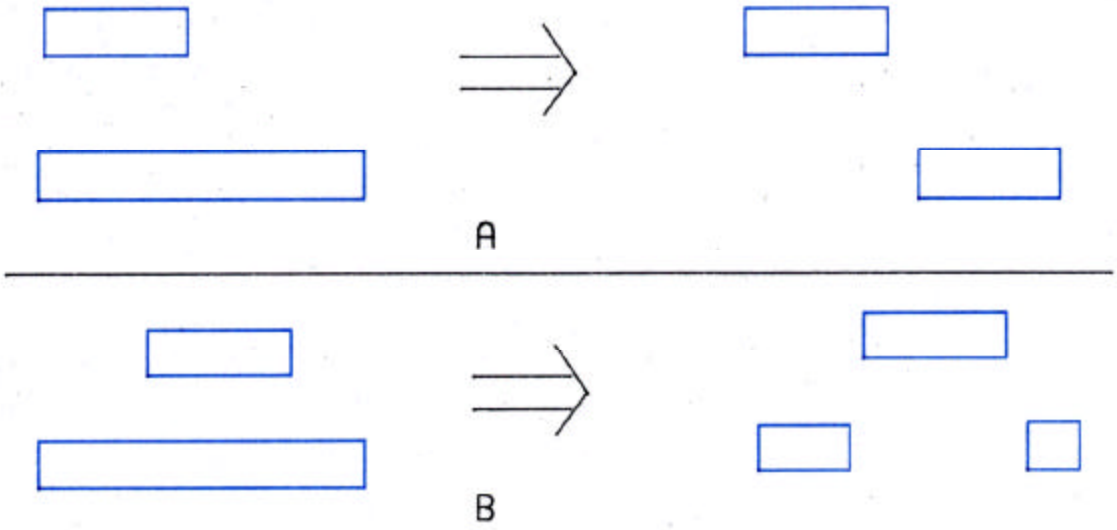


Figure 6-5: Box Covering

at one end it is reduced by the amount of coverage. If it is covered in the middle it is split. This is shown in figure 6-5.

In addition to reducing the design rule boxes the left and right values of the feature are also deflated. This decreases the scope of additional comparison for the feature as an entity.

The design rules box is a tuple(color,netnum,left,right,low,high). The left and right are the limits for the box as in the feature. The low and high are the orthogonal limits used in the determination of the branch w value. The netnum is a unique number for all geometry that is connected. That is if two blue boxes were connected through a path of segments they would have the same number. If two design rules boxes are connected they will not be compared.

To reduce the amount of core memory REST uses, the design rules boxes are created on demand. That is in the comparison process if a features boxes have not been created they will be constructed at that time. Since the showing method reduces the number of boxes during comparing process and boxes are only created on demand there is only a

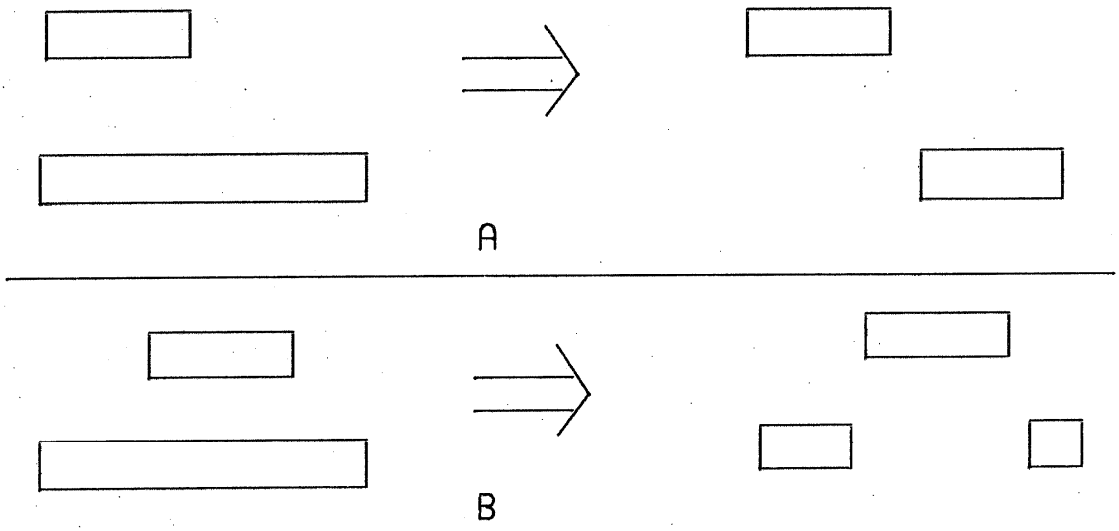


Figure 6-5: Box Covering

at one end it is reduced by the amount of coverage. If it is covered in the middle it is split. This is shown in figure 6-5.

In addition to reducing the design rule boxes the left and right values of the feature are also deflated. This decreases the scope of additional comparison for the feature as an entity.

The design rules box is a tuple(color,netnum,left,right,low,high). The left and right are the limits for the box as in the feature. The low and high are the orthogonal limits used in the determination of the branch w value. The netnum is a unique number for all geometry that is connected. That is if two blue boxes were connected through a path of segments they would have the same number. If two design rules boxes are connected they will not be compared.

To reduce the amount of core memory REST uses, the design rules boxes are created on demand. That is in the comparison process if a features boxes have not been created they will be constructed at that time. Since the showing method reduces the number of boxes during comparing process and boxes are only created on demand there is only a

small number of boxes in core relative to the total number of boxes.

The order of the boxes in the box set BX in the feature is sorted from left to right to further reduce the comparison time. This is effected by having a moving window pass along the two features being compared.

This deletion and reduction of boxes is effected after each complete comparison of two features. This is so that a box is not reduced prematurely. A simplified program follows.

```
for each member F of features do
for each follow member F1 features and F1.BX.Cardinal >0 do
if  F intersects F1 then
begin
      w:=compareboxes(F1.BX,F2.BX);
      append new branch(w) to F1.BK
      append new branch(w) to F2.BK
      reduce boxes of F1
      reduce left,right of F1
end;
```

The comparison routine in the above program accomplishes the moving window and marks the boxes to be deleted. Also the routine splits the boxes. Generally it does not take but a few comparisons of features to stop the comparison process of a single feature.

The ability to allow features to cross that are connected limits the time at which reduction can occur. The reason for this is that if two features are connected and reduction occurs, any boxes above the second feature are not added to the lower features branches. What this means is that if a feature connected to one above crossed it, there would be no branches for features above and this feature would undoubtedly short to the features above. Therefore reduction does not occur for those colored boxes that are in common.

The actual value for the separation is retrieved from a table of design rules. This contains the minimum separation for sets of layers or sometimes noted as colors. This table is easily changed.

After the branches are constructed the affinity and user constraints are added. Directly connected features will have their related branches updated with the affinity

factor for the type of connection and width of connection. For two features to be directly connected there must be a single segment that connects them both. The value for the affinity like design rules is taken from a table. The current used values are: 1 for metal and 3 for diffusion and polysilicon. The larger values increase the attractiveness of the two features. If two features are directly connected but do not have a design rule a branch, one will added with a nil w value and the appropriate affinity.

The user constraints are added as branches with cflag set that it is a user constraint. This flag is used in the graph solving phase when loops are detected.

The next process is to solve the constructed graph. This is effected by a depth first method marking each node in the process so that loops can easily be detected. If a loop occurs the depth process stops and retreats, deleting and giving a warning on the appropriate user constraint. The graph is then cleared and resolved.

At this point the cell is at its minimum size in the graph. There will be several features that can move within limits. The next process is to adjust the features for affinity. This is accomplished by examining each for an attractiveness toward another feature. If this occurs the feature will move as close as possible to the other feature. The two features will then be merged. After which the process repeats.

The final step in the compaction process is to adjust the locations of each element in the features. After this is completed the graph is deleted.

This completes the compaction process. This algorithm is order(n) exclusive of the sort time due to the shadowing and other techniques. It could be further reduced by using the method of Bentley[Bentley 80a, Bentley 80b]. This involves using buckets for the features and boxes and only comparing data in common buckets.

6.3 Expansion

There are three types of cell expansion in REST. The first is the bounding box expansion. The second is the general expansion, that is separation of each feature by some user defined amount. Third is the expansion around a point that is to open an area for addition logic. These cell expansions are used mainly to add room to place more

logic in the cell.

The bounding box expansion is the simplest of the three. The user specifies the side and amount of expansion to REST. REST in turn moves the appropriate side by the user specified amount. In addition to moving the bounding box the ports on the side to be expanded are also moved. Since the location of the segments is determined from the joints and, in this case they are ports, the attached segments are properly adjusted.

The general expansion uses the graphs that are described in the compaction process in section 6.2 page 43. The feature definition routine and graph solver are used for this process. The features are constructed as previously described. The branches are constructed by intersection of left and right features. The w value is set from the distance between the two features plus the user defined expansion value.

The nodes in the graph are initialized to their current location. The graph is then solved as described before. The effect of this is to separate the features from each other by the amount the user specifies.

The expansion around a point is the most elaborate expansion and used greatly by the designer. The designer marks a position with the cursor and gives a value for expansion, and the direction: left, right, top, bottom, X, or Y. This process uses the compaction graph previously described in section 6.2 page 43. The features and branches are constructed as previously described. The graph is initialized to its current position. This sets the feature to its current position.

An initial feature is then found depending on the direction specified. For example, if the direction was left, the initial feature would be set to the first feature on the left of the mark point that covers the point with its left and right values. The feature is then moved out by the user defined value and the graph is solved from the initial feature. The graph is then cleaned up as described before.

7. RESULTS AND CONCLUSIONS

7.1 Example

An initial sample input from a designer is shown in figure 7-1. This is a one bit two bus register cell in NMOS. There is a clock line for the refresh cycle. The first thing to notice in this sample is that it is a crude sketch of the desired circuit. The contact or black boxes are irregularly shaped. The lines overshoot and come close to other lines in the sketch. The orange or implant covers the pullups irregularity.

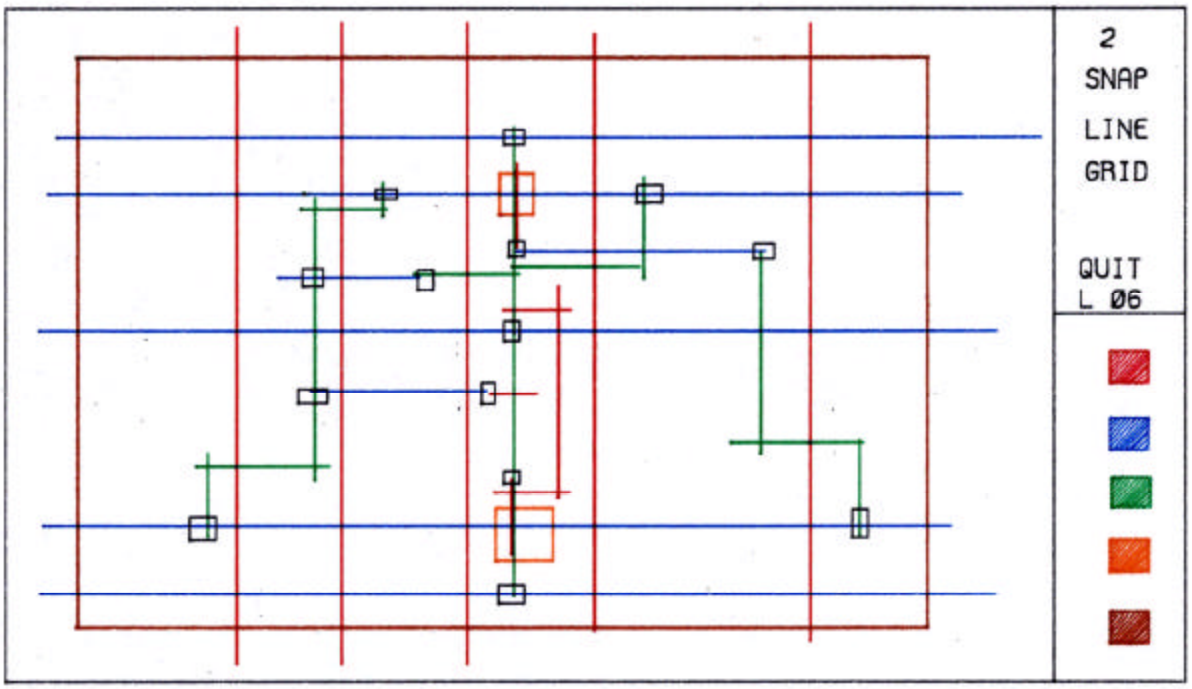


Figure 7-1: Two Bus Register Cell

7. RESULTS AND CONCLUSIONS

7.1 Example

An initial sample input from a designer is shown in figure 7-1. This is a one bit two bus register cell in NMOS. There is a clock line for the refresh cycle. The first thing to notice in this sample is that it is a crude sketch of the desired circuit. The contact or black boxes are irregularly shaped. The lines overshoot and come close to other lines in the sketch. The orange or implant covers the pullups irregularity.

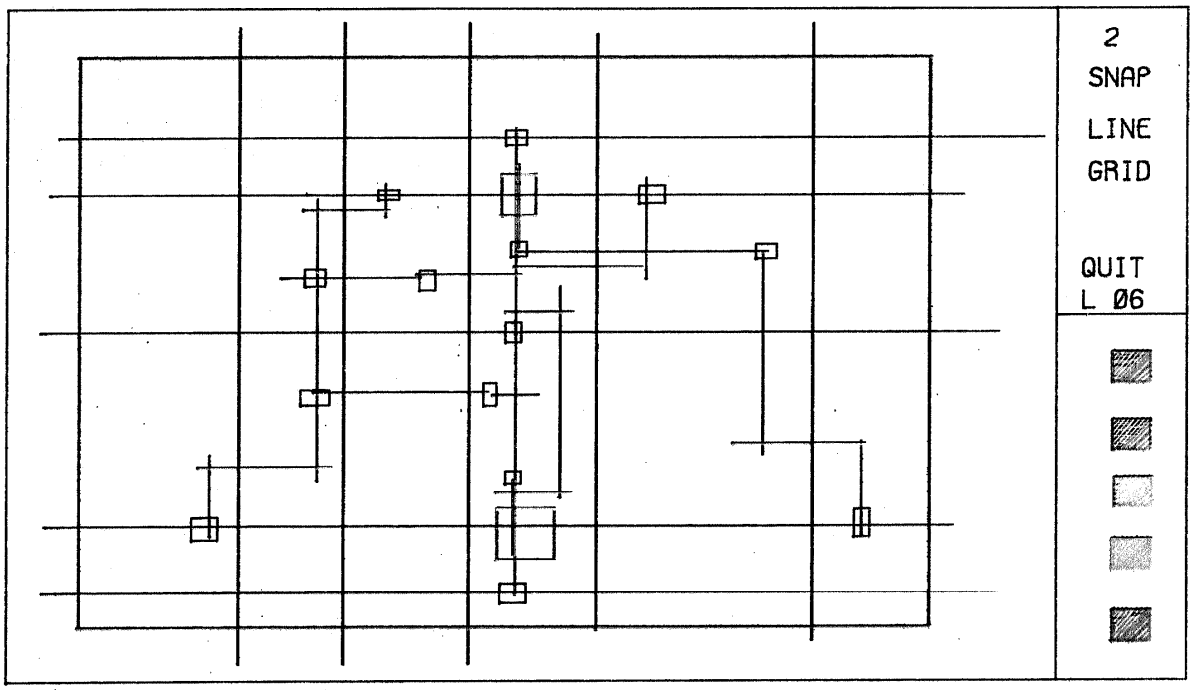


Figure 7-1: Two Bus Register Cell

In the sketch the interface signals are drawn so that they extend beyond the cell box shown in black. This box is the boundary of the cell, Any line that extends outside will be construed as a port. In the initial sketch the blues or metal on the left and right are the power and data bus while the red or polysilicon lines from the top to the bottom are the control lines.

The intent for this cell is that it would be replicated in a design for the word size and number of words. This imposes requirements on the cell that the ports on each side be at the same relative location as the opposite port.

The recognized stick and physical drawing is shown in figure 7-2. The first thing to notice is that the transistors and contact have been recognized. The lines that overrun or just came close to other lines were recognized as connected. After recognition, the lines were snapped to be orthogonal. The snapping was required because of the various points at which a line may enter a component.

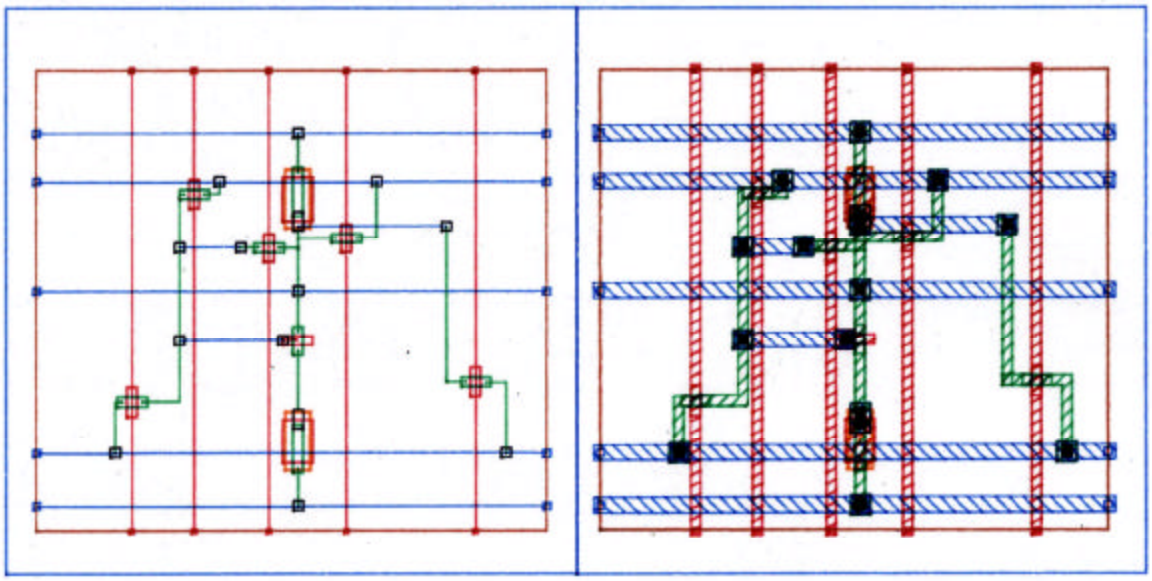


Figure 7-2: Recognized Two Bus Register Cell

In the sketch the interface signals are drawn so that they extend beyond the cell box shown in black. This box is the boundary of the cell, Any line that extends outside will be construed as a port. In the initial sketch the blues or metal on the left and right are the power and data bus while the red or polysilicon lines from the top to the bottom are the control lines.

The intent for this cell is that it would be replicated in a design for the word size and number of words. This imposes requirements on the cell that the ports on each side be at the same relative location as the opposite port.

The recognized stick and physical drawing is shown in figure 7-2. The first thing to notice is that the transistors and contact have been recognized. The lines that overrun or just came close to other lines were recognized as connected. After recognition, the lines were snapped to be orthogonal. The snapping was required because of the various points at which a line may enter a component.

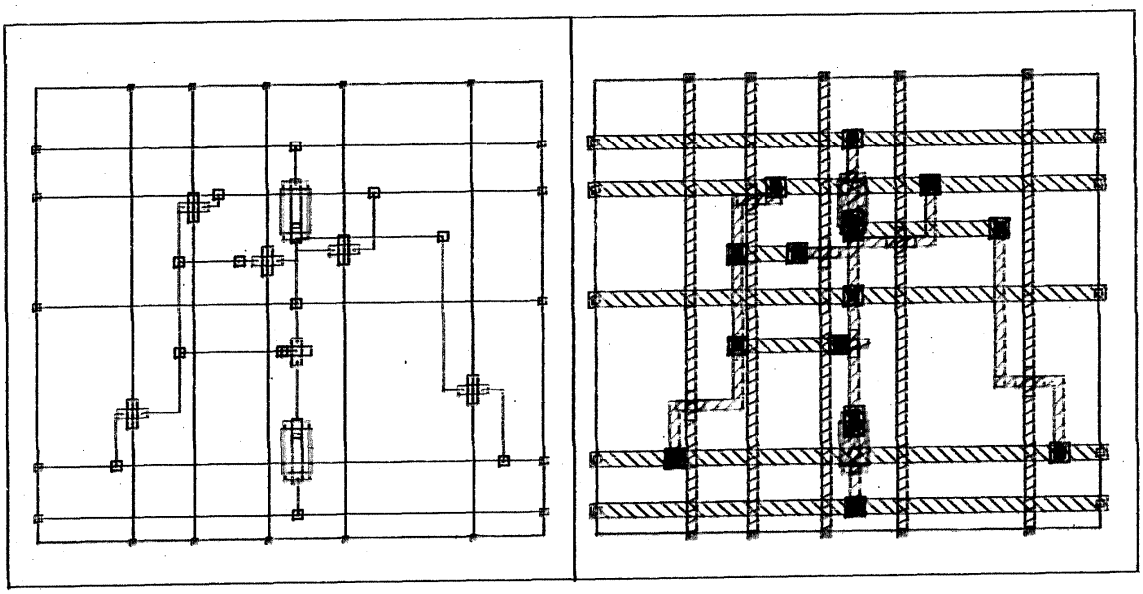


Figure 7-2: Recognized Two Bus Register Cell

The drawing also shows the recognized connection points called ports. The ports are on the perimeter of the cell. They are shown with a black box token marker in the drawing 7-2.

The cell has not been spaced to design rules as yet. This can be seen by examining the physical picture in figure 7-2. Notice the closeness of the contact boxes. The spacing to design rules is accomplished in the compaction phase.

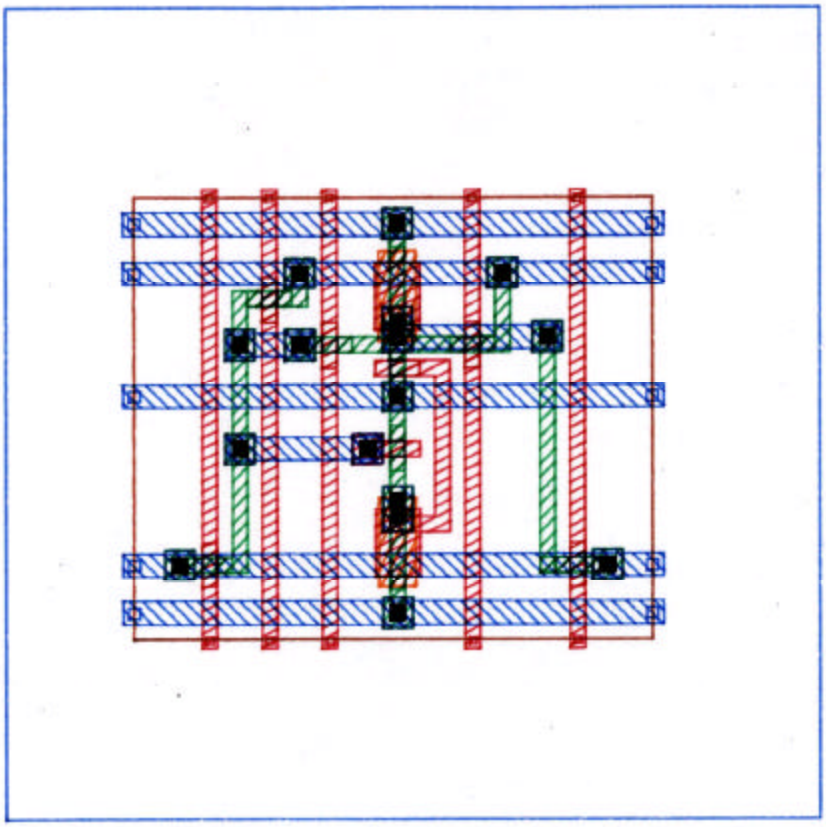


Figure 7-3: Compacted Two Bus Register Cell

The first compaction is shown in figure 7-3 to scale with figure 7-2. This cell is now design rules correct. The cell is the smallest it could be. Several iterations will be

The drawing also shows the recognized connection points called ports. The ports are on the perimeter of the cell. They are shown with a black box token marker in the drawing 7-2.

The cell has not been spaced to design rules as yet. This can be seen by examining the physical picture in figure 7-2. Notice the closeness of the contact boxes. The spacing to design rules is accomplished in the compaction phase.

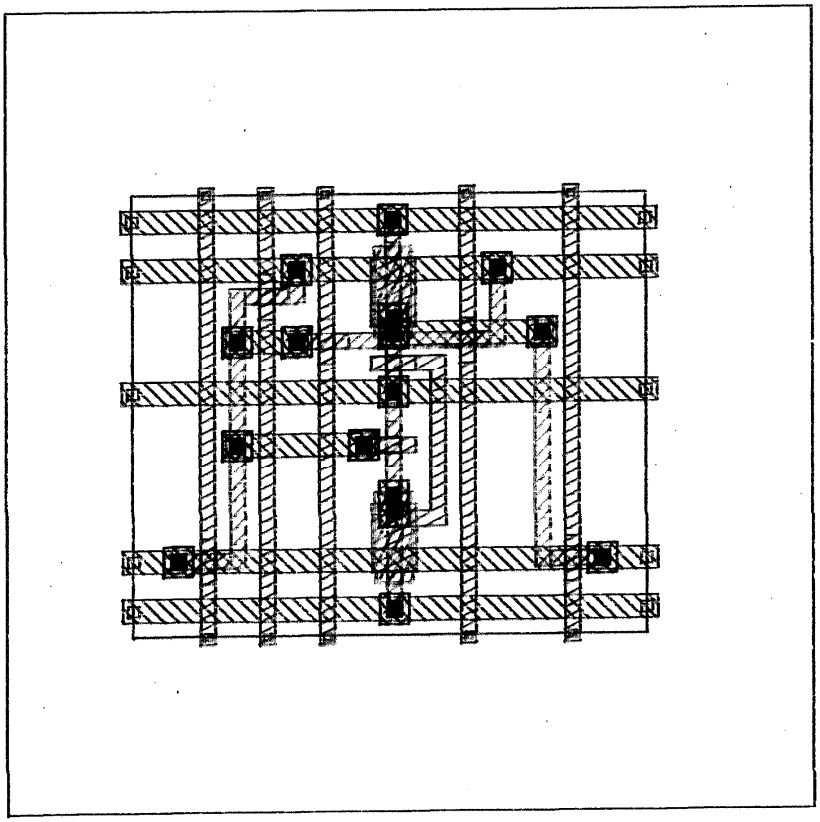


Figure 7-3: Compacted Two Bus Register Cell

The first compaction is shown in figure 7-3 to scale with figure 7-2. This cell is now design rules correct. The cell is the smallest it could be. Several iterations will be

required to enhance the area size.

This cell will not function properly in a circuit since the device ratios are incorrect. The changing of the ratios is a simple command in REST. It requires a statement to REST to change the length or width or both, and place the cursor over the desired device to be changed. In addition the cell will need the power lines widened to carry the power for several cells. This is because the cell will be used in an array and the power bus will be shared among many cells.

The size of the initial compacted cell is 69 by 59 lambda.

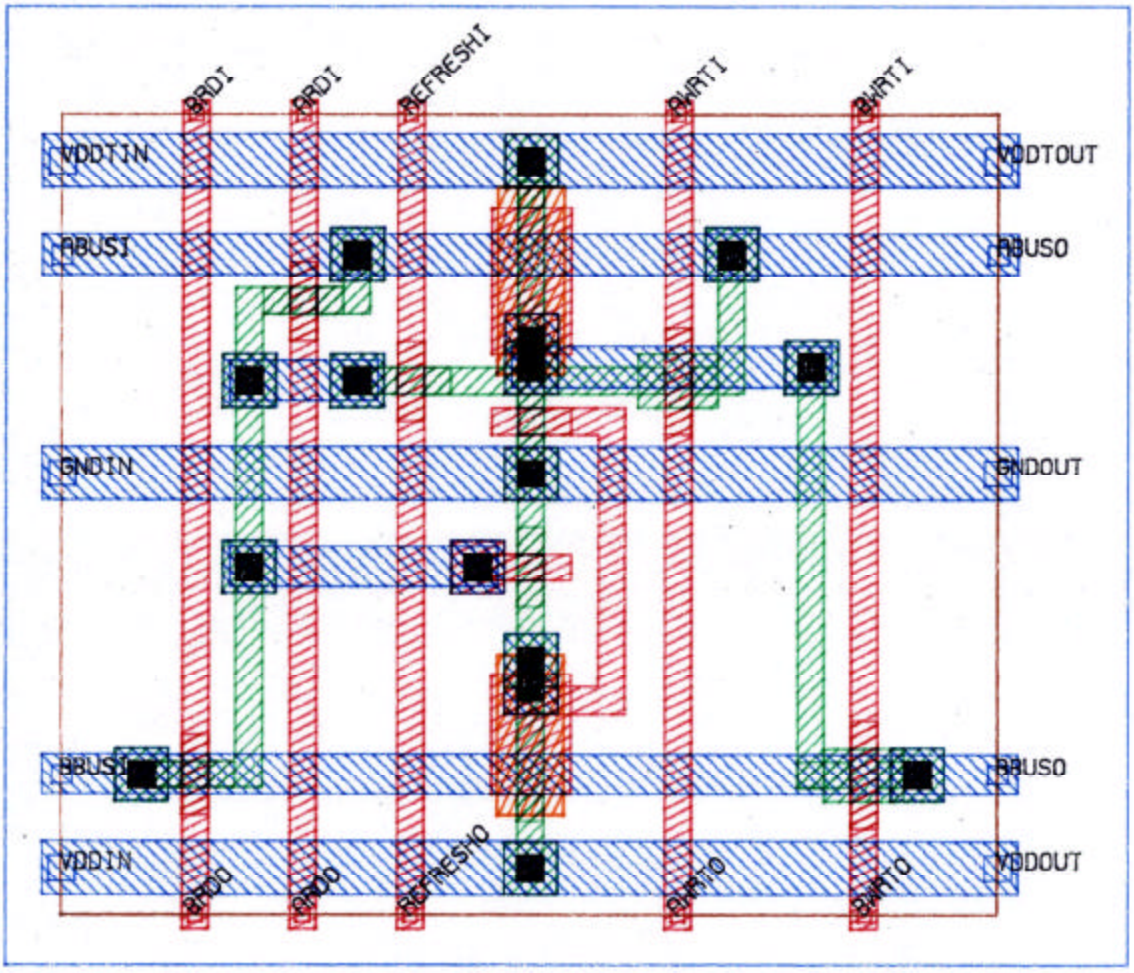


Figure 7-4: Several Compaction Two Bus Register Cell

required to enhance the area size.

This cell will not function properly in a circuit since the device ratios are incorrect. The changing of the ratios is a simple command in REST. It requires a statement to REST to change the length or width or both, and place the cursor over the desired device to be changed. In addition the cell will need the power lines widened to carry the power for several cells. This is because the cell will be used in an array and the power bus will be shared among many cells.

The size of the initial compacted cell is 69 by 59 lambda.

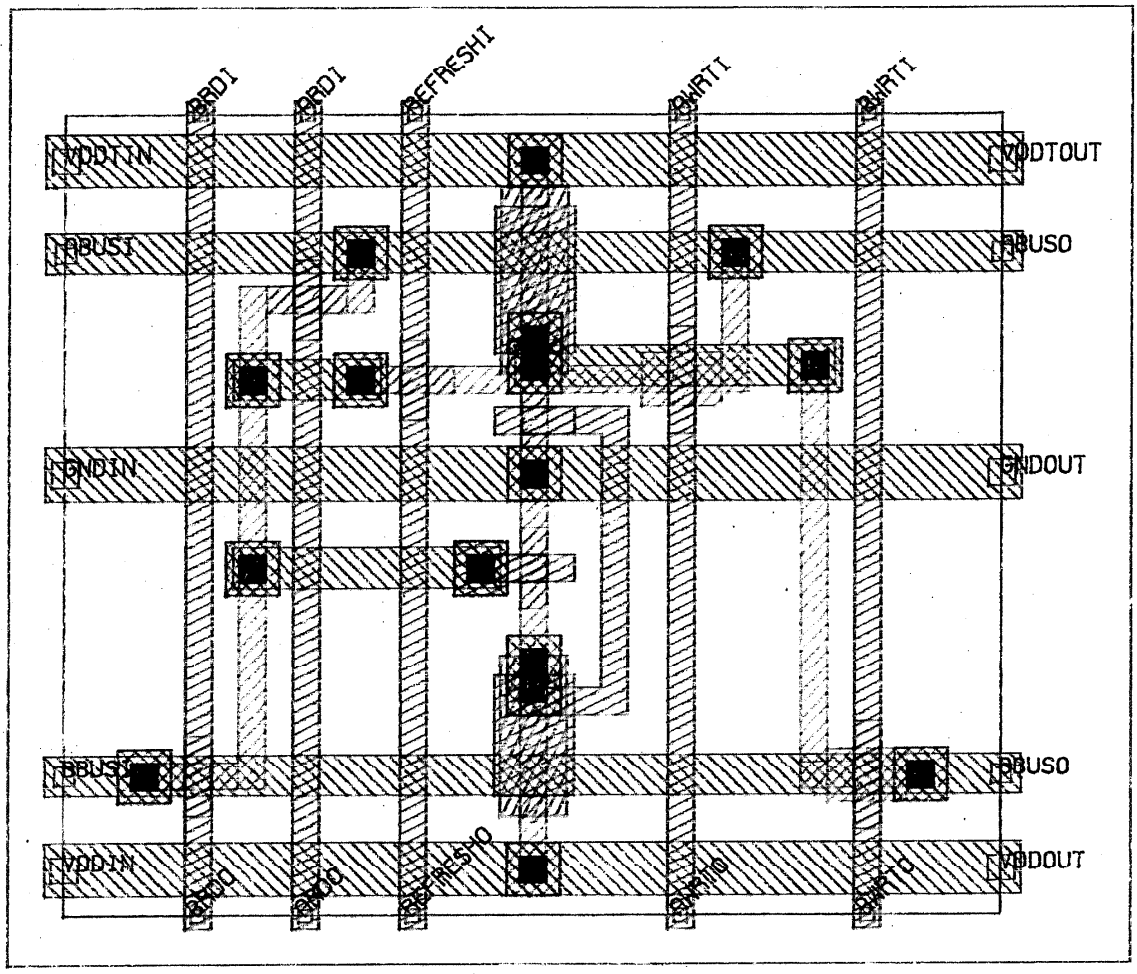


Figure 7-4: Several Compaction Two Bus Register Cell

The cell is shown in figure 7-4. with the device ratios changed properly. The power lines widths were also extended from 3 to 4 lambda. Annotation was added to the ports. Annotation is easily added. This is done by giving the cursor command to set the names. Then the cursor is placed over the desired component, joint or ports with a press of the button. A name is now entered textually from the terminal.

So far in the design of this cell approximately ten minutes were spent. This cell could be compacted more by adding jogs and some topological changes. In any design of a cell it would be required to effect several passes through the compactor and editor to achieve a reasonable cell area.

Now if we are to add jogs the ports would be free to float away where they wished. The reason they do not now is that the ports on opposite sides are on the same wire. A constraint will be needed to hold the ports on opposite sides at the same location. This is easily accomplished by the cursor constraint command. This is accomplished by giving the cursor constraint command with a horizontal or vertical constraint and the relation operator. In this case the equal operator will be used to constrain the ports on opposite sides to be at the same relative location.

The size of the second compacted cell is 70 by 60 lambda, which is an increase from the original compaction of 1 by 1 lambda. This increase is due to the added device sizes.

Figure 7-5 shows the cell after several additional compactions with designer jog insertion. The addition time spent is about five minutes. This time is for a designer that is familiar with REST.

Several jogs were added in the cell by the designer. This decreased the size of the cell by employing unused space. In addition constraints were added to align the ports on opposite sides of the cell.

The size of the several compactions cell is 60 by 54 lambda which is a decrease from the original compaction of about 20 percent.

An example of cell that is very large in scope for a leaf cell is shown in figure 7-6. This cell is a one bit self timed adder cell that uses ternary signaling. The top ports and bottom ports have been constrained to be equal so that the leaf cells may be stacked to

any size self timed adder desired.

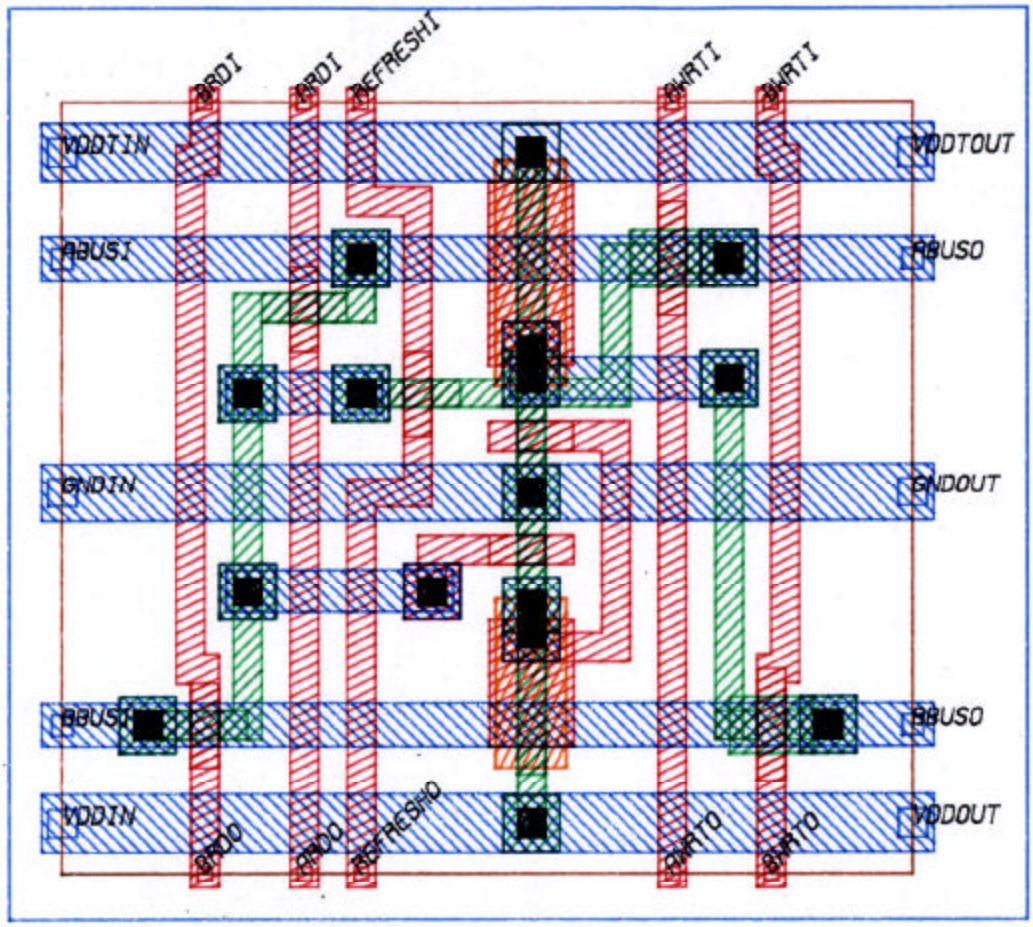


Figure 7-5: Several Compaction with Jogs Two Bus Register Cell

any size self timed adder desired.

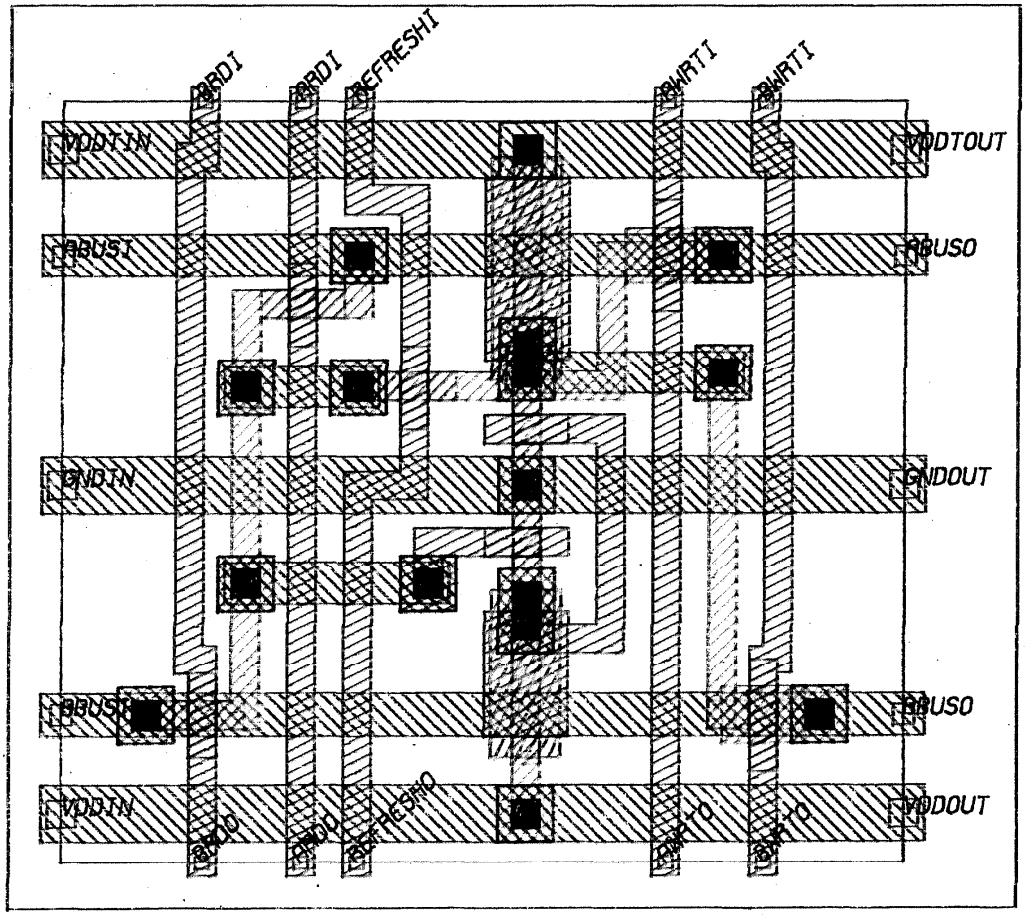


Figure 7-5: Several Compaction with Jogs Two Bus Register Cell

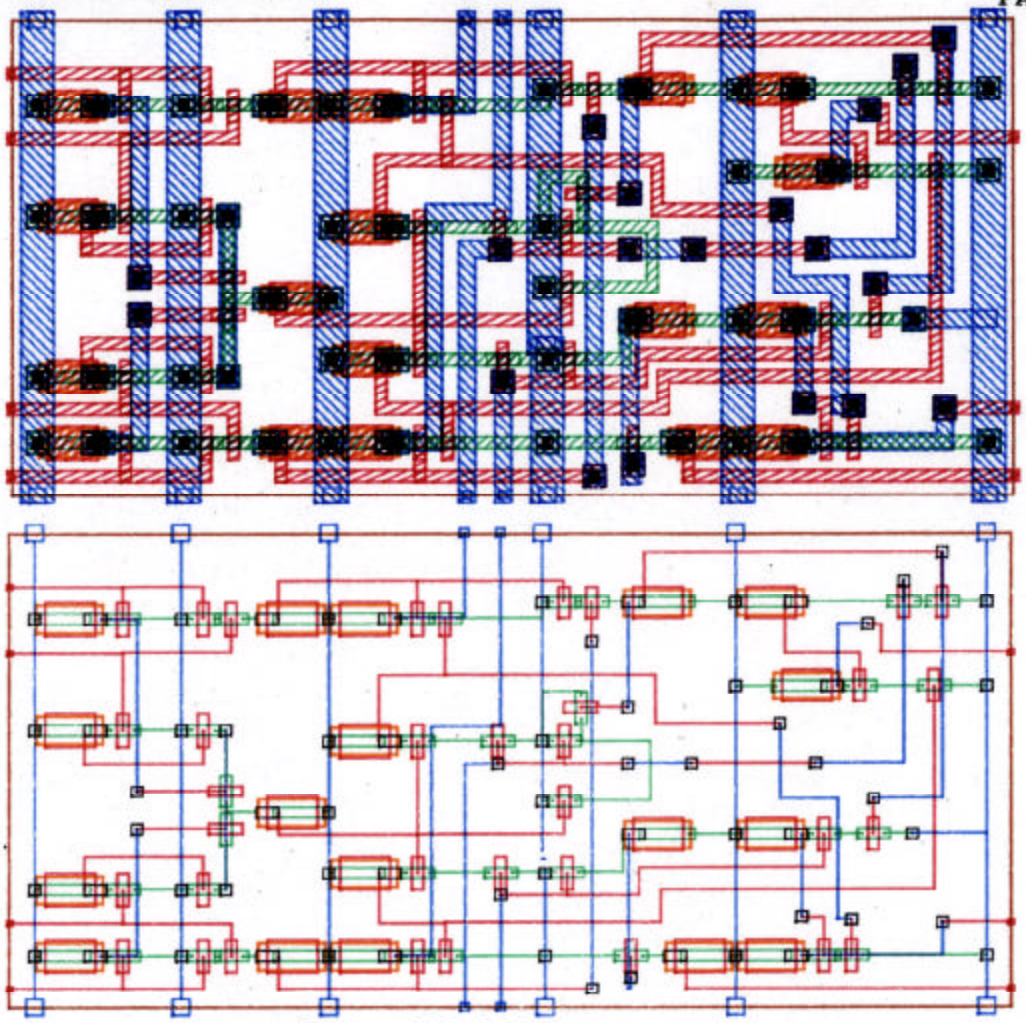


Figure 7-6: Self Timed Adder Cell

7.2 Manual Vs Automatic

The following figure 7-7 shows the two bit register cell as designed before compared to a hand layout version. The differences in the cells are that the hand layout cell allows 45 degree lines, path transistors and polygon transistors. It is not surprising that the hand layout is smaller.

The size of the of the manual layout is 54 by 53 lambda while the sticks is 60 by 54 lambda. The sticks cell is about 10 percent bigger than the manual cell. This is not

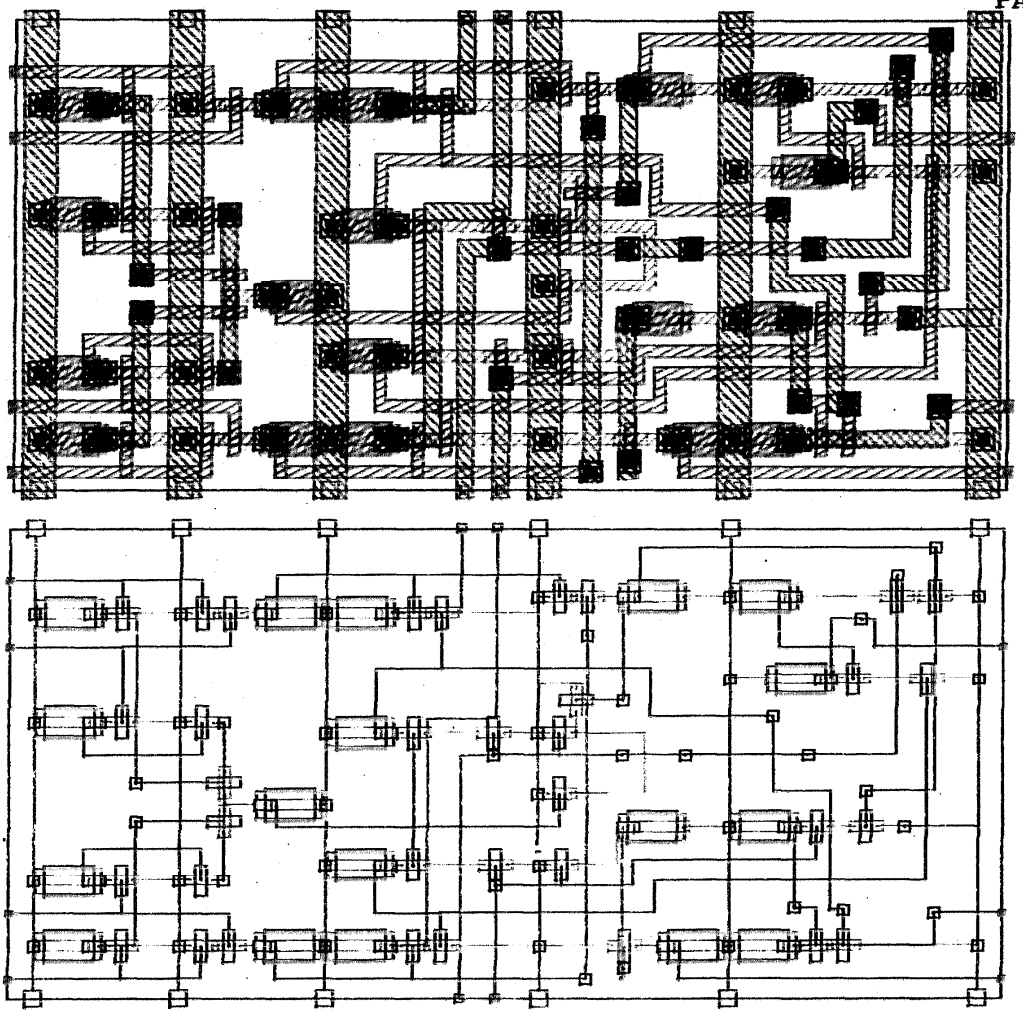


Figure 7-6: Self Timed Adder Cell

7.2 Manual Vs Automatic

The following figure 7-7 shows the two bit register cell as designed before compared to a hand layout version. The differences in the cells are that the hand layout cell allows 45 degree lines, path transistors and polygon transistors. It is not surprising that the hand layout is smaller.

The size of the of the manual layout is 54 by 53 lambda while the sticks is 60 by 54 lambda. The sticks cell is about 10 percent bigger than the manual cell. This is not

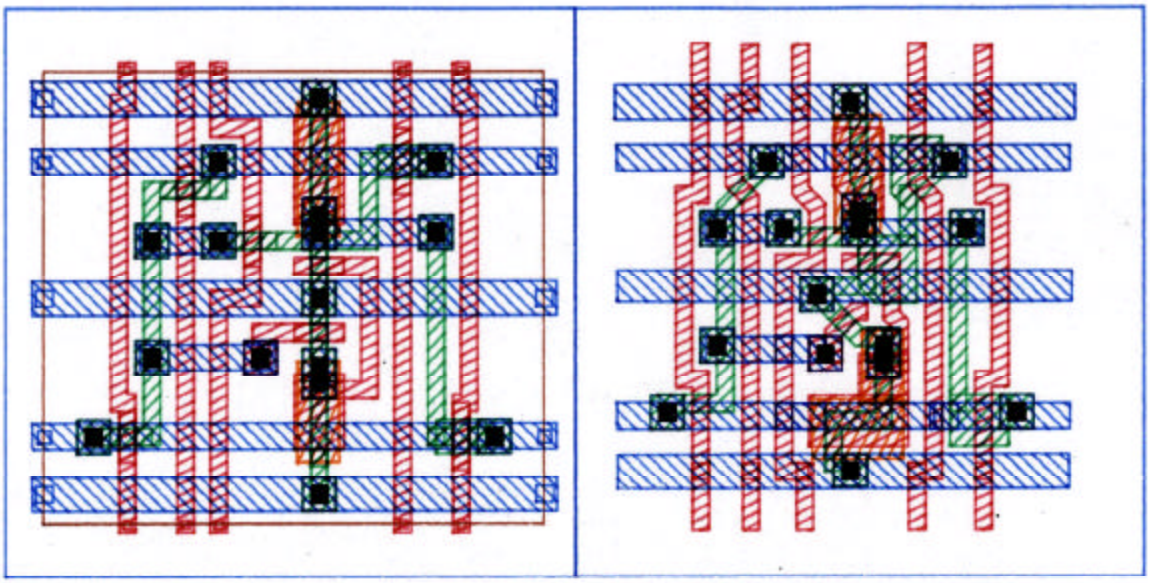


Figure 7-7: Two Bus Register Cell

surprising due to the flexibility of the hand layout. The hand layout design time was greater than that of the sticks and it required a large number of iterations. In addition the sticks cell can easily be changed. That is, the circuit can be changed, the ratios of devices can be changed or the wire width can be changed in a short time. Also the sticks cells is design rules correct by virtue of the spacing routines while the hand layout may not be. The hand layout would require several iterations through a design rules checker to determine the errors and through an additional digitize phase.

This example contains universal features found in most cells design with the

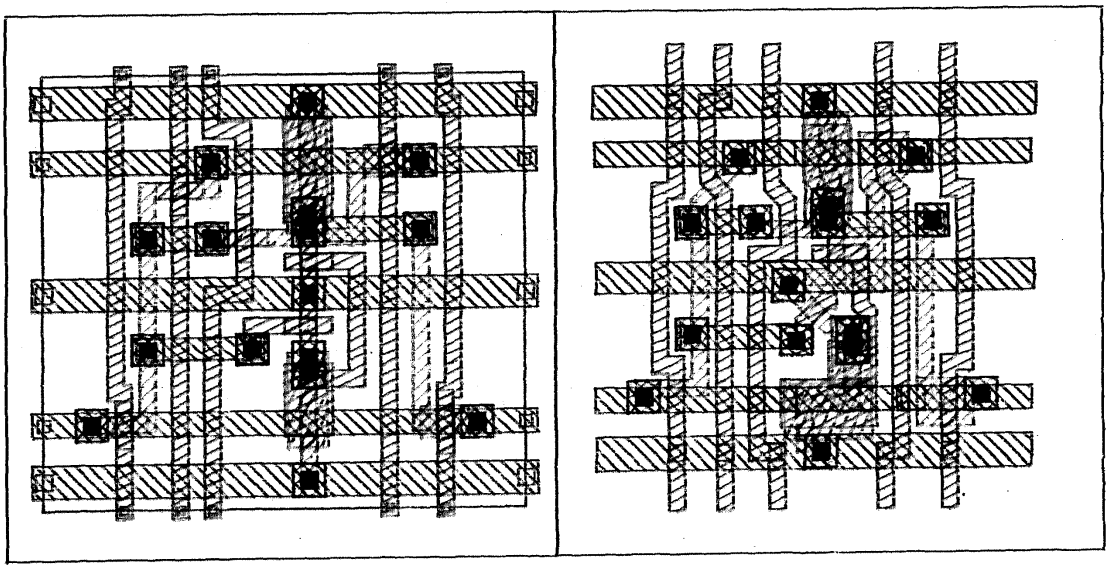


Figure 7-7: Two Bus Register Cell

surprising due to the flexibility of the hand layout. The hand layout design time was greater than that of the sticks and it required a large number of iterations. In addition the sticks cell can easily be changed. That is, the circuit can be changed, the ratios of devices can be changed or the wire width can be changed in a short time. Also the sticks cells is design rules correct by virtue of the spacing routines while the hand layout may not be. The hand layout would require several iterations through a design rules checker to determine the errors and through an additional digitize phase.

This example contains universal features found in most cells design with the

philosophy as described in section 1 page 2. Typical cells designed with this type of philosophy have data, control and power busing structure. In addition there is some internal logic. This cell contains two data buses, three power buses, three control buses, pull-up and pull-down transistors that are typical of a regular leaf cell. Therefore this is a reasonable cell for a comparison.

7.3 Conclusions

VLSI leaf cells can easily be designed with REST in far less time than hand laid out cells. In contrast to hand laid out cells REST cells are design rules error free. The size of a REST cell is about 10 percent bigger that a hand laid out cell. This is not much of a penalty considering the benefits.

The REST design cell wins in conveniences and speed of task during the cell update process. With the hand laid out cell the tedium of digitizing and design rules checking are required for each iteriation of the cell update. Using REST we need only to edit the cell.

It has been shown that wire model[6] used in REST simplifies the connection of wires of different widths both in sticks and as a final output to CIF. This model has reduced a complex wire problem of non-standard width to dealing with wires as paths and width.

The affinity factor[7] can reduce the total polysilicon and diffusion wire length as a trade of metal wire length on some cells. The affinity factor also shortens wires in general by moving floating features to the most attracting feature based on the intervening connection. By changing the weighting[8] of the affinity factors other trade-offs may be defined.

---

[6] See section 3.2.1 page 14 for a description of the wire model.

[7] See section 4.3 page 30 for a description of the affinity factor. Also see section 6.2 page 43 for a description of the algorithms.

[8] The affinity factor weighting is defined in the rules tables.

The compaction algorithms presented herein and used in REST is linear with time due to several techniques presented in section 6.2 page 6.2. These methods do not limit the ability of the compaction process, giving the users a quick response necessary in an interactive environment.

REST is currently being used on several chip designs at Caltech. Several leaf cells for a graphics project chip have been designed using REST. There are several students using REST on this project with very satisfactory results. An additional project is an ethernet chip, also using REST. REST has provided a reduction of design time in these projects. REST has been used extensively at Caltech with great success.

Future investigation in Sticks will be centered on three tasks. First, allowing non-orthogonal lines with curved wires. Second, investigation of compaction in the 2-dimensional plane. Third ,exploring typing on ports and components with a trend toward automatic definition of power widths, transistors parameters, and other property alteration. In addition to explorations in sticks, composition tools will be investigated. Both graphically and textual user specifications will be explored. Future work in sticks and composition will be consistent with the general design philosophy presented in this thesis.

CALIFORNIA INSTITUTE OF TECHNOLOGY

Computer Science Department

Silicon Structures Project

R E S T

Users Guide

by

R.C. Mosteller

I. R E S T Users Guide

I.1 GENERAL

Richards editor for Sticks (REST) is a leaf cell design system for the creation, editing and compaction of NMOS stick diagrams. REST only deals with leaf cells that contain primitive elements: transistors, contacts and connectors. A leaf cell is one that contains primitive elements and does not contain instances to other cells. It is intended that the leaf cells created by REST will be used as primitive building blocks in various Silicon Compilers: like Bristle Blocks[Johnannsen 79], and chip assembliers SPAM [Segal 80].

The REST system uses the Box Editor that runs in the Charles graphic station as the front end for editing. The stick sketches while REST proper digests the stick sketch, does compaction, utility plotting and leaf cell data management.

REST processes sticks sketches that are orthogonal. The transistor models that REST uses are the same as in Mead and Conway [MEAD 80]. Non-orthogonal lines are not allowed in REST, nor are other device models. Although this is a restriction, it is not severe. For general digital chips about 80 percent of the primitive cell could be designed in REST.

I.2 The Environment -- Hardware and Software

Rest is partitioned into two programs - the Box Editor which runs in the LSI-11 that is part of the Charles graphic station, and the main sticks processor which runs in the DEC-20. The graphic interface is provided by the Charles graphic station which consists of a raster scan display, HP plotter, and a Xerox mouse. See figure I-1. The Charles terminal, box editor and interfaces are described in the Charles Terminal Care Package[Minter 80, Burke 80]. The box editor provides the primitive editing functions for the sticks sketch. The command input to the Box Editor is provided by the attached Xerox mouse.

The second program, REST proper, which runs on the DEC20, does the major part of the processing. REST is written in SIMULA and it uses the general SSP graphic software package [Wipfli 78].

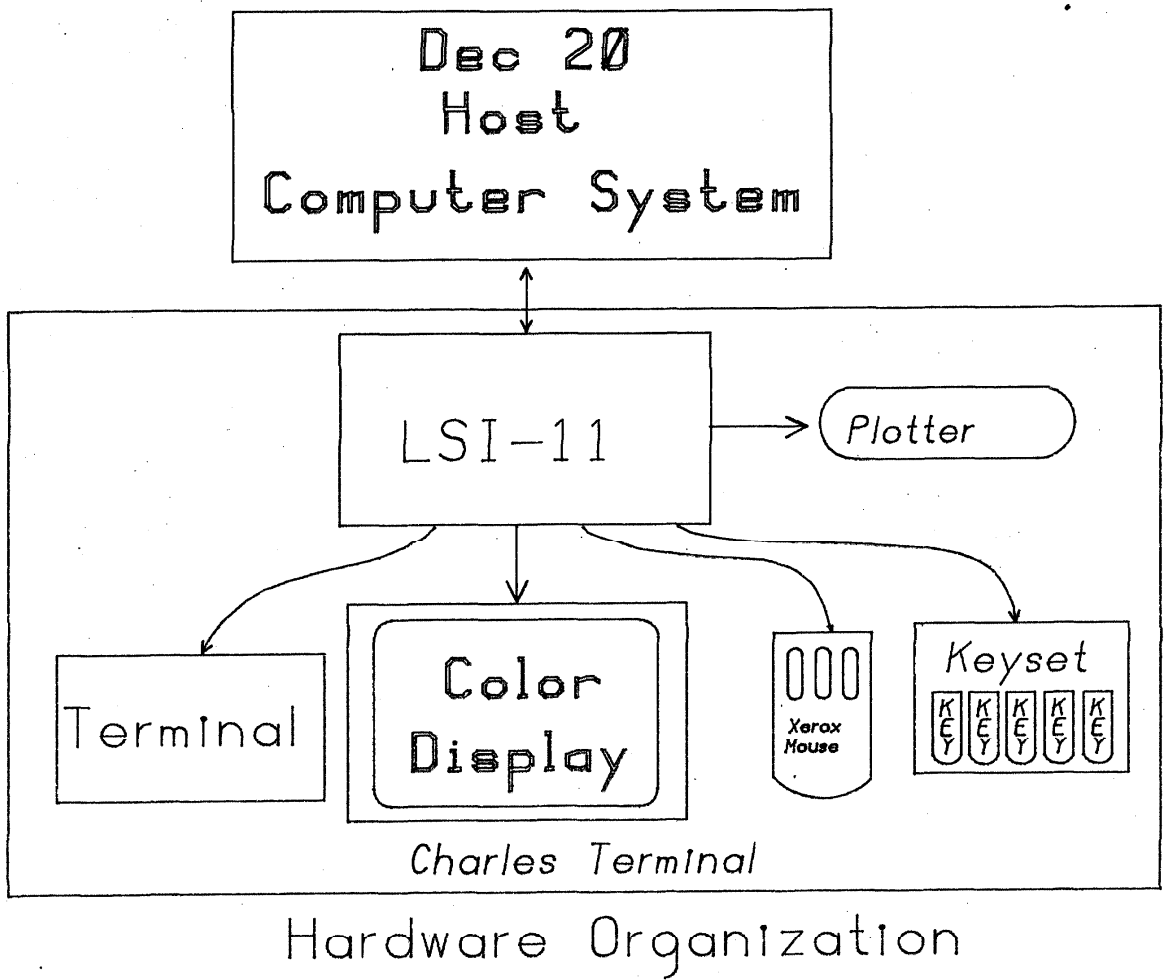


Figure I-1: Picture of the Hardware

I.3 FEATURES

The editing function is centralized in the Charles Graphic Station for optimum performance where it is needed. This separation of stick editing and processing also allows various other input media for the sticks sketch. After each graphic editing function a recovery file is written to disk so that if there is a system failure recovery is possible.

The sticks sketch which comes from the Box Editor may be very crude. It is not necessary to be exact in drawing line interconnections, transistors or contacts. REST

will snap lines to adjacent lines of like color, and recognize generally sloppily designed transistors. See figure I-2 for a sample input to the box editor. Notice that in the figure the sketch is very crude. The cells that are drawn are shift register cells similiar to the examples in Mead and Conway[MEAD 80].

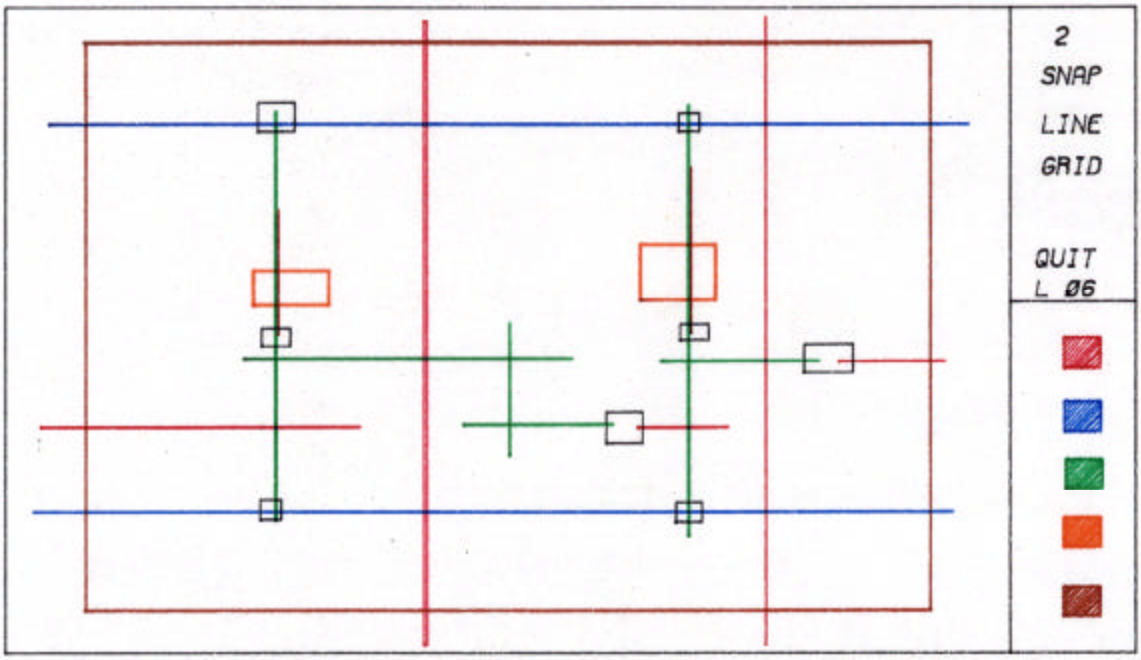


Figure I-2: Initial Input for Box Editor

The edited sticks sketch is isomorphic to the sticks internal representation. This allows editing after compaction to further reduce the size of the cell. The intrepreted sticks drawing is shown in figure I-3. The figure shows both the sticks drawing and the physical representation. The compacted sticks drawing is shown in figure I-4. The figure shows both the sticks drawing and the physical representation. In general use, one creates an initial cell, compacts it, then follows with an edit session and repeated compaction until an acceptable degree of compaction is achieved.

REST is oriented toward amplifying what the man can do and what the computer can do best. That is, the man defines the topological sticks sketch while the program

will snap lines to adjacent lines of like color, and recognize generally sloppily designed transistors. See figure I-2 for a sample input to the box editor. Notice that in the figure the sketch is very crude. The cells that are drawn are shift register cells similiar to the examples in Mead and Conway[MEAD 80].

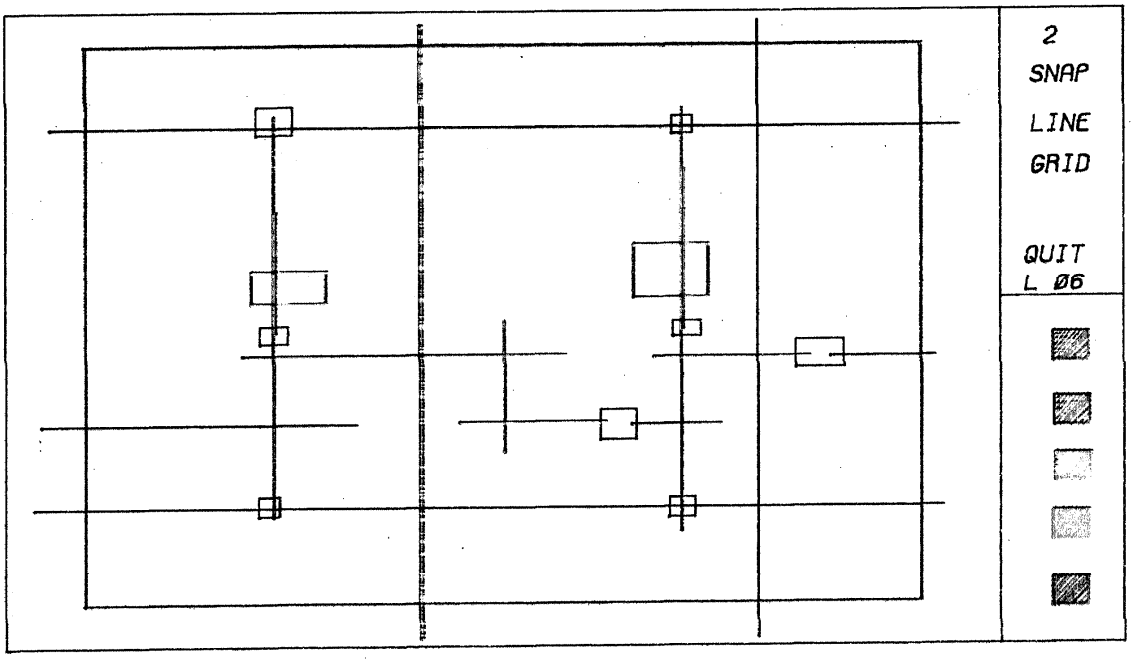


Figure I-2: Initial Input for Box Editor

The edited sticks sketch is isomorphic to the sticks internal representation. This allows editing after compaction to further reduce the size of the cell. The intrepreted sticks drawing is shown in figure I-3. The figure shows both the sticks drawing and the physical representation. The compacted sticks drawing is shown in figure I-4. The figure shows both the sticks drawing and the physical representation. In general use, one creates an initial cell, compacts it, then follows with an edit session and repeated compaction until an acceptable degree of compaction is achieved.

REST is oriented toward amplifying what the man can do and what the computer can do best. That is, the man defines the topological sticks sketch while the program

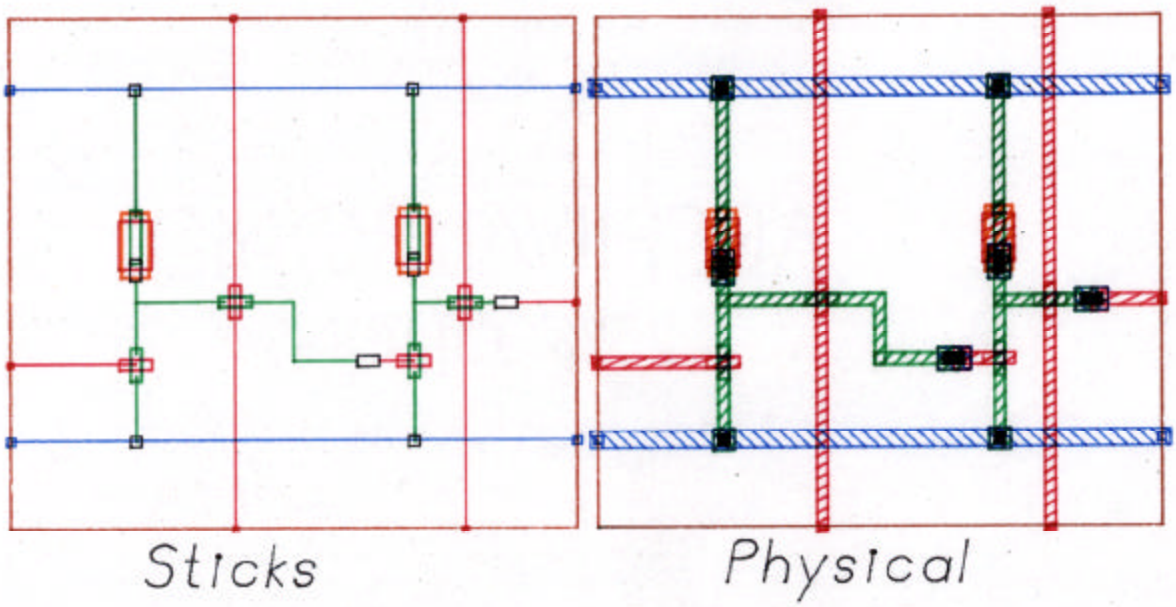


Figure I-3: Intrepreted Sticks Drawing

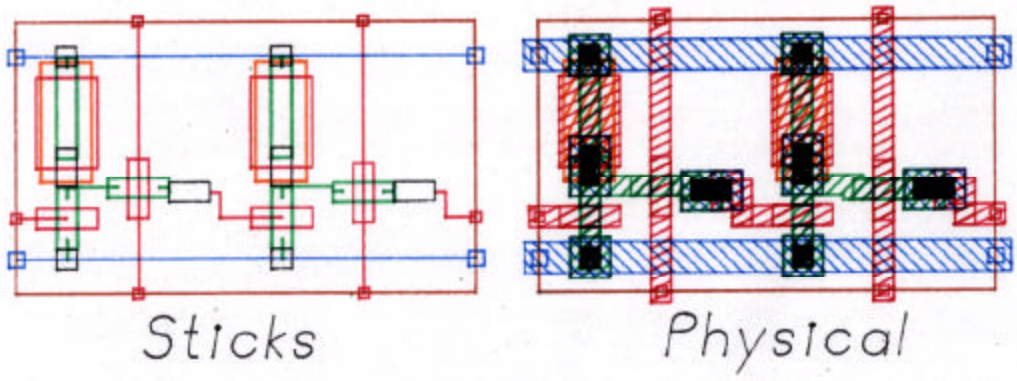


Figure I-4: Compacted Sticks Drawing

compresses the sticks diagram to minimum size. REST does not make topological changes.

REST interfaces a generalized sticks representation file called "The Sticks Standard"

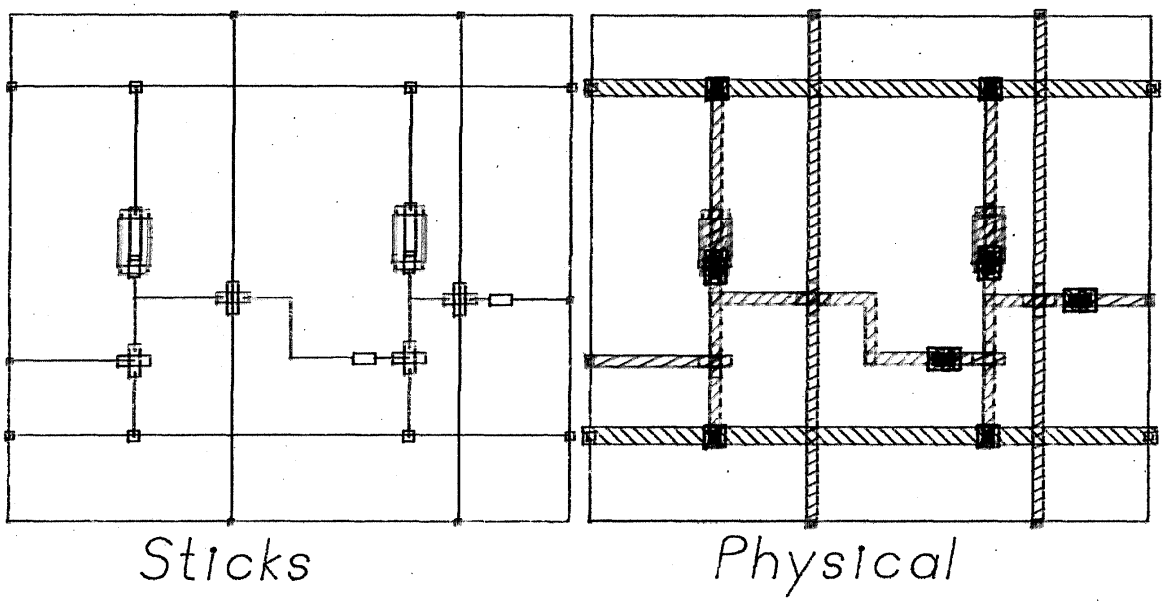


Figure I-3: Intrepreted Sticks Drawing

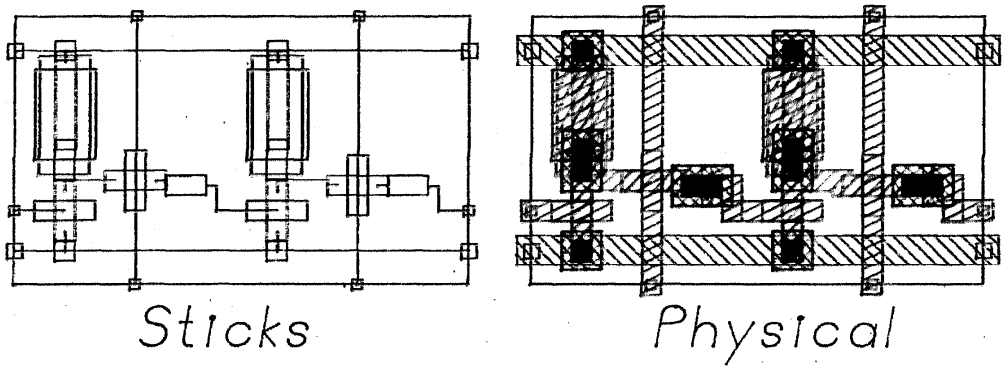


Figure I-4: Compacted Sticks Drawing

compresses the sticks diagram to minimum size. REST does not make topological changes.

REST interfaces a generalized sticks representation file called "The Sticks Standard"

which can be read in by various Silicon Compilers for chip assembly[Trimberger 80]. This file could also be used for porting designs.

I.4 STICKS INTERPRETATION

The input to REST from the box editor consists of colored lines and boxes. These lines and boxes are interpreted by REST as transistors, contacts, and connectors. All lines and boxes sent to REST must be orthogonal, others will be thrown away with appropriate error messages. There are three major colors: RED, GREEN, and BLUE. These colors are interpreted as polysilicon, diffusion, and metal respectively. The subordinate colors are YELLOW which denotes an implanted transistor, and GREY which denotes a contact. See figure I-5 for a drawing of the models both in sticks and physical. The modeling of stick devices and the colors used in REST are consistent with Mead and Conway[MEAD 80].

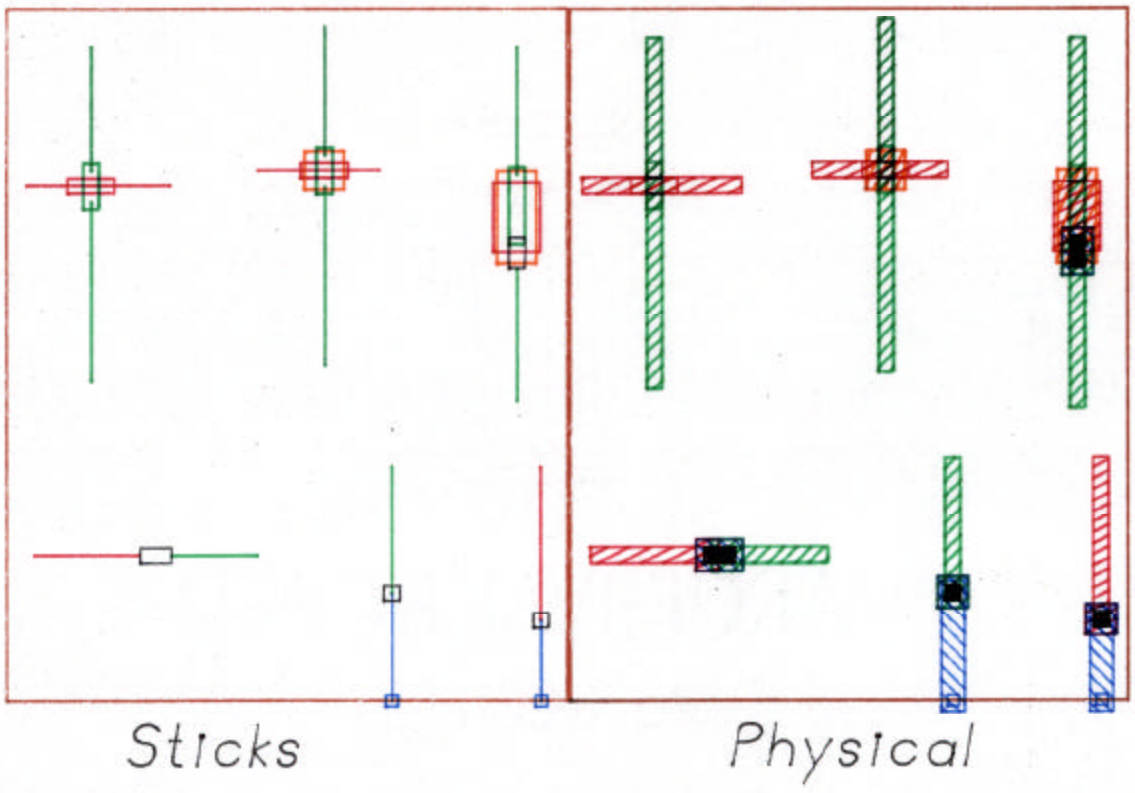


Figure I-5: Sticks Models

which can be read in by various Silicon Compilers for chip assembly[Trimberger 80]. This file could also be used for porting designs.

I.4 STICKS INTERPRETATION

The input to REST from the box editor consists of colored lines and boxes. These lines and boxes are interpreted by REST as transistors, contacts, and connectors. All lines and boxes sent to REST must be orthogonal, others will be thrown away with appropriate error messages. There are three major colors: RED, GREEN, and BLUE. These colors are interpreted as polysilicon, diffusion, and metal respectively. The subordinate colors are YELLOW which denotes an implanted transistor, and GREY which denotes a contact. See figure I-5 for a drawing of the models both in sticks and physical. The modeling of stick devices and the colors used in REST are consistent with Mead and Conway[MEAD 80].

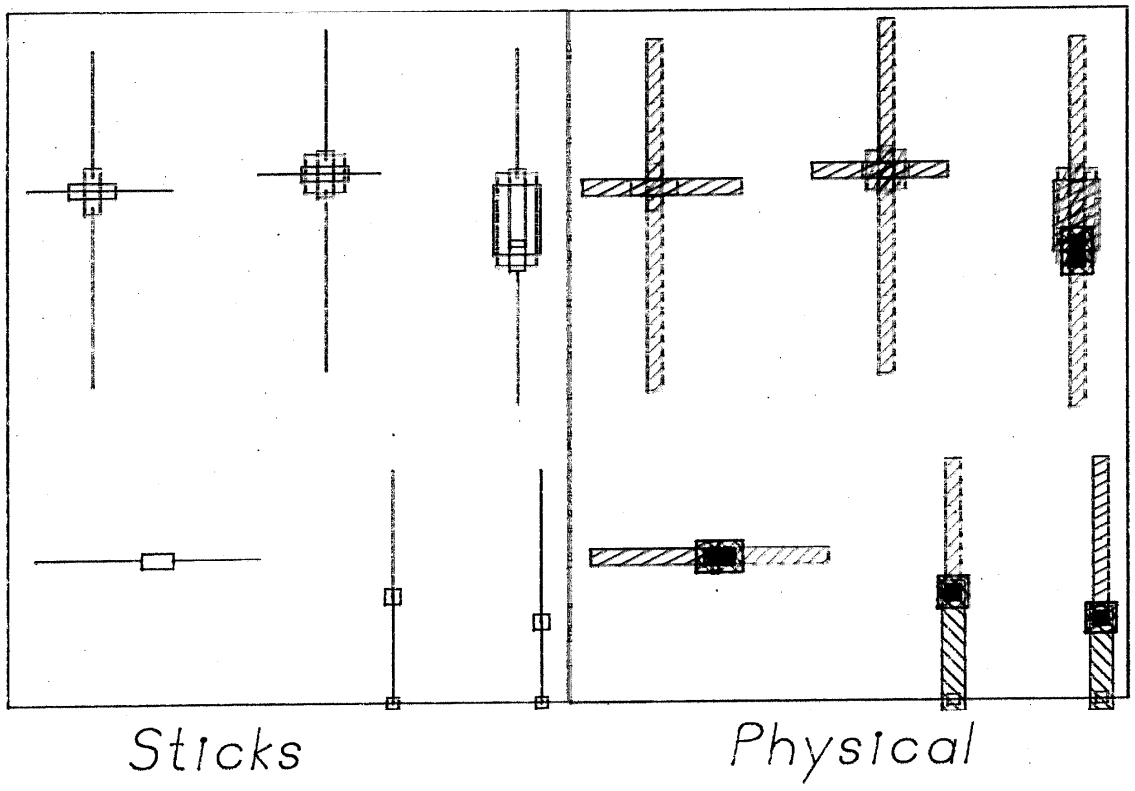


Figure I-5: Sticks Models

A RED line crossing a GREEN line is interpreted as an enhancement mode transistor. The RED line need only touch the GREEN line to create a transistor. The length and width of a newly created enhancement transistor are taken from the defaults which may be set by the user.

A RED line crossing a GREEN line with a YELLOW box or a YELLOW line on top of the intersection is interpreted as a depletion mode transistor. As with the enhancement transistor, the length and width for newly created depletion transistors are taken from the defaults.

A RED line colinear with a GREEN line with a YELLOW box or a YELLOW line on top of the intersection is interpreted as a pullup type model, provided that there is a contact on one end of the RED line. The RED line must be shorter than the GREEN. As in the case of the enhancement transistor, the length and width for a newly created pullup type model are taken from the defaults.

General Contacts can be created by placing a contact box on top of the lines that are to be connected. Normal contacts are created for BLUE to RED or BLUE to GREEN. A butting contact is created for RED and GREEN. Butting contacts have the property that the RED and GREEN lines may not cross. If this occurs an appropriate error message with be given and one of the connecting lines will be disconnected from the butting contact. The line that is disconnected from the butting contact is chosen based on what is connected to its other end. An end with nothing attached will be the first choice.

I.5 FUNCTIONS

Rest is partitioned into five major functions: line drawing interpreter, spacing, expansion, cell management, and utilities.

The line drawing interpreter processes the crude sticks sketch coming from the Box Editor. Lines of the same color or layer that are in close proximity are recognized as connected. Enhancement transistors are recognized as RED crossing GREEN lines. It is not necessary that they cross, but they must be perpendicular. Depletion transistors are recognized when an enhancement transistor is partially covered by a YELLOW box line. Pullup transistors are recognized as a RED line colinear to GREEN with YELLOW on top

and a contact box at one end of the RED line. Connection points are defined by lines close to or crossing the bounding box. In addition, after the stick sketch is interpreted, it is compared to the prior diagram to transfer annotation, line widths and device ratios.

The spacing function can be used to space the stick diagram to Mead and Conway design rules. This function can be used at any time. The spacing or compaction can be directed to compact along the vertical or horizontal axis. In addition spacing to the left, right, top, or bottom can be performed. If after compaction it is necessary to add extra logic in the cell, it can be expanded to allow room for this logic. The minimum bounding box can be stretched or pulled out at the left, right, top, or bottom. This stretching will also lengthen any attached wires.

REST has commands for reading or writing a sticks diagram file [Trimberger 80]. In addition, a file can be read in that contains lines and boxes. This method of input is provided so that other methods of supplying the crude stick sketch can be used. REST also writes a recovery file in the same format so that if there is a problem after running the box editor a recovery can be done.

REST has a set of utilities for plotting in various formats on the Hewett-Packard plotter, and on the Charles terminal.

I.6 AN EXAMPLE - GETTING STARTED

The executable file for REST can be found on <SSPLIB>REST.EXE. REST requires the help file <SSPLIB>REST.HLP.

The first step is to execute REST by typing in REST to the DEC20. REST will respond with an appropriate greeting. Before editing, a new cell must be created. This is done by typing in new foo where foo is the name of new cell. To edit the newly created cell, type in edit, REST will respond by starting the box editor. You can then design your cell as in figure I-6 which is the raw input from the box editor. The commands to the box editor are summarized in the command section under edit and in the manual [BURKE 80]. Please note in the figure that the drawing is very crude. It is not necessary to be precise in your sketches.

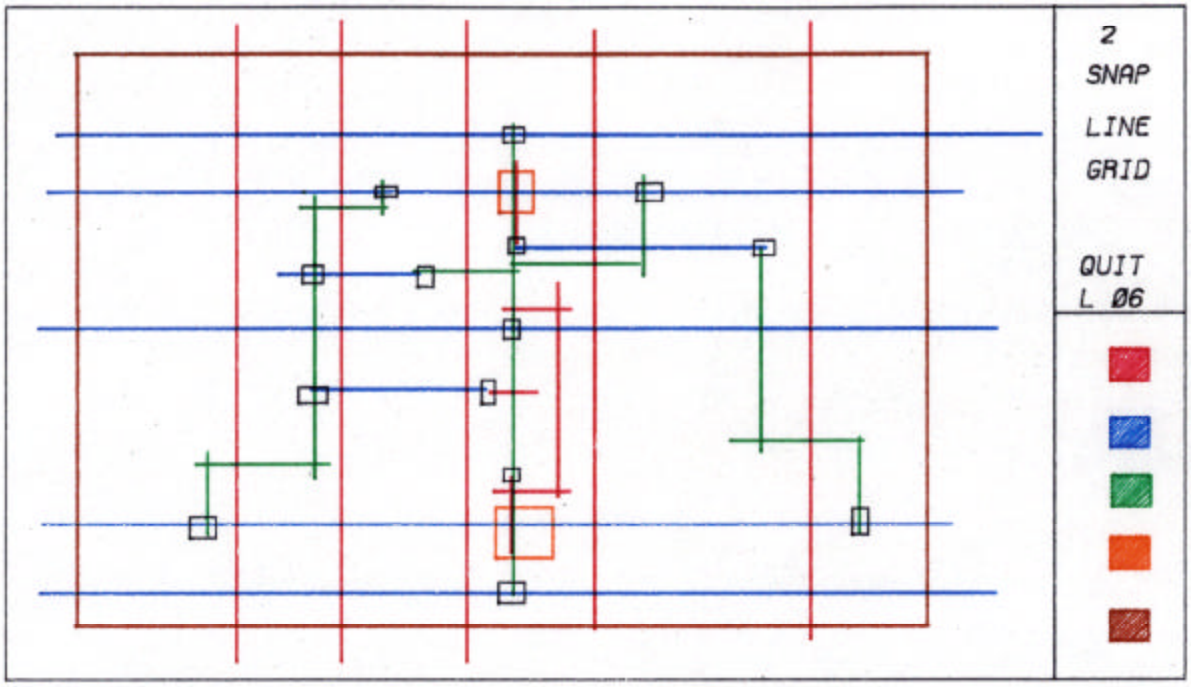


Figure I-6: - Sticks Input - from Box Editor

When the cell is in the desired form, terminate the box editor by using the quit command. REST will then interpret your drawing as in figure I-7. Notice that REST snaps the lines together in an orthogonal manner. There are two commands for cleaning the sticks drawing. First is absorb which combines connected lines, and second trim which will trim off lines that overhang. REST will automatically trim off lines that overhang a little.

Figure I-8 shows the sticks drawing fleshed out. At this time it has not been adjusted for design rules. By using the command pack the drawing will be compressed to design rules as in figure I-9. Packing can be performed in either of two directions - parallel to the X-axis or Y-axis - or both. Figure I-9 shows the cell compacted to design rules in physical form while figure I-10 shows it in stick form.

Device ratios can be changed by using the cursor commands with the set command or the set command with the device name. Figure I-11 shows the cell in physical layout

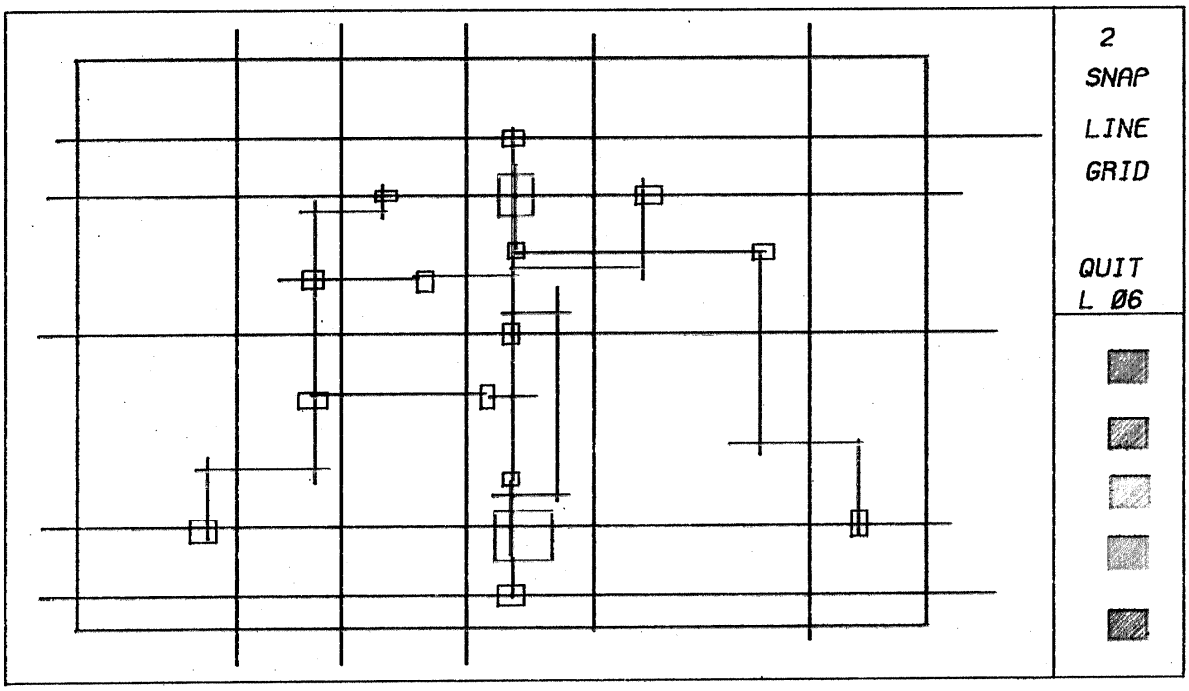


Figure I-6: - Sticks Input - from Box Editor

When the cell is in the desired form, terminate the box editor by using the quit command. REST will then interpret your drawing as in figure I-7. Notice that REST snaps the lines together in an orthogonal manner. There are two commands for cleaning the sticks drawing. First is absorb which combines connected lines, and second trim which will trim off lines that overhang. REST will automatically trim off lines that overhang a little.

Figure I-8 shows the sticks drawing fleshed out. At this time it has not been adjusted for design rules. By using the command pack the drawing will be compressed to design rules as in figure I-9. Packing can be performed in either of two directions - parallel to the X-axis or Y-axis - or both. Figure I-9 shows the cell compacted to design rules in physical form while figure I-10 shows it in stick form.

Device ratios can be changed by using the cursor commands with the set command or the set command with the device name. Figure I-11 shows the cell in physical layout

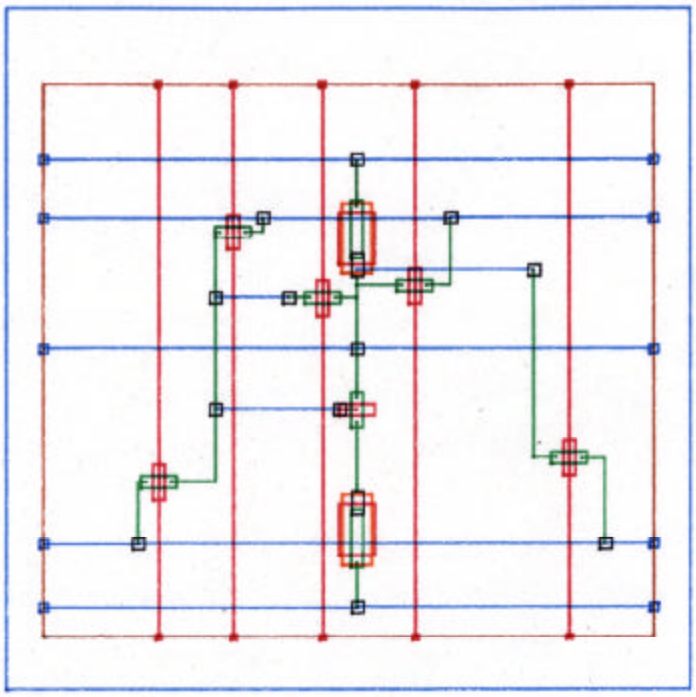


Figure I-7: Sticks After Interpretation

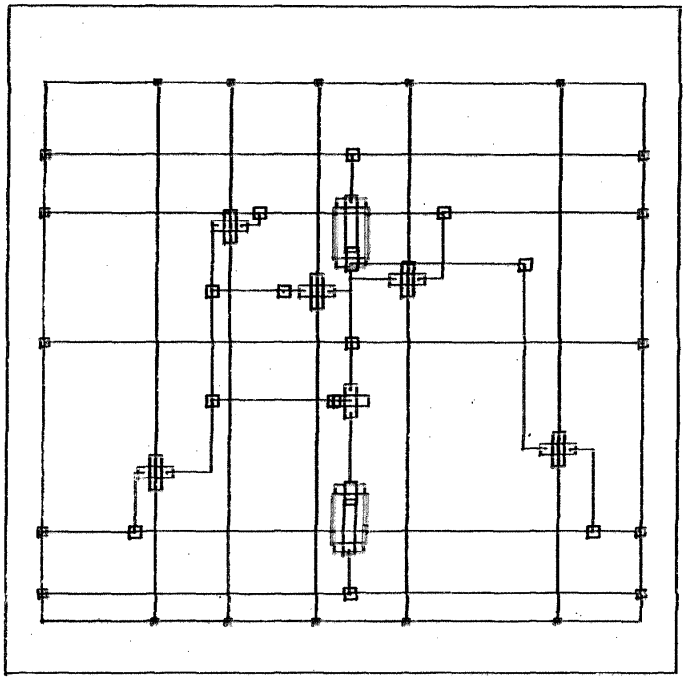


Figure I-7: Sticks After Interpretation

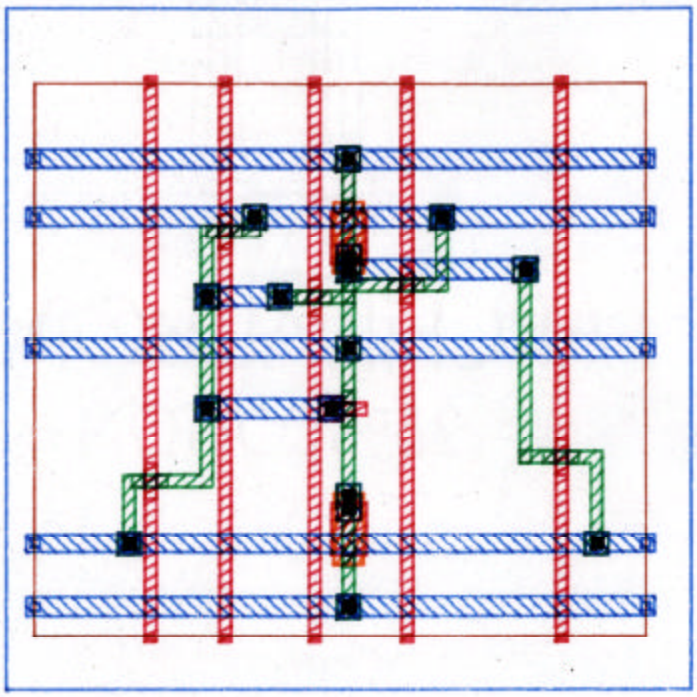


Figure I-8: Sticks Physical Representation

with appropriate name changes with annotation.

The put and get command can be used to save your created cells for later use by REST or other programs that use the sticks standard.

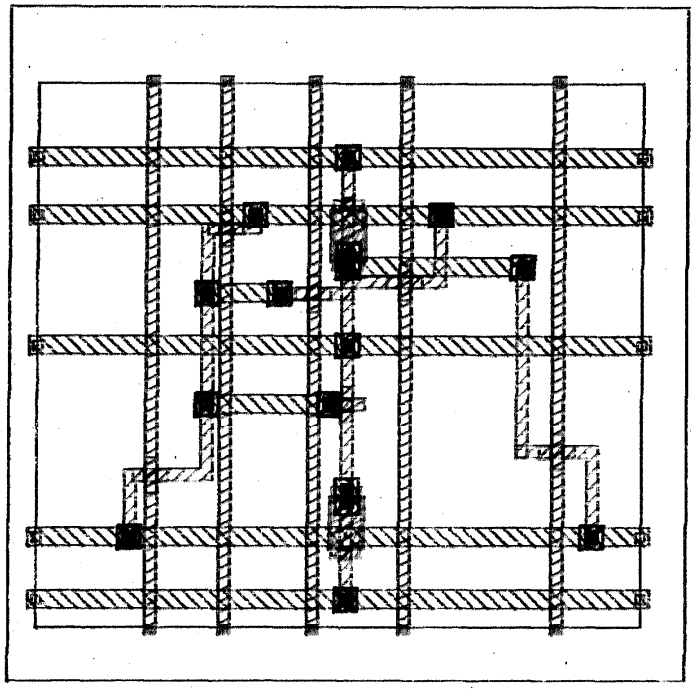


Figure I-8: Sticks Physical Representation

with appropriate name changes with annotation.

The put and get command can be used to save your created cells for later use by REST or other programs that use the sticks standard.

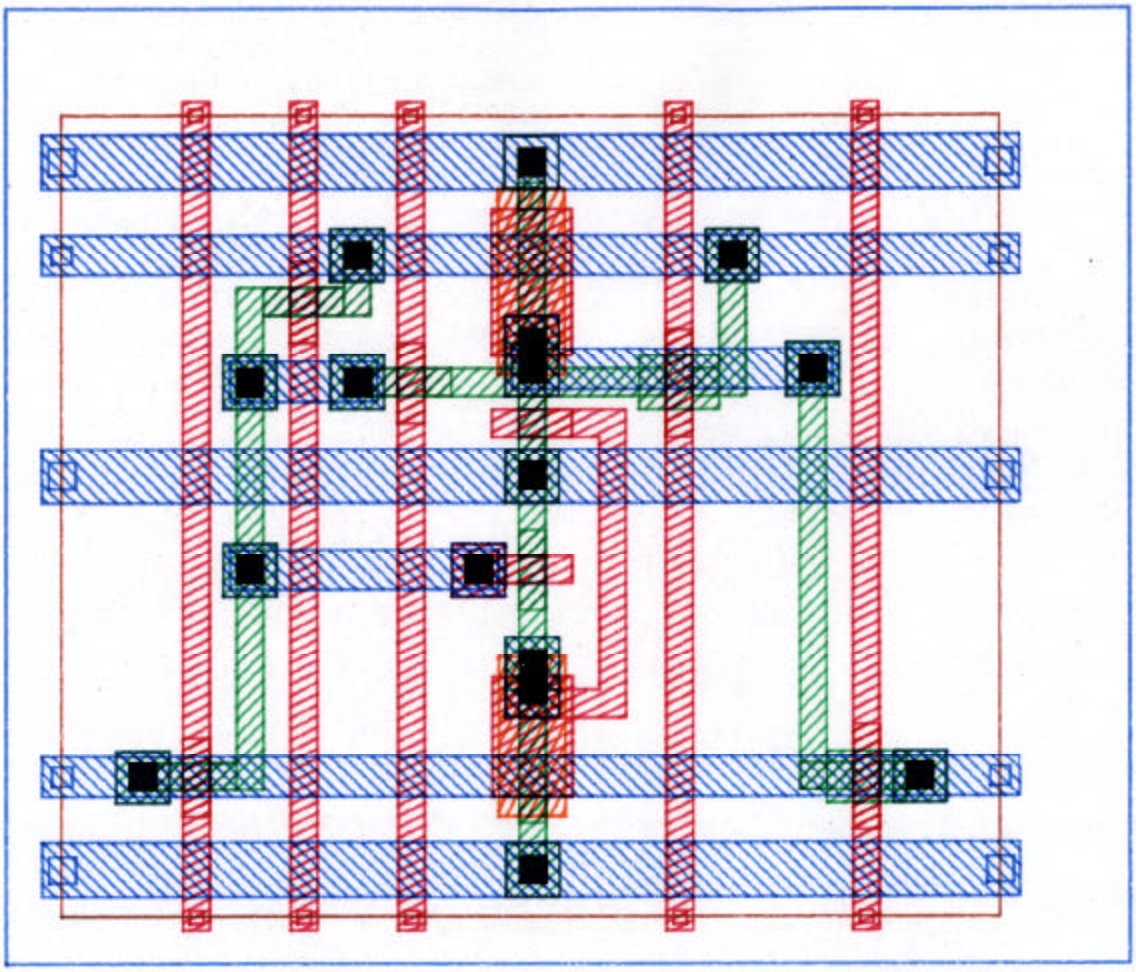


Figure 1-9: Compacted Sticks Physical Representation

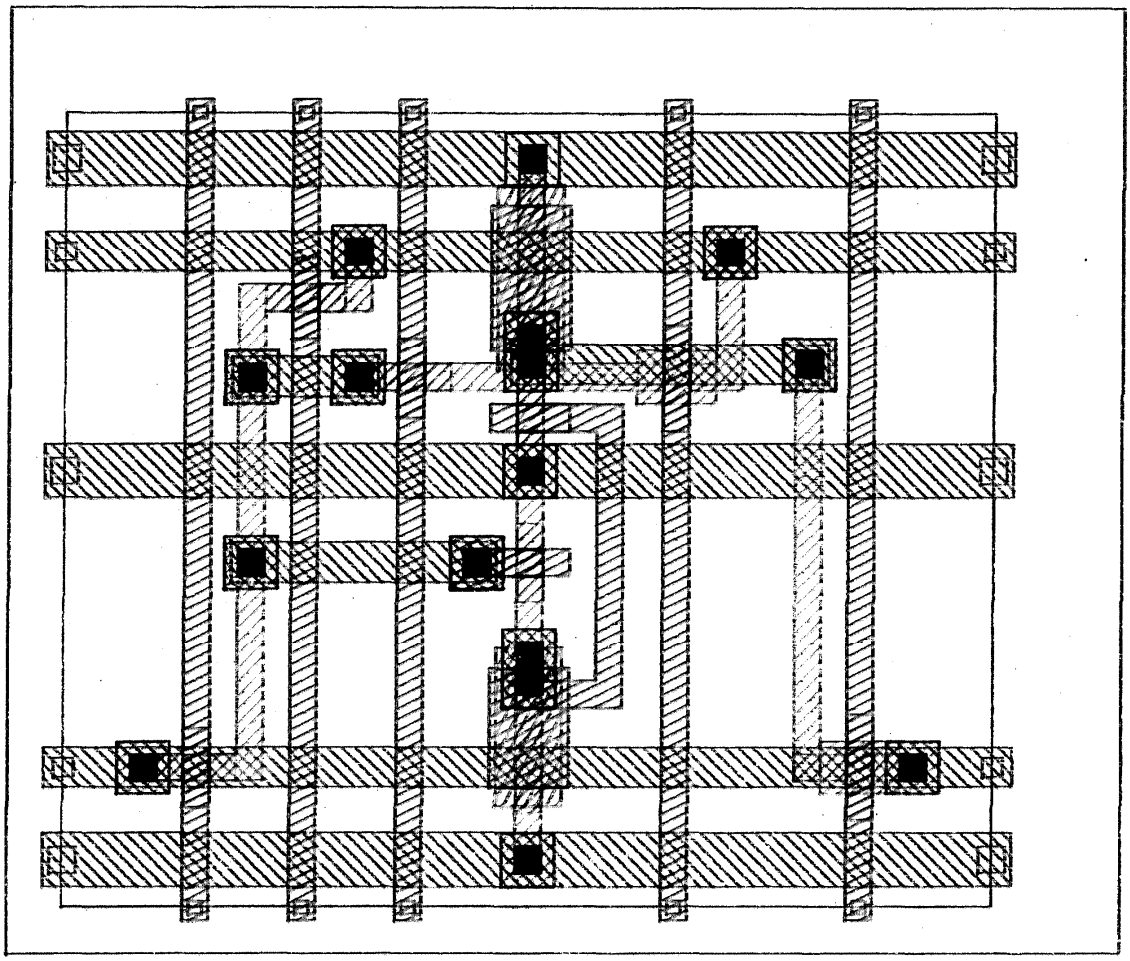


Figure I-9: Compacted Sticks Physical Representation

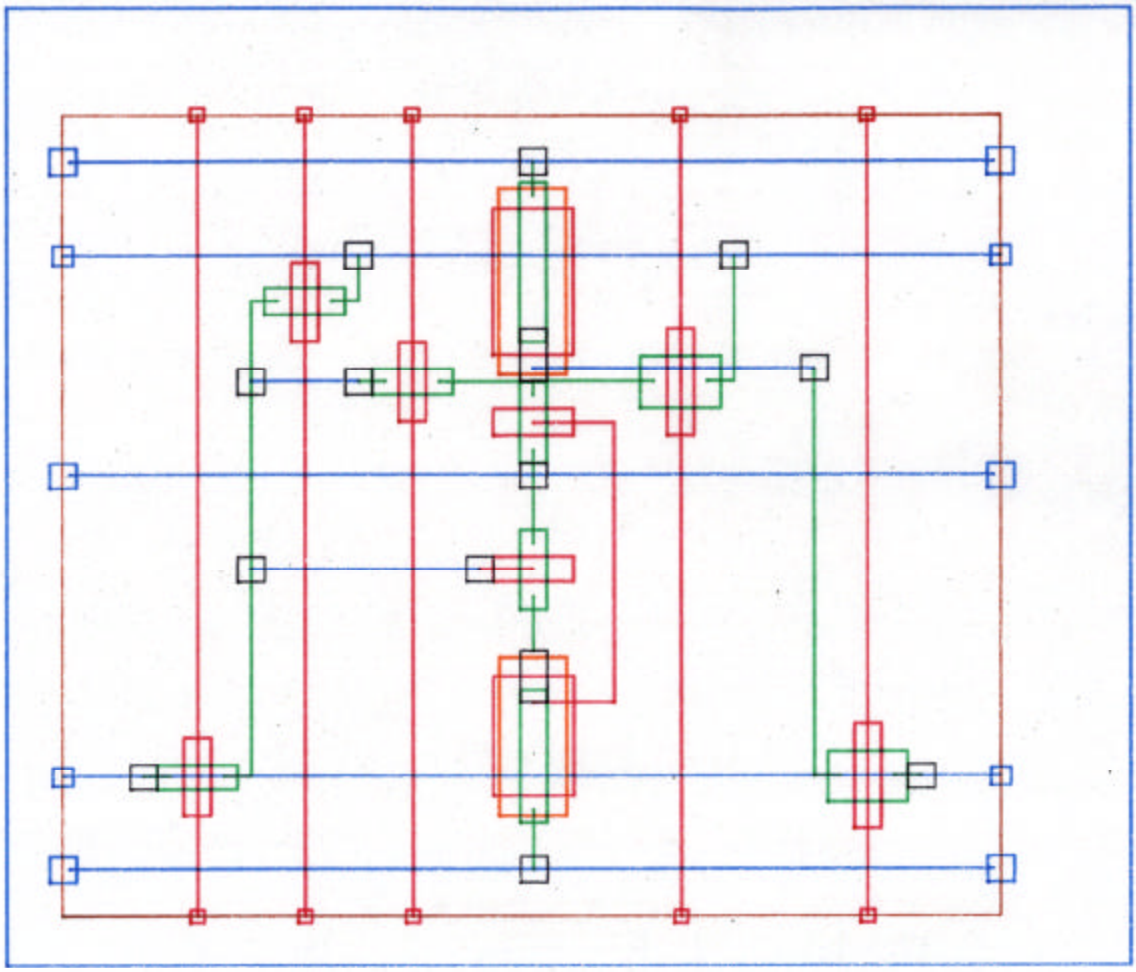


Figure I-10: Compacted Sticks

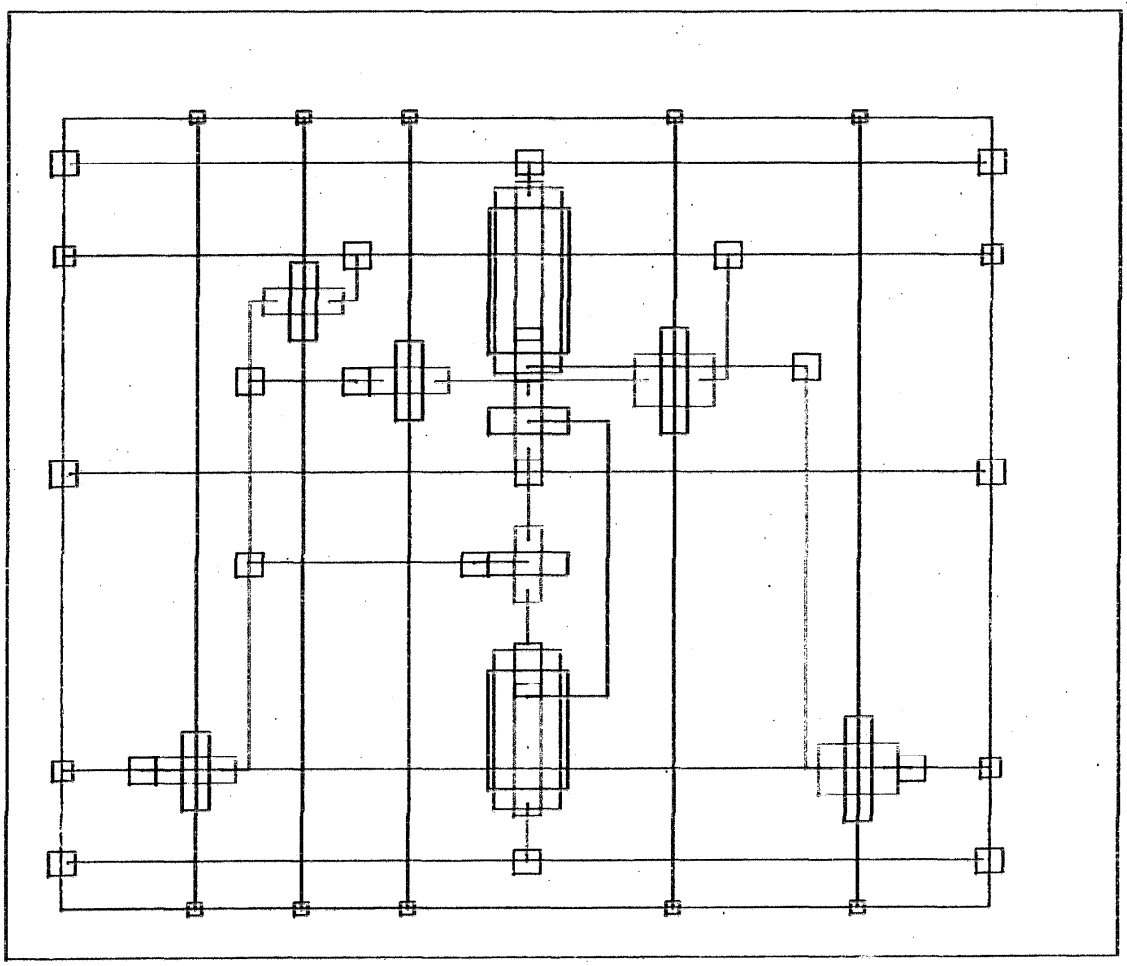
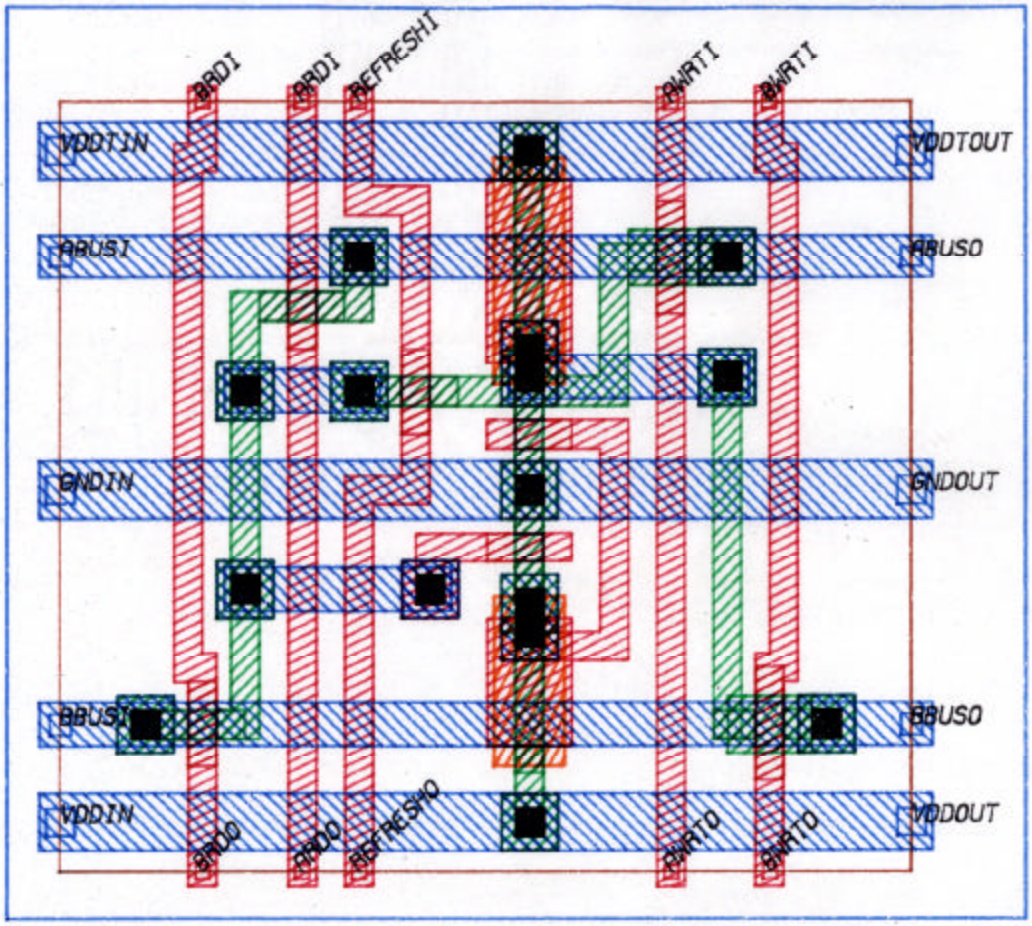


Figure I-10: Compacted Sticks

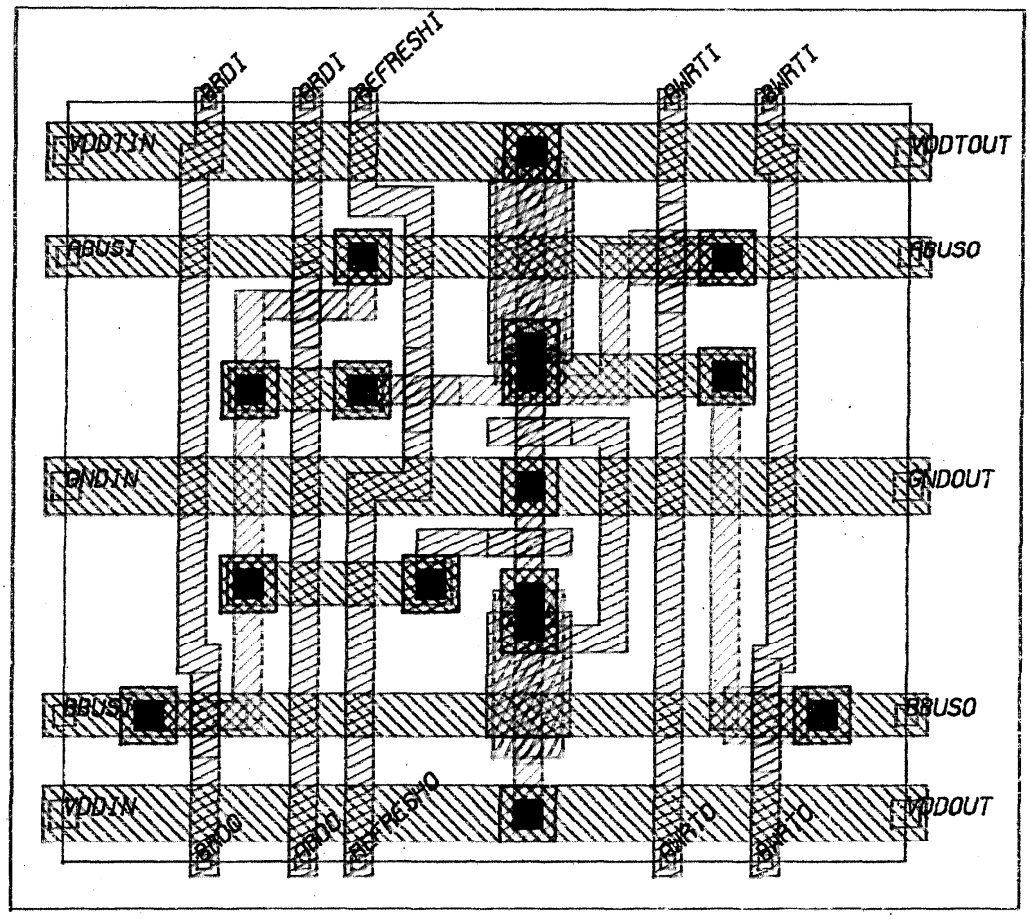


Several Compactions, and Constraints

Figure I-11: Annotated Sticks Physical Representation

I.7 REST INPUT COMMANDS

The commands to REST are described here in a syntactical graph notation. Choices or branches are always followed down while loops are always followed up and back. The number between the symbols /n\ is the maximum number of times the path may be taken. If an asterisk appears between the slashes the path must be taken at least once. In the example below, the possibilities would be A BRANCH , BRANCH ANOTHER



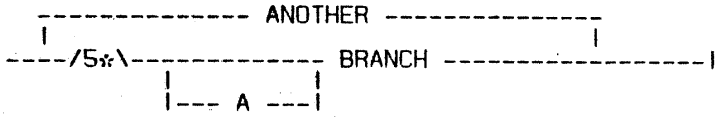
Several Compactions, and Constraints

Figure I-11: Annotated Sticks Physical Representation

I.7 REST INPUT COMMANDS

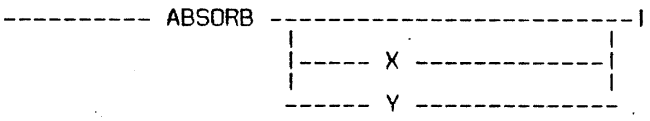
The commands to REST are described here in a syntactical graph notation. Choices or branches are always followed down while loops are always followed up and back. The number between the symbols /n\ is the maximum number of times the path may be taken. If an asterisk appears between the slashes the path must be taken at least once. In the example below, the possibilities would be A BRANCH , BRANCH ANOTHER

BRANCH, etc. The end of the path is always terminated by a |. Keywords are spelled out in capital letters and references to other productions are between the symbols < >.



The following are the commands for REST listed alphabetically. REST accepts one command per input line and does not allow the command to span more than one line. The keywords in the command may be abreviated as long as they are unique among the other commands. All units in REST are in lambda except where explicitly stated.

ABSORB This command causes all connected lines that are not a part of a component (a component is a transistor ,contact, or connector) along a colinear path to be merged into one segment. The X and Y part allow selective absorbsion of segment in the horizontal (X axis) or the vertical(Y axis).

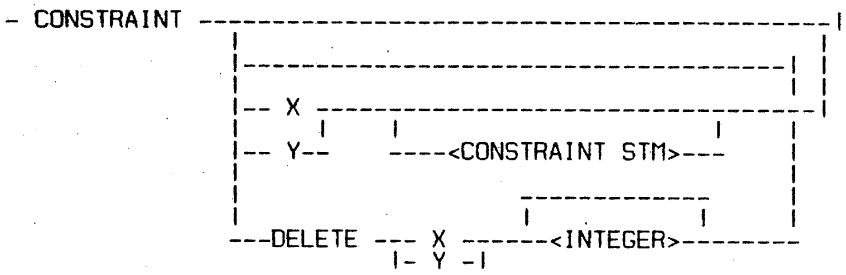


BURIED This command sets the buried option. When set all butting contacts of the current cell will be changed to buried contacts. Any new poly to green contacts will be of the buried type.

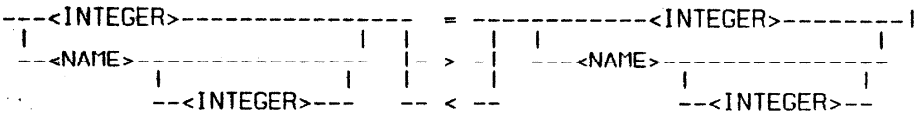
CONSTRAINT

This command allows inquiry and modification to the current cell constraints. A constraint defines proximity rules for component or connector, or segment intersection point. The X or Y defines the axis to which the rule is to apply. If no constraint is supplied the current constraints are listed on the terminal. The delete allows deletion of

constraints. The integer refers to the index value of the constraints after they have been listed.

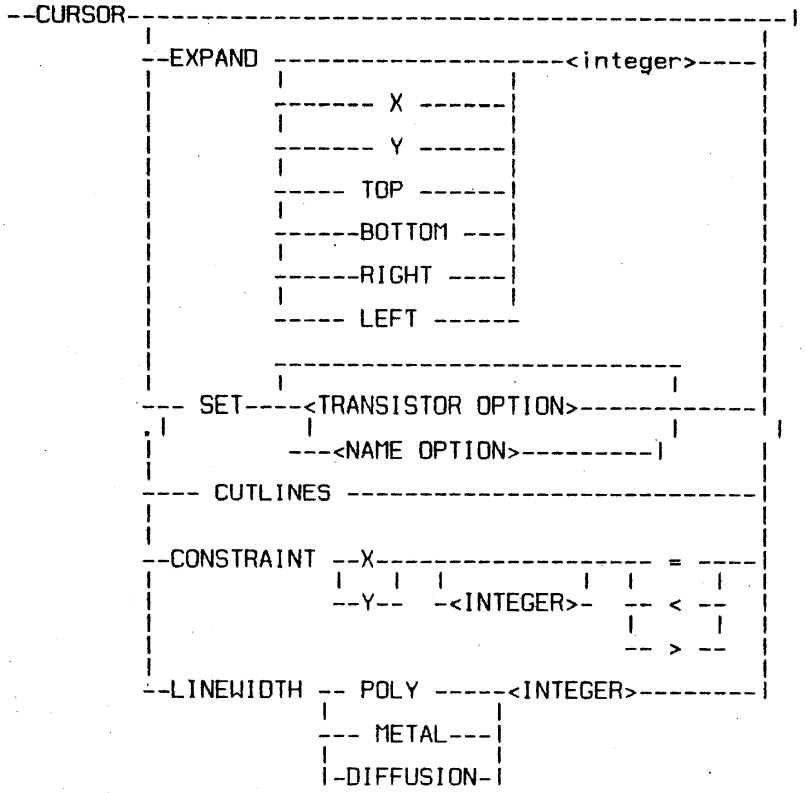


CONSTRAINT STM



The CONSTRAINT STM provides the rule. First either an integer or a component name and an optional integer will be supplied to provide the left hand part of a constraint inequality. Second a relation is supplied followed by an integer or a name and optional integer. REST does all necessary rule checking.

CURSOR This command allows inquiry and modification to the current cell using the Charles Graphic Station. The mouse buttons are numbered from right to left. Button one or the first button is used for inquiry. Button two is used to exit the cursor mode. Button three is used to cause action depending on the type of request. A DISPLAY command must have been done prior to the cursor command.



A general cursor request with no options will allow inquiry only. The first button is used for inquiry while the second will quit.

EXPAND This option will cause expansion around the point selected by the third button. The amount of expansion is defined by the integer in Lambda units while the direction is defined by X, Y, TOP, BOTTOM, RIGHT, or LEFT.

SET This option is the same as the set option in the SET command, however the component to be changed is defined by the third button of the mouse and the position of the cursor.

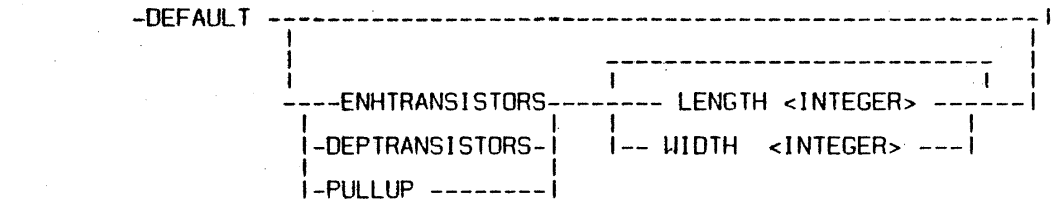
CONSTRAINT

The CONSTRAINT provides a constraint rule. First the axis is defined by the X or Y. Second the optional integer will be used to provide the offset for the left hand part of a constraint inequality. Third, a relation is supplied: integer or a name and an optional integer. The first and second components are defined by two consecutive presses of the third button. REST does all necessary rule checking after

the second button is pressed.

LINEWIDTH The LINEWIDTH option allows setting a segment path to a specific line width. The path is defined by two points with the third button. The POLY, METAL, or DIFFUSION define the layer for the path. The integer value is the width for the path.

DEFAULT This option sets default transistor length and width for the transistor active area. The defaults only apply to newly created transistors. Old transistors will remain at their current parameters. Any time a default is set, values of the defaults are displayed on the terminal. Setting a default to zero clears that specific value.

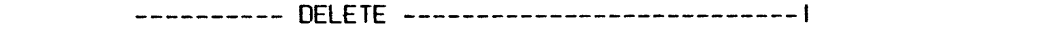


ENHTRANSISTORS
This option defines the default length and width for a new enhancement transistor. New transistors are those created after the default command.

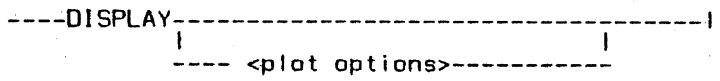
DEPTRANSISTORS
This option defines the default length and width for a new depletion transistor.

PULLUP This option defines the default length and width for a new pullup type device.

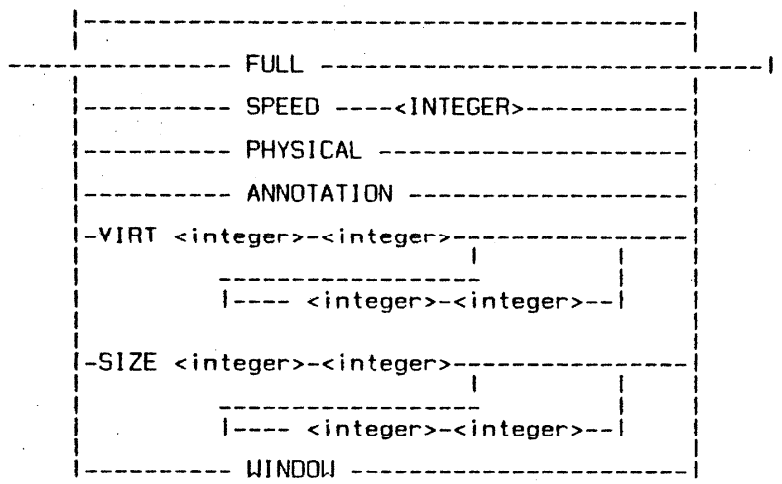
DELETE This command deletes the current cell and allows the user to create a new one.



DISPLAY This command plots the current cell on the Charles terminal. Several options are allowed which are defined in the plot options.



PLOT OPTIONS



ANNOTATION
This option causes the component name to be printed next to the component.

FULL This option allows plotting the cell on a full size sheet on the HP plotter. Note that the normal is 8 by 11.

SPEED This option changes the pen speed on the HP plotter. The valid value for this is between 1 and 36. Slow plot speed gives better transparency slides. About 12 is right for transparencys and the default for normal. The default is speed 36.

PHYSICAL This option plots the stick drawing in real units.

SIZE This options sets the plotting window. If two sets of coordinates are given, they define the window. If one coordinate is given it defines a window where the low point is negative of the point and the point is the other end.

WINDOW This option requests the Charles to get a down button press for the lower side of the window and a button up stroke for the the upper side. This new window is defined from the previous window on the Charles terminal if one was present. The window command allows getting a closer view

of a portion of a cell.

VIRT This option sets the virtual plotting window. If two sets of coordinates are given, they define the window. If one coordinate is given it defines a window where the low point is negative of the point and the point is the other end. The units are as in the definition of VIRT[Wipfli 78].

DUMP This command writes a disk file of lines and boxes in a simple format for easy reload from the load function. The name of the file is <cell name>.srf.

```
-------- DUMP ----------------------------|
                |                         |
                ---- <cell name>-----
```

EDIT This commands starts the box editor in the LSI-11. The commands in the box editor are defined below. When a quit is done REST interprets the lines and boxes returned to it from the box editor.

```
-------- EDIT ----------------------------|
```

BOX EDITOR COMMANDS

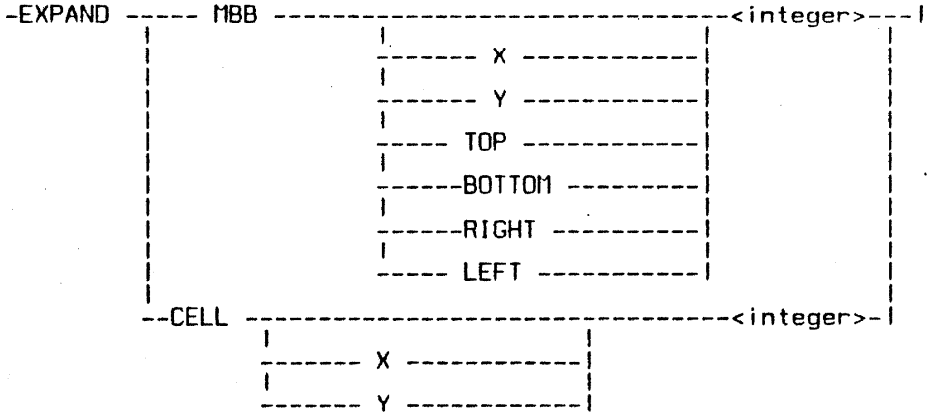
The box editor command menu is displayed on the right half of the graphic display. A palate color is selected by moving the cursor over the color desired. This color will be outlined when selected. To quit the box editor move the cursor over the quit command on the screen and press any button on the mouse. To switch from line to box move cursor over the line or box in the menu and press any button. Magnification is controlled by placing the cursor over the magnification value in the menu and pressing the right button for higher and the center button for lower. The following is a summary of the box editor commands. The key refers to the piano keyset with the numbers from left to right while the button refers to the mouse with the numbers from right to left. If there is not a number under the key no keys should be pressed.

EDIT: (currently selected box is outlined)

Key	Button	Action	Notes
	1	Define new box/line	Down for one corner, Up for other.
	2	Move selected box/line	Down attaches box to cursor, up lets go.
	3	Delete selected	Deletes the current outlined line or box.
1	1	To front	Moves selected box to front of the list.
1	2	No operation	
1	3	To rear	Moves selected box to rear of list, so that the one beneath him becomes selected.
2	1,2,3	Attach corner	Attaches closest corner of selected box to the cursor.
3	1	Center screen	Moves the cursor location to the center of the editing window.
3	2	Refresh	Clears the screen and rewrites the boxes on the screen.
3	3	Return	Restores the original editing window.

EXIT This command terminates the program. If the current cell has been modified, it must be deleted before a exit can be done.

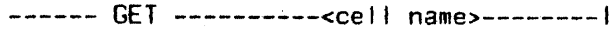
EXPAND This option is used to expand the cell or minimum bounding box for the current cell. The value expanded is in Lambda. For the bounding box the limit of the expansion is 50 while for the cell it is 10.



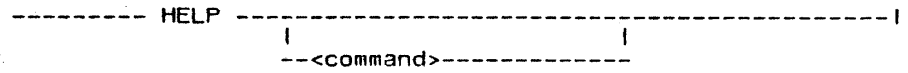
MBB This refers to the minimum bounding box. It may be expanded along the X-axis or the Y-axis, or just to the left or right or top or bottom or any combination. If no direction is given, all directions will be expanded.

CELL This option will separate every line and component from everything else by the supplied value. The cell may be expanded along the X-axis or Y-axis or both.

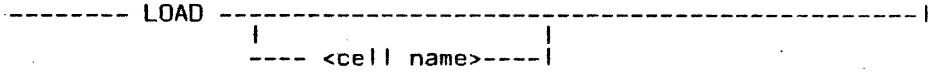
GET This command will read from a disk file with the cell name and create a current cell of that name.



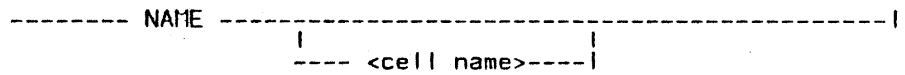
HELP This command is used to get aid for a command or information about REST.



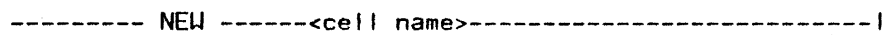
LOAD This command reads a disk file of lines and boxes in a simple format. The name of the file is <cell name>.srf.



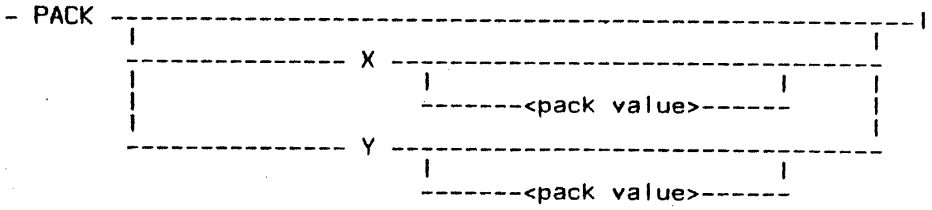
NAME This command can be used to type or set the name of the current cell. Suppling a cell name after the command alters the cell name.



NEW This command creates a new cell with the given name.



PACK This option compresses the sticks representation to NMOS design rules.

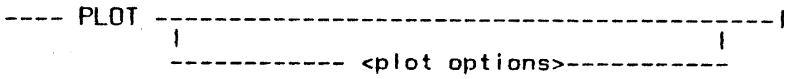


X This causes compaction along the x-axis of the leaf cell.

Y This causes compaction along the xy-axis of the leaf cell.

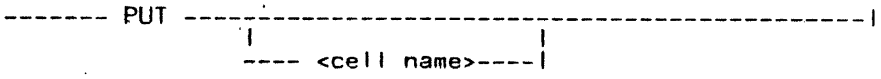
PACK VALUE
The pack value is either the keyword LOW or HIGH depending on the desired direction of compaction. The default is low.

PLOT This command plots the current cell on the HP plotter. The options are the plot options defined above in the display command.



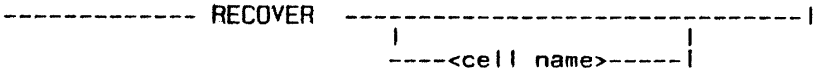
PLOT OPTIONS
The plot options are defined with the display command see page 79.

PUT The command writes the current cell on disk in the sticks standard format for later reload via the GET command or use by other programs.

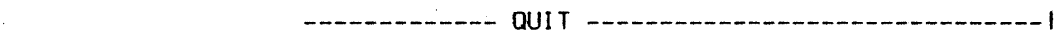


REBUILD The command causes the current cell to be run through the line drawing interpreter.

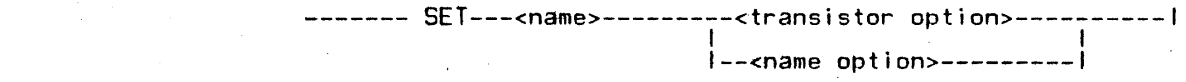
RECOVER This command allows recovery from a "<cell name>.rec" file on disk.



QUIT This command terminates the program. If the current cell has been modified, it must be deleted before a quit can be done.

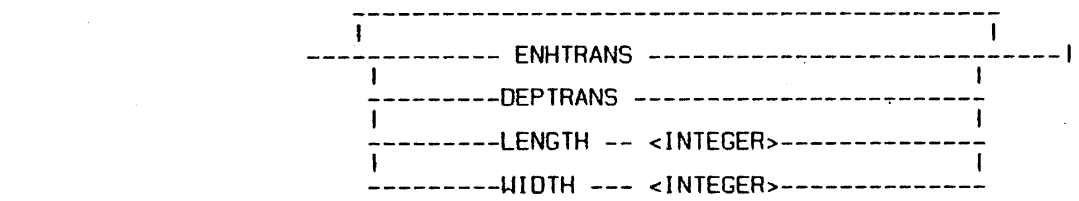


SET This option changes various parameters of the cell, such as the length and width of transistors.



NAME - The name is the name of the transistor to be changed. This name can be found by displaying the cell on the Charles or HP plotter with annotation.

TRANSISTOR OPTION



NAME OPTION

DEPTRANS This specifies that a depletion transistor is to be changed.

ENHTRANS This specifies that an enhancement transistor is to be changed.

PULLUP This specifies that a pullup device is to be changed.

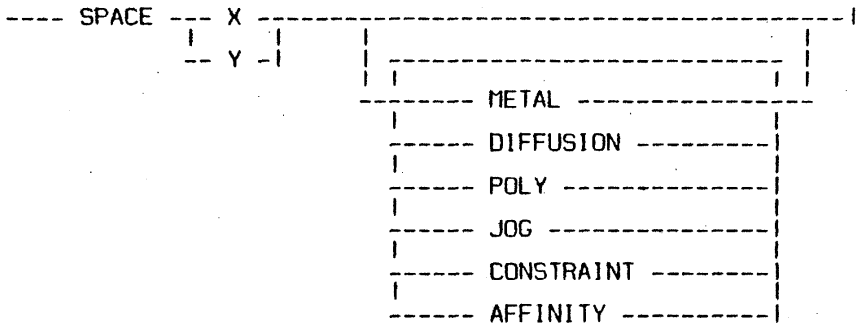
LENGTH This is the new value for the transistor length.

WIDTH This is the new value for the transistor width.

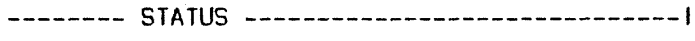
NEW NAME This is the name that is to replace the current name.

SPACE This option allows absolute control over the packing algorithm. The X or

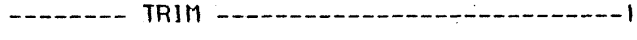
Y defines the axis to be packed. The LOW and HIGH define the side to be packed too. The METAL, POLY or DIFFUSION define affinity for those wires while AFFINITY defines for all. Jog allows jogging at line segment intersection points. The CONSTRAINT option says that constraints are to be applied. Only those option specified will be applied.



STATUS This command prints the current status of the cell.



TRIM This option is used to remove dangling segments. A call on trim will remove segments that have an open end point.



References

[Bentley 80a]

J.L. Bentley,D. Haken,R.W Hon.
Fast Geometry Algorithms for VLSI Tasks.
I E E (CH1491-0/80/0000-0088 IEE), 1980.

[Bentley 80b]

J.L. Bently,D. Haken,R.W. Hon.
Statistics on VLSI Designs.
Technical Report, Carnegie-Mellon University, April, 1980.

[Birtwistle 73]

Birtwistle G.M., Dahl O-J., Myhrhaug B. & Nygaard K.
SIMULA begin.
Auerbach Publishers Inc.,, 1973.

[Buchanan 80]

Irene Buchanan.
Modelling and Verification in Structured Integrated Circuit Design.
PhD thesis, Computer Science University of Edinburgh, 1980.

[Burke 80]

W.T. Burke.
Graphic Workstation Protocol, Version 4.0.
Silicon Structures Project SSP MEMO #3489, California Institute of Technology, 1980.

[Dunlop 79]

A.E. Dunlop.
Integrated Circuit Mask Compaction.
PhD thesis, Carnegie-Mellon University, 1979.

[Gray 80]

John P. Gray.
Private Communication with John Gray 1980.
1980.
Communications with John Gray during his stay at Caltech 1980.

[Heller 79]

W.R. Heller.
An Algorithm for Chip Planning.
Silicon Structures Project SSP MEMO #2806, California Institute of Technology, 1979.

[Hsueh 80]
Min-Yu Hsueh.
Symbolic Layout and Compaction of Integrated Circuits.
PhD thesis, University of California, Berkely, 1980.

[Johnannsen 79]
D. Johnannsen.
Bristle Blocks: A Silicon Compiler.
In Charles L. Seitz, editor, **Very Large Scale Integration.**
California Institute of Technology, Pasadena, California, 22-24 January 1979.

[Kahle 81]
C. Kahle,R.C. Mosteller.
NMOS Sticks Standard Components.
Silicon Structures Project SSP File #4300, California Institute of Technology, 1981.

[MEAD 80]
C.A. Mead and L.A. Conway.
Introduction to VLSI Systems.
Addison Wesley, 1980.

[Miller 56]
G.A. Miller.
The magical number seven, plus or minus two: some limits on our capacity for processing information.
Psychology Review :81-97, 1956.

[Minter 80]
Charles Minter.
Charles Terminal Care Package.
Silicon Structures Project SSP MEMO #3804, California Institute of Technology, 1980.

[Moore 79]
Gordon E. Moore.
Are We Really Ready for VLSI?.
In Charles L. Seitz, editor, **Very Large Scale Integration.**
California Institute of Technology, Pasadena, California, 22-24 January 1979.

[Noyce 77]
Robert N. Noyce.
Microelectronics.
Scientific American :Chapter 1, September, 1977.

[Rowson 80]
J.A. Rowson.
Understanding Hierarchical Design.
PhD thesis, California Institute of Technology, 1980.

[Sedgewick 77]
R. Sedgewick.
The Analysis of Quicksort Programs.
Acta Informatica 7 :327-355, 1977.

[Segal 80]
Richard Segal.
SPAM.
Silicon Structures Project SSP File #4029, California Institute of Technology, 1980.

[Sproull 79]
R. Sproul and R. Lyon revised S. Trimberger.
The CALTECH INTERMEDIATE FORM for LSI LAYOUT DESCRIPTION.
Silicon Structures Project SSP MEMO #2686, California Institute of Technology, 1979.

[Sutherland 77]
I.E. Sutherland and C.A. Mead.
Microelectronics and Computer Science.
Scientific American :210-228, September, 1977.

[Trimberger 80]
Stephen Trimberger.
The Proposed Sticks Standard.
Silicon Structures Project SSP MEMO #3487, California Institute of Technology, 1980.

[Williams 77]
John Williams.
Sticks - A New Approach to LSI Design.
Master's thesis, Dept. of Electrical Engineering and Computer Science M.I.T., June, 1977.

[Williams 80]
John Williams.
Private Communication with John Williams 1980.
1980.
Communications with John Williams during his stay at Caltech 1980.

[Wipfli 78]

J. Wipfli.
A SIMULA Graphic Package.
Silicon Structures Project SSP MEMO #1929, California Institute of Technology, 1978.

[Wulf 73]

W. Wulf and M. Shaw.
Global Variables Considered Harmful.
SIGPLAN Notices , February, 1973.