A STRUCTURED DESIGN METHODOLOGY & ASSOCIATED SOFTWARE TOOLS

by

Stephen Trimberger, James A. Rowson, Charles R. Lang
& John P. Gray

Technical Report 4336

July 1981

Computer Science Department
California Institute of Technology
Pasadena, CA 91125

Silicon Structures Project

sponsored by

Burroughs Corporation, Digital Equipment Corporation,

Fairchild Corporation, Hewlett-Packard Company,

IBM, Intel Corporation, Motorola, Xerox Corporation

and the National Science Foundation

A Structured Design Methodology and Associated Software Tools

by

Stephen Trimberger

James Rowson

Charles R. Lang

John P. Gray

California Institute of Technology

Computer Science Department

Abstract

The problems encountered designing very large scale integrated circuits (VLSI) are fundamentally different from the problems encountered in the design of small scale integrated circuits. The differences require a new methodology of design for the new large scale circuits, and the new design methodology requires a new set of tools. The computer aided design work at Caltech has progressed from a recognition of the inherent differences and has produced a new design methodology and a set of tools which attack the new problems in integrated circuit design.

This paper is intended to give the reader an introduction to the ideas and tools that underlie VLSI design research at Caltech. These ideas and tools have evolved not only from the faculty and students, who have been designing integrated circuits since 1970, but also from industrial representatives from Caltech's Silicon Structures Project. This project, in existence since 1977, staffed in part by sponsor's representatives on one year assignments at Caltech, has been a valuable source of insight into the problems being faced and about to be faced by VLSI designers.

The paper is divided into three parts: a description of the design philosophy, a summary of important tools built around this design philosophy, and a discussion of results and conclusions.

1.0 Design Philosophy

VLSI technologies have the capability to produce chips containing a hundred thousand transistors [Lattin 1981]. With current design techniques, it takes around 60 man years to design, and another 60 to debug, such a chip. In the theoretical limits, VLSI chips will contain nearly 10 million transistors. Without some method for reducing the complexity of design, a 10 million transistor chip would take somewhere near 6000 man years to design.

These numbers clearly indicate that there is a widening gap between what VLSI technologies can produce and what system designers can design. At present, the only chips that approach the available density are memory chips, and those only because of their extremely regular patterns. With less regular systems like microprocessors, designers are having difficulty merely completing complex designs.

The *structured design methodology* of Mead and Conway [Mead 1980] in use at Caltech is an approach to VLSI system design that attacks the problems of complex designs. Elements of this structured style are apparent in many successful designs.

The structured design methodology has two major parts: hierarchy and regularity. Hierarchical techniques have long been used to design complex systems [Simon 1962] [Koestler 1967]. Hierarchies are used to partition designs among the design team members. Also, common parts of a design can be factored out and specified only once. By introducing regularity into a system, the design problem is reduced in complexity as subunits are replicated many times and connections between units are simplified. Traditionally irregular control structures have their regular counterparts in ROM and PLA. Wiring strategy and regularity are addressed from the start, eliminating inefficient and costly routing.

The structured design methodology is similar in concept to structured programming [Dahl 1972]: the design proceeds in a top-down manner in which the problem is decomposed and refined. The designer is limited in the kinds of structures he may use to implement a function. The advantage of restricting the structures is that the design can be implemented more quickly and reliably. Proofs of correctness are easier to produce and more of the design can be automated.

In the words of E.J. Dijkstra [Dijkstra 1972]: "Testing can be used to show the presence of bugs, but never to show their absence." A major goal of computer aided design at Caltech is to produce correct designs without the need for checking. This goal of *correctness by construction* permeates the design methodology and tools at Caltech. Emphasis is placed on tools which generate designs to be correct in some sense, rather than on tools which check designs after they have been generated. These tools can be very specific to the design methodology in use at Caltech. The mating of the tools to the design style allows tools to be constructed which are much more powerful and leads to much greater productivity from the designers for whom the tools work.

1.1 The Separated Hierarchy

Hierarchical design at Caltech is predicated on a restricted hierarchy called the *separated hierarchy* [Rowson 1980]. The separated hierarchy, shown in Figure 1.1.1 has two different kinds of cells, *leaf* cells and *composition* cells. A leaf cell is defined only in terms of primitives. No instances of other cells are allowed. A composition cell contains only logical interconnections of instances of other cells, no primitives allowed. The tools needed for manipulating these two kinds of cells are

vastly different.

Figure 1.1.1 The Separated Hierarchy.

Composition cells in the separated hierarchy form a *representation* independent language for specifying a design. A representation is one particular view of a design. There are many possible representations including mask geometry, stick diagram, schematic diagram, English language description, behavioral description and so on. Leaf cells, containing only primitives, must be specified for each representation. The same composition cells can be used for all representations because they contain no primitives. In order to deal with a composition cell within a particular representation, a *composition rule* for each representation is specified. A composition rule is just an algorithm that will produce a definition of any legal composition in the corresponding representation. A legal composition could be as simple as "No two outputs may drive the same node."

A set of design tools built around the separated hierarchy provides mechanisms for designing leaf cells in many different representations, a composition rule for each representation, and a way to design composition cells. One problem inherent to a multiple representation system is that of consistency checking. If a designer specifies a particular cell twice, i.e., as geometry and as circuit schematic, some method for guaranteeing consistency between the two representations is needed. Composition cells, being representation independent, need not be checked for consistency. Composition rules must be consistent between representations, but this consistency need only be checked once per representation. Leaf cells clearly need to be checked. The hierarchical decomposition of the design leads to relatively simple leaf cells, making consistency checking (even by eye) relatively simple.

1.2 Algorithmic Design

Large chips closely resemble large programs in their variety and complexity. The need for powerful design systems to make these chips pushes us into the most powerful forms we have: algorithms represented by programming notations. The algorithm may be very simple, such as a series of rectangle placements. The use of a programming language allows placement of features to be done in a relative way, so that if one item or cell moves or grows, others follow. The procedures may be very

high level, such as routing of bus connections, deforming cells to interface without routing, designing a PLA, or computing the width of power wires according to the current required.

Cells can be parameterized so that they can adapt to the environment in which they are instantiated. For example, a transistor may change size to react to a change in load capacitance. The list of parameterizations is as long as the number of designers involved. Adaptable cells designed in this parameterized fashion will "live" longer and be more useful as building blocks in later designs which may have different requirements. They also allow decisions about detailed characteristics of the cell to be delayed until later in the design cycle.

As an example, consider a register cell in a dual-bus system. This register may be required to read data from or write data to either of the two buses in the system [Johannsen 1979]. Depending on intended use, the cell register may require refresh and preset logic. These registers may be used many times on each chip, and each usage, or instantiation, of the cell may require a different subset of the six available functions.

Since the cells are programs, we need not design all 64 possible register implementations and select the appropriate one as required. Instead, we can generate one register cell program that can remove unused circuitry and thereby generate any particular register instance as it is required. Figure 1.2.1 illustrates this technique. Figure 1.2.1a shows the six function register cell described above. It is possible for the same parameterized cell to be generated without the preset capability and without access to the lower bus, in which case the smaller register is produced, as shown in Figure 1.2.1b. The user of the register can specify its configuration with a list of the desired functions for the register, a behavioral description.

Figure 1.2.1. Algorithmically defined register cell. a) Register Cell with All Functions. b) Minimal Register Cell.

The recognition that the difficulties of complexity management overwhelm the problems of geometry generation leads to a new set of constraints on the design tools. Design tools must address the problems induced by the complexity of the design at all levels. Cells defined algorithmically can adapt to changes in their

neighbors, thereby restricting small changes in the design to small amounts of effort by the designers as they incorporate those changes. Programming languages provide the necessary versatility, although the precise characteristics of an ideal design language have not been determined.

1.3 Descriptions of the Design

It is possible to identify three domains of design description [Buchanan 1980] which must be addressed in a finished design. The three descriptions are: the behavioral, the description of the function of the design; the structural, the description of the form of the implementation; and the physical, the description of the physical implementation of the design. There are many possible representations of a circuit in each description, and judicious choice of representations is important in tool design.

1.3.1 Behavioral Description

Integrated circuit designers are in the business of building behaviors. A part which does not have the desired behavior is not acceptable no matter how clever the design. In this sense, designing integrated circuit hardware is similar to designing software and the problems associated with the two are often similar.

The global solutions to the problems in these two kinds of designs are similar. Designs are structured, hierarchical, and divided functionally into meaningful pieces. Tools are needed which help convert the high-level description of the behavior into a low-level implementation description.

Since we are concerned with the behavior of the circuit, work on design tools is influenced by rigorous disciplines and notations for describing circuits that can be guaranteed to be correct in some sense. For example, the self-timed discipline [Seitz 1980] can guarantee correct sequencing based on interconnection rules. Another good example is signal restoration in CMOS, ordinarily assured by checking, which can be guaranteed by rules of construction [Rem 1981].

1.3.2 *Structural Description*

A design is not merely a behavior. Designing integrated circuits is a mapping of a behavior onto a physical structure. There are fundamental differences between software design and integrated circuit design. One cannot blindly implement a function as a chip without addressing physical implementation issues. Integrated circuit design deals with inherent massive parallelism; software traditionally does not. Traditional software solutions to problems may not be proper in silicon. The designs, as well as the design methodology, must address the physical aspects of the medium.

Integrated circuit design has a severely limited communication space which shares the computation space on the two-dimensional surface of the silicon. Thus, communication costs are high in silicon, but very low in software. In order to effectively use the silicon area, a design system must take into account these properties of integrated circuits.

The structural description, is a description of the logical connection of blocks in the system. Hierarchical decomposition of the behavior of the design into blocks is done along both geometrical and functional lines. The functional and geometrical hierarchies must match, because the logical connection, the interface between functional units, is precisely along the geometrical interfaces.

The hierarchical decomposition is driven by a very high-level part-behavioral, part-physical *floorplan.* The floorplan is a general functional decomposition strategy which includes a wiring strategy as part of the decomposition. A good floorplan recognizes the two-dimensional nature of the silicon chip and addresses physical problems such as global wiring.

1.3.3 *Physical Description*

The physical characteristics of VLSI chips introduce new difficulties in complexity management not present in programs: the production of a geometrical structure under many constraints of topology and physics. Some systems have been built which implement function on an integrated circuit without regard to the inherent communication limitations of the silicon [Persky 1976] [Chen 1977]. These systems work well for small scale designs, but are overburdened with communication overhead with larger designs [Sutherland 1977]. For large designs, the physical constraints of

the silicon medium are so restrictive that they must be addressed early in the design. Systems which deny the physical nature of the silicon implementation cannot hope to effectively use the silicon medium in large designs.

Correctness by construction has been applied to the geometrical design task. Stick diagrams are used to generate geometrical layouts without the need for design rule checking. Designs can be produced much more quickly, and more iterations on a design can be quickly done to obtain an acceptably optimized layout.

1.4 Design Process Flow

This section describes the design process as practiced at Caltech. This design process embodies the structured methodology described above. This section also provides a framework for the design tools described in section two.

The design process has two distinct parts: design and implementation. Design proceeds top-down, with global decisions made first. The implementation then proceeds bottom-up, where constraints from low-level implementations are propagated to higher levels in the design hierarchy.

The design process is divided into five pieces: architectural design, where the design is partitioned into functional pieces and the general floorplan of the design is decided; cell estimation, where cell interfaces, -- size and interconnection -- are decided; cell detailing, where detailed cells are laid out; chip integration, where cells are assembled into chips; and preparation for fabrication, where the finished design is converted into a form suitable for fabrication equipment. Of course, the design process may iterate in any of the loops seen in Figure 1.4.1.

Figure 1.4.1 Design Process Flow.

The high-level top-down architectural design is still predominantly a human task. In order to achieve a good design, the designer must not only concern himself with functional decomposition, but with wiring strategy as well. The product of the architectural design is a floorplan, a tiling of the plane with functional units. The floorplan includes a rough specification for each of its elements, including constraints, interfaces and desired features. At this point, expected critical paths, both in area

and speed can be estimated. This estimate is important in order to properly direct later optimization work.

Cell estimation is an important precursor to cell detailing. In the cell estimation phase, approximate geometries are tried for the blocks defined in the floorplan. If these designs cannot be made, or if optimization of the design is too difficult, the floorplan may have to be altered. Stick diagrams are a useful notation in the cell estimation phase, they allow both structural and geometrical information to be expressed in a highly readable form. The cell estimation phase proceeds until all interfaces between cells are reconciled.

In the cell detailing phase, final designs are produced for each low-level cell using graphic or program oriented design aids. Detailed cells may be generated in the form of hard mask geometry, malleable Sticks or algorithmically-defined cells. Cell detailing is the start of the bottom-up implementation of the design.

Cell detailing has typically been given the lion's share of the CAD vendor's effort. In large systems, most cells are not in the critical path for speed of area use and therefore do not require optimization. Systems which automate the layout task are useful and desirable even if the result is not as good as hand layout. The fact that it is faster than hand layout is enough to make it worthwhile. A cell design must be consistent with the floorplan and it should have the ability to interface to high-level assembly tools.

Correctness by construction leads to the exploration of means by which classes of errors are avoided. Therefore, symbolic layout or stick diagrams are an important means for the generation of detailed cells as they guarantee designs free from geometrical design rule violations. In addition, since the stick diagram geometry is malleable by the very nature of the representation, sticks provides a good interface to the chip integration phase that follows.

The chip integration phase is the one in which cells are assembled, according to the floorplan, into a finished design. The assembly task is extremely difficult if the floorplan is badly done or if proper attention was not paid to the cell interfaces. Much work has been done at Caltech on chip assemblers and silicon compilers, which, working with a given floorplan and properly defined cells, will assemble low-level cells into complete integrated systems. These design systems have been very successful,

able to quickly produce many different chips in one class of floorplans.

Programming language based systems are preferred tools for the chip integration task because of their versatility. Powerful compositions can be easily defined in this algorithmic manner. Properly defined compositions allow changes to individual cells to be made without requiring a change in the way those cells are composed into a system. This leads to a very versatile design method. New ideas and optimizations as well as bug fixes can be made without requiring large changes to the composition of the system.

The final step in the design process is preparation for fabrication. This is typically a batch process, requiring a large processor for vast amounts of time. The hierarchical nature of Caltech designs enable this phase of the design to proceed faster than traditional methods. Plotters, mask generation programs and design rule checkers exist which take advantage of the hierarchy in the Caltech designs to speed up the processing enormously.

2.0 Design Aids

Caltech has developed aids ranging from pattern generation software to silicon compilers. Many of the tools have been tested by extensive use in designing large chips.

2.1 Overview

The design aids at Caltech are being developed almost entirely on a DECsystem-20 computer. The DEC-20 has a number of design stations consisting of an LSI-11 driving a text terminal, a four-color pen plotter, a color frame buffer for graphics output and a Xerox "mouse" pointing device for graphics input. The graphic workstation is shown schematicly in Figure 2.1.1.

Figure 2.1.1 Graphic Workstation.

The computer science department has access to an IBM 3032 computer belonging to the Caltech Computer Center. This machine is used for large scale batch-mode processing and use is made of the additional plotting facilities on the IBM machine. The department has also recently procured a Digital Equipment Corporation VAX 11/780. Some of the software has been transported to that machine, and much more will be developed for the VAX environment.

2.1.1 Data Structure Overview

Caltech has forsaken the standard approach to a unified set of tools, that of the all-encompassing restricted-access database. There are inherent problems with the database approach. Much of the effort in a database is in access control, file storage and optimization of memory usage. These are precisely the tasks performed by an operating system and executive. Rather than re-implement these functions, Caltech has decided to use the externally-supported, highly-debugged facilities provided with the DECsystem-20 computer.

A database allows tools to communicate, but only after conformance with the pre-defined database model. This model is usually extremely restrictive, since it was designed before the tools which use it. Extensions to the database are disruptive and costly. Interface to the database is typically a considerable fraction of the effort required for the development of a new tool. In a university, where innovation and experimentation are necessary, being locked into a stagnant database is surely a death knell.

Rather than limit future design systems with an outdated data description language, simple interchange standards allow the tools to communicate to achieve compatibility (see Figure 2.1.2). Each set of tools: leaf tools, composition tools, and foundry tools, is a collection of loosely coupled programs communicating through a standard file format. Thus, a tool may communicate with other design systems while still maintaining any internal representation necessary or desirable for its application. Such a data structure allows easy incorporation of new forms of data and new storage techniques. In a sense, we are following our own chip design advice by organizing our software to communicate through simple, well defined interfaces.

Figure 2.1.2 Tool Communication Overview.

The Caltech Intermediate Form (CIF) [Sproull 1980] is the standard interface to all the "foundry" (fabrication) tools. CIF is a hierarchical geometrical description, and contains commands to construct rectangles, wires, polygons and circles as well as facilities to define and call symbols. The Sticks Standard [Trimberger 1980] is the standard interchange form for systems which deal with symbolic layout and is being used as input to composition tools. The Sticks Standard is a symbolic layout description, and contains cells described with components, interconnections, and constraints on positions of the components in the sticks diagram. These standards provide the means for communication in a precisely-specified language, they clarify the essential concepts of the form of data they describe, and they stimulate tool development around the standard.

2.1.2 Tool Overview

Tools developed at Caltech can be grouped into three categories: those that help design leaf cells, those that help design composition cells, and those that help interface to fabrication. The leaf design tools aim at producing the primitive functional units used in a chip design. Leaf cells contain only primitive elements: polygons, rectangles, and wires. Composition cells contain only instances of other cells, either leaf cells or other composition cells. The interface to fabrication is needed because of the large number of designs that Caltech produces, both for research projects and class projects.

An overall view of the tools is illustrated in Figure 2.1.3. A set of "foundry tools" underlies a system of "compilers" which produce full silicon systems based on leaf cells designed using "cell design" tools. These sets of programs are not tightly bound into a single system, they communicate through standard data interchange formats.

Figure 2.1.3. Tool Overview.

The split between leaf cell tools and composition tools is deliberate. By separating the two, we not only match the structure of the separated hierarchy, but gain a great

deal of independence and flexibility. The independence allows tools to be built with little detailed knowledge of other tools. Flexibility is manifested in the ease with which new tools and new design approaches can be added to the overall system.

2.2 Leaf Cell Tools

A leaf cell is generally a simple functional unit that is implemented using only primitive elements. The set of leaf cells represent the semantic units the designer can use to describe a chip. Deciding which pieces of a design are to be made into leaf cells is a trade-off, matching a desire to have it be a self-contained functional unit with some limit on design difficulty. Too small, and the leaf cell won't be complete enough to specify an understandable function. Too big, and the designer won't be able to understand the relationships among all the parts, making it difficult to understand and modify.

Figure 2.2.1. Leaf Cell Examples. a) Super Buffer Cell. b) Static Register Cell.

In chip design, the design of a leaf cell is basically graphical in nature. The cell designer must specify a mask geometry which implements the desired function.

There are three methods used at Caltech to design leaf cells: interactive graphics, graphical languages, and a combination of both. All three have their advantages. The interactive graphic systems provide a very good human interface, allowing rapid on-line design. Graphical languages provide a less interactive environment because designs have to be "executed" before a plot is produced. But they give the designer more power, in the sense of variables, expressions, conditions and loops, than the interactive systems. Hybrid systems have some of the advantages of both of these systems.

The three leaf design methods are embodied in the following design tools: REST, LAP/PAUL, and SAM.

2.2.1 REST (graphics)

Richard's Editor for STicks (REST) [Mosteller 1981] is a leaf cell design system based on symbolic layout techniques of [Williams 1977] and closely related to the

work of [Hsueh 1979]. REST runs on the high-resolution color display with editing functions handled by code in the display processor. The symbolic layout manipulation is done in the DECsystem-20. Since REST is a leaf design tool, it does not support a design hierarchy. REST interfaces to the Sticks Standard which can be read by silicon compilers for chip assembly.

The REST system deals with a form of symbolic layout known as *stick diagrams*. A stick diagram is a notation midway between transistor diagrams and full mask layouts. Stick diagrams specify more geometrical information than a transistor diagram in that the relative positions of transistors and wires are meaningful, but less than a full layout in that the absolute positions of transistors and wires are not meaningful. This intermediate position has many advantages. The designer can specify his layout in a very sketchy manner, with no regard to exact positioning, yet still have control over the relative topology of the layout. Component recognition and circuit compaction free the designer from worries related to details about mask making. Contrast the sticks drawing, of a static register cell in Figure 2.2.2b with its derived layout in Figure 2.2.3. The stick diagram includes device sizes and connectivity information, which are essential for performance estimation and simulation. This abstract specification allows the designer to concentrate on the system design. Stick diagrams were designed to be deformable, making them a preferred input form for silicon compilers and chip assemblers (see section 2.3).

Figure 2.2.2. Stick Diagram of Static Register Cell. a) Uncompacted. b) Compacted.

Figure 2.2.3. Layout of Static Register Cell.

The compaction algorithm in REST compacts along the coordinate axes one axis at a time. The algorithm is simple and very fast, providing a unified, highly-interactive man-machine interface. The user can control the direction of compaction, which axis is compacted first, and add constraints to the graph. REST does not make topological changes. The results of compaction are illustrated in the contrast between an uncompacted stick diagram input by the designer, and the stick diagram resulting from compaction in Figure 2.2.2.

REST has been used successfully in conjunction with other design tools at Caltech to produce working designs. Its usefulness will expand as additional composition tools become available.

2.2.2 LAP/PAUL (procedures)

There are many advantages to using an expressive language to design integrated circuits (see section 1.2). Almost any programming language can be used to design mask geometry simply by adding some procedures to output primitives of the Caltech Intermediate Form (CIF). A simple example of such a procedure is "Rectangle" which simply outputs the CIF code for a rectangle. The coordinates are the parameters of the procedure. Most programming languages allow those parameters to be expressions, enhancing the adaptability of the procedure. A more complex procedure might be "PLA" to generate a programmed logic array. The parameters of this procedure are the coordinates of the origin, number of minterms, inputs, and outputs, and the names of the files holding programs for the AND and OR planes. There is no limit on the complexity of the design generated by the embedded procedures: the PLA procedure may optimize the PLA without any changes needed by the PLA user. These procedures can be called by the designer in the midst of a program that "computes" the chip. Such procedures create a kind of "embedded language" that can be used as a design language.

LAP [Lang 1979] and PAUL are two such design languages, both embedded in the programming language SIMULA. LAP produces CIF files, PAUL produces Stick Standard files. The full power of the SIMULA is available to the user in either of these languages. LAP has been in use since 1977 at many installations in research, education, and industry and has been a major chip design system for both leaf and composition cells. Over the years, it has been enhanced with a PLA generator, several simple routers, mask lettering generator and Sticks Standard to CIF translator. PAUL is a recent development designed to allow the generation of stick diagrams without the use of expensive graphics equipment, and to give the user the power of the SIMULA programming language with which to generate the stick drawing.

2.2.3 SAM (graphics and procedures)

Layout languages provide users with the capability to algorithmically define cells. But the specification language is so non-intuitive that it is impossible to debug a design in that language. One must plot the design. Interactive graphics systems, on the other hand, allow the user to debug in the form in which he sees the design, but severely restrict the language he may use to express the graphics. For example, he cannot

express loops or conditionals. What is really needed is a single interactive system that combines layout language and graphic modifications to the data. SAM [Trimberger 1980a] is just such a system.

SAM provides the user with a two-part viewing window on the display as seen in Figure 2.2.4. The left side shows the program view of the design under edit, the right side shows the graphics view of that cell. The user may move the viewing location in either window and may make edits to the data in either window. When the design is changed in either window, the change is reflected immediately in both windows.

Figure 2.2.4. SAM Display.

The data displayed in the windows are "pictures" of the data structure. The data structure is the base form, the program view and the graphic view are merely different ways of looking at the base form of the data. When either the graphic bitmap form or the program character-string form is needed for display it is generated from the data structure. When the user makes what appears to be a modification of the data in either window, the commands are translated into calls on procedures in the data structure to carry out the action. The data structure makes the modification and causes both displays to be updated. The two views are kept consistent because they are both refreshed from the same data in memory.

SAM allows a designer to create algorithmically-defined cells easily, cells defined as a program rather than a list of graphics. These cells are parameterized as well. When an instance of the cell is made, parameters are passed to the cell which specify desired attributes of that instance -- for example, cell size or the locations of the connectors. The parameterization can be used with the program features of looping and conditional execution to process properly such input parameters as driving power, number of bits of length of a register, or PLA coding.

SAM is experimental system. It was written to test the concepts and feasibility of combining the program and graphic methods of design. Since SAM is an experimental system, it is not in use as a design tool. Several test designs were made with SAM with encouraging results. The knowledge gained from those tests is directing further work in this area and has affected the development of several other new tools.

2.3 Composition Cell Tools

The problem of VLSI design is largely a problem of composition. It is a problem of organizing separately designed and debugged modules so that they work together correctly. The process of composition is at least as error prone as the process of leaf cell design, and the means for discovering errors in composition are poor at best and extremely costly. Compostion errors are more likely to persist until late in the design process.

In order to make correct composition possible, the designer must be freed from responsibility for all the myriads of details that must be specified. This emancipation is often accomplished by inposing structure through the design tools. By imposing some structure, the design tool is able to perform the mechanical aspects of the design, leaving the more intuitive portions of the design to the user.

Once the composition tool is debugged, the designer is also freed from worry about whole classes of mistakes. By accepting a small set of restrictions brought on by the imposed structure, the designer is relieved of most of the particulars of the design, and the tool can be trusted to do them correctly. Once the user has gained sufficient confidence in the tool much of the post-design checking like geometrical design rule checking can be bypassed. "Correctness by checking" has been replaced by "correctness by construction."

Composition tools come in many forms, and under many names, but they all trade designer restrictons for degrees of computer automation. An example of a composition tool s the common PLA generator. This tool imposes the structure that the design will be in the form of a PLA. The input is generally in the form of some form of logic equations or AND and OR plane code and all of the layout details are handled by the progran. It is of course unreasonable to expect every design to be implemented efficiently as a single giant PLA. However, the PLA generator can be used to autonatically implement appropriate pieces of a large design. Drawing on software analogies, more general composition tools are often termed "chip assemblers"or "silicon compilers".

Caltech has three powerful composition tools currently under development: Bristle Blocks, SLIP/Earl, and SPAM. These are described in the sections below.

2.3.1 Bristle Blocks

Bristle Blocks [Johannsen 1979] is a silicon compiler that specializes in the construction of *datapath* chips. A datapath chip consists of data processing elements connected by and communicating across data busses. Typical data processing elements include register files, arithmetic/logic units, and shifters. The chip is controlled by a microcode word which is decoded on-chip to drive each of the individual control lines of the processing elements. The functional block diagram for a Bistle Blocks datapath chip is shown in Figure 2.3.1. Notice that the physical floorplan of the chip is identical to the functional floorplan. Like many other automatic layout systems [Persky 1976] [Chen 1977], Bristle Blocks imposes a generic floorplan in return for ease in automating the layout. While not all designs can fit into the high-level "datapath" floorplan, those that can are implemented as efficiently as hand layout.

Figure 2.3.1 Chip Floorplan. a) Block Diagram. b) Physical Floorplan.

The primary characteristic of Bristle Blocks cells is that they are programs rather than data (see section 1.2). Rather than designing cells as wires and boxes, the cells are designed as programs which, when executed, generate the required wires and boxes. This allows the cells to perform computations and participate in the design of chip.

Because of the need to minimize area, the cells composing the datapath processing elements will not all be the same size, which means that the cells cannot be interconnected by simple abutment. Analysis has shown that global chip area can be reduced and global chip performance increased by stretching the cells to connect by abutment instead of placing the cells and routing between them. See Figure 2.3.2. Bristle Blocks makes use of stretching to assure that all cells are of uniform height. Bristle Blocks leaves the mechanics of stretching to the individual cell: each cell is designed with the means for stretching built into it. Figure 2.3.3 shows a stretched register cell.

Figure 2.3.2 Routing versus stretching. a) Connections made by routing. b) Connections made by stretching.

Figure 2.3.3 Stretched register cell.

The input to Bristle Blocks consists of the cell definitions as programs and a high level description of the chip. The high level description is in terms of calls to the cell programs, passing the appropriate parameters for the chip to be designed. These parameters typically consist of behavioral information, such as the conditions required for the register cell to load information from a bus. The description for a fairly large chip is approximately three pages long. To build the chip, Bristle Blocks executes the cell programs as specified in the description to create the datapath portion of the chip. Bristle Blocks stretches cells to be of uniform height and abuts them. The datapath timing and control information is used to add control line buffers, parallel load shift registers and a semi-optimized instruction decoder to drive the datapath. Finally, Bristle Blocks adds bonding pads and wiring to complete the chip and provides documentation about the locations of pads and signals.

Bristle Blocks has been operational since 1978 and has undergone several improvements and enhancements. The user may now direct the compiler to insert LSSD registers for testability [Eichelberger 1977]. Recent enhancements allow a much more general floorplan to be compiled, enabling the system to compile entire multiple processor computer systems. Bristle Blocks can compile systems with circuit densities comparable to hand design which are well beyond the capabilities of current integrated circuit fabrication technologies to implement. The process of compiling a sixteen bit datapath chip from the high-level description took fifteen minutes of elapsed time on the DECsystem-20/60 computer. The resulting design was 3600 by 4250 lambda (lambda equal to 2.5 microns) and contained approximately 13,500 transistors.

2.3.2 SLAP/Earl

SLAP/Earl are two implementations of a system tied closely to the separated hierarchy.

Composition cells specify interconnections between instances of other cells. Any of a wide variety of notations can be used to describe composition cells. The notation familiar to most designers is a *netlist* form. A netlist is simply a list of instances along with a list of logical connections between instances.

Logical connections are actually made between *connectors* on instances of cells. Each cell has a list of connectors that define the *interface* to the cell, the means by which the outside world may interact with the cell's instances. The connectors are the only available attachment points to an instance of the cell. By explicitly declaring the interface, interactions with the cell can be controlled, localizing the behavior of the cell and simplifying the task of verifying correct usage.

One particularly important representation for VLSI design is the geometrical representation. The composition rule presented here is only one possible algorithm and certainly not the best, merely one of the first tries. This rule is based on the structured layout ideas developed in Mead and Conway [Mead 1980]. In particular, the layout is designed to be a regular array of instances of cells designed to *abut* perfectly. Perfect abutment implies that all the connectors along the shared edge (see Figure 2.3.4) are designed to be at the same position so that abutment will perform the logical interconnection desired, i.e., all the connectors on the right edge of instance A will connect to corresponding connectors on instance B.

Figure 2.3.4 Connection by abutment.

This method for structuring the layout design fits very nicely into the separated hierarchy. Composition cells, containing only logical interconnections of instances of other cells, are logically very similar to the abutment style shown in Figure 2.3.4. An automatic mapping can be made between the netlist description of a composition cell and an abutting cell layout description.

To implement geometry composition, interconnection is defined to be *superposition* and is accomplished through *stretchable cells*. Every logical interconnection between connectors on instances results in those two connectors being placed at the same physical location. In order to guarantee that this placement can be accomplished, all cells must be able to increase the distance between any two adjacent connectors. This use of stretchable cells is a slight generalization of the identical notion in Bristle Blocks.

The composition rule must perform some algorithm that satisfies all the constraints introduced by the interconnections. By restricting cells to be rectangular with all connectors lying on the boundary, the basically two-dimensional layout problem is split into two one-dimensional problems. Each dimension is translated into a directed

graph and solved using graph techniques.

The nodes of the geometry solution graph represent *equivalent* sets of connectors. A logically connected group of connectors is represented by one node in the graph. The weighted, directed arcs represent minimum separation constraints between connectors resulting from constraints inherent in the leaf cells. Figure 2.3.5 illustrates the resulting graph. Notice that the graph must be acyclic, but there may be multiple arcs between the same two nodes, possibly with different weights. The graph is solved with an algorithm that is very similar to the algorithms used in stick diagram compaction [Hsueh 1979].

Figure 2.3.5 Solution Graph for Horizontal Direction.

The SLAP system was embedded in SIMULA. Currently, work is in progress to produce an interpretive implementation, called Earl, which will provide a concise composition cell description language. The Earl language is based on list manipulation primitives that allow simple construction and interconnection of lists of instances and/or connectors. A major design project to build a tree connected parallel processor [Browning 1980] is currently underway using the Earl system as a primary design tool.

2.3.3 SPAM

Structure, Placement And Modelling (SPAM) [Segal 1980] is a system that can be used to describe a hierarchical design which can then be simulated at any level of detail. The system provides a concise method for describing composition cells. SPAM deals with a structural description from which a physical positioning might be generated using Earl. In addition, the behavior of a composition cell can be described. An entire design can be simulated to any desired level of detail by choosing which cells are the "leaves" of the simulation, i.e., at which points in the hierarchy the behavioral descriptions are used instead of the behaviors of its parts.

Since individual physical devices are not important in SPAM, the task of describing the structure of a design is primarily the task of *interface specification*, the specification of the organization of communication between modules. Cell names, pin declarations, and pin typing are all the parameters necessary to completely describe the structure of a leaf cell. Typed pins allows a more complete structural

verification. SPAM provides six types: *Input, Output, IO, Power, Ground, and Clock.* When the hierarchy is instantiated, every connection of two pins is checked for legality.

Composition cells must specify an interconnection of instances of other cells. Cell instances can be declared individually or as arrays. The interconnection of instances is done only through abutment, which can be described either graphically or textually. No individual pins are mentioned, whole edges of rectangular instances are interconnected. Thus, "the left of X abuts the right of Y" implies that all the pins on the left of X connect to all the pins on the right of Y. Since it is occasionly desired that some pins be left unconnected, a clause can be added to abutment declarations to omit some connections. From the interconnection description a floorplan can be derived, as in Figure 2.3.6.

Figure 2.3.6 SPAM Floorplan Output.

Until it is possible to "compile" function into designs it will be necessary to use simulation to assist in the verification of behavioral integrity. The SPAM environment favors designs produced and verified in a top-down manner. That is, an initial chip description will be produced and simulated. When the results are deemed correct, the design is broken up into components and each module is given a behavior. The resulting network is simulated and the results are compared to the original test. If the same results are produced, then this refinement has been successful. The process continues until the leaf cells have been described.

Simulation in SPAM is highly interactive. SPAM contains a four-value, combination event-driven [Ko 1979] and clock-driven [Bell 1970] functional simulator. The two simulation modes may be used in the same simulation run in different cells. The user may examine any node at any time. The user interface to the simulator is similar to the interface to a symbolic debugger for a programming language.

Once a circuit description has been compiled, the user may request a *documentation workbook* consisting of a hierarchical map of the entire circuit, an interface specification diagram for each cell definition, and a floorplan diagram for each composition cell in the description.

2.4 Silicon Foundry Tools

The most important aspect of a silicon foundry is its interface to the designers it serves. Designers communicate their designs to the foundry using the Caltech Intermediate Form (CIF) and receive from the foundry bonded dice and appropriate documentation. Designs sent to a foundry are specified as the actual geometry of the desired devices on the chip. Acceptance or rejection of the chips is determined by the performance of standardized test patterns. The designers are insulated from any detailed knowledge of the foundry's process, but require assurance that their designs have been fabricated correctly.

In the past, Caltech has acted as its own foundry, accepting chip designs described in CIF and performing the necessary steps to produce pattern generator tapes. The mask generation and chip fabrication services were then purchased from commercial houses. The advent of the Multi-Project Chip (MPC) program, administered first by Xerox Palo Alto Research Center [Conway 1980] and currently by USC's Information Sciences Institute, has eliminated the need for Caltech to perform these functions in house. In the future, it is hoped that commercial organizations will develop to provide this service.

A number of tools have been developed at Caltech to aid in the transformation of individual chip designs to fabricated chips. Not all of these tools are restricted to foundry use, many are useful in the design process as well. In particular, interactive checkplotting programs are of great use to designers. At the top level, there are check plotting facilities and a design rule checker. These act as a filter to prevent the unwitting fabrication of designs which cannot work due to the presence of design rule violations or other errors which preclude fabrication. Once individual designs are deemed acceptable, they must be packed into one or more dice. Test circuits and patterns, alignment marks and scribe lines must be added to suit the process line on which the designs are to be fabricated. Finally, the image of the die must be converted to a format from which masks can be generated.

2.4.1 Check Plotting Facilities

At several points, the image of the designs, or of the entire die, must be viewed to detect any gross errors. If one had complete confidence in the accuracy of the tools

that manipulate the artwork, check plotting might not be necessary. However, the cost of mask making and fabrication is high enough to make such checks worthwhile.

Two plotting tools have been developed at Caltech for the purpose of plotting CIF files. The first of these, an interactive program named CIF2OP, was originally developed for use by designers in verifying their layout designs. It has become useful in assembling multi-project chips as well. It accepts any legal CIF file as input and, in response to commands by the user, plots various portions of the file's geometry on a variety of output devices. The primary virtue of this program is its fast response. It is essential that the designer be able to make small, localized modifications to a design and view the result quickly. Interactive views of individual cells and areas of the design must be available to the designer within minutes. This fast feedback is an important characteristic that allows small changes in the design to be viewed immediately regardless of the overall size or complexity of the layout.

A second plotting program, ALEX, is a FORTRAN program intended for batch use. Like CIF2OP, it can utilize a variety of graphic output devices. ALEX is used primarily for making large plots on a Calcomp four-color plotter. Users can specify a window within their designs and can control which symbols and which layers are to be plotted.

A third check plotting program provides verification that the pattern generator tapes have been made correctly. This program is called CHKPLT and it takes as input both Electromask and Mann 3000 pattern generator (PG) formats. It plots the PG instructions as they would be encountered by the PG machine, verifying that no obvious errors have occurred in the generation of the PG instructions and also showing that the instructions have been sorted in the order appropriate for the particular PG machine. This is the last check made of the data before it is sent to the mask making house.

2.4.2 Design Rule Checking

While many of the design tools developed at Caltech endeavor to produce correct artwork by construction, until these tools gain the confidence of designers and are shown to be error free, design rule checking will remain a necessary part of the design process. A significant improvement in design verification has been made recently at Caltech by the development of a hierarchical design rule checker (DRC) [Whitney 1981]. The ideas and motivations for this tool are described by McGrath

and Whitney [McGrath 1980].

Rather than taking the traditional approach of doing geometrical operations on a fully instantiated description of a design, this DRC makes use of the symbol hierarchy found in the CIF description to eliminate as many redundant comparisons as possible. The correctness of geometry inside a CIF symbol is checked only once, regardless of how many instances of that symbol are made. The environment in which each instance of a symbol is found is remembered so that a particular set of symbol interactions is only checked once. This technique gains a great speed advantage on regular chip designs without restricting the complexity of the geometrical shapes taken as input. Design rule checks can be done on arbitrarily complex rules.

This approach contrasts other attempts to design efficient DRC programs which generate a rasterized picture of each mask using a one lambda resolution and perform design rule checking using this representation [Baker 1980]. Such an approach gains performance at the cost of restrictions on the form of the input and on the complexity of design rules which can be checked.

As much of the design intent as is available in the CIF description is used to reduce the redundant checking of duplicate geometric situations. Of course, some of the design intent is lost in describing the design in CIF. Some extensions have been made which allow more of the structural data associated with a design than is normally found in CIF to be made available to the DRC. Also, some conventions have been imposed on the input CIF file. Primitive elements have been defined for such objects as transistors and contacts. Since many design rules control the legal construction of these primitive elements and their interactions with other objects, considerable simplification and performance are gained if these objects are identifiable in the design description. Some of the most obscure and difficult design rules are related to the construction of transistors. Since these are irreducible structural objects, they may be defined as primitive elements without reducing the flexibility available to the designer. Primitive elements can be checked for correctness prior to their use by the designer and only the interactions of primitive elements with other objects need be checked, avoiding any need to check their internal parts.

These techniques minimize the number of geometrical spacing and width checks that must be performed between primitive geometrical shapes. The use of the design

hierarchy can eliminate the high complexity inherent in the design rule checking of fully instantiated designs. The number of false or spurious errors is also reduced. Since few redundant checks are made, few redundant errors are generated. The dramatic reduction in the number of shapes compared is shown in Figures 2.4.1 and 2.4.2. The first shows a small design, the second shows those shapes that were actually compared with other shapes during the check. For clarity, only the polysilicon and metal layers are shown in the figures. When design rule checking, of course, all layers are processed.

Figure 2.4.1 Design Rule Checker Example.

Figure 2.4.2 Shapes Actually Checked by DRC.

The extraction of the design topology, a side effect of DRC, can provide an important verification tool for designers in determining whether or not they have implemented the circuits they intended in the artwork. When two shapes are later checked against design rules, their membership in the same or in different nets can play an important role in detecting whether or not an error truly exists, reducing false errors. The ability to compare the designer's topological data with that found by the DRC is not part of the DRC function but it will hopefully be available at Caltech in the near future.

2.4.3 Assembling Multi-Project Chips

A set of three programs have been developed at Caltech to automate the process of assembling a collection of independent project designs into a single die image. The projects are input as individual CIF files with no restrictions on the use of legal CIF commands. The result is a single CIF file containing all the projects placed inside a rectangular die, including scribe line geometry, mask labelling and fiducial marks.

The first step in the process verifies that each project file contains legal CIF commands and extracts the bounding box of the geometry. The second step is the packing of the projects into a single rectangle representing the size of the die. When a trial packing has been accepted, the location and possible rotation of each project on the die are saved for use in the third and final step.

Lastly, the packing information and the original CIF files are used to produce a single, large CIF file representing the entire die of the multi-project chip. Fiducial marks are added and each layer is labeled with the name of the multi-project chip and the mask identifier. Process line and mask house dependent information is read from a file describing what type of fiducial marks, scribe lines, layer labeling are necessary for the organizations who are to make the masks and process the wafers. Test patterns for checking the lithography, circuits for checking electrical characteristics and alignment marks are added to the die merely by including the appropriate CIF files as additional projects.

At the completion of this last step, a CIF file is available for conversion to pattern generator format. To document the placement of projects on the die, a plot is made of the die with outlines of each project and their names included. The entire process of taking individual CIF files and producing the image of a multi-project chip in this fashion requires less than an hour of elapsed time.

2.4.4 Translating CIF to Pattern Generator Formats

A FORTRAN program named PAT was written at Caltech to convert CIF 2.0 data to the formats required by Mann and Electromask pattern generator machines. This program removes the CIF symbol hierarchy by recursively replacing instances of symbols with their corresponding geometrical primitives as it moves through the input file. PAT fractures all the primitive CIF shapes, including arbitrary non-self-intersecting polygons, into the set of rectangles available on the PG machines.

To isolate the user from the details of chip fabrication, CIF files specify the actual geometry of devices on the chip as they are returned to the designer. To translate from the fabricated geometry to the actual mask artwork requires knowledge available only to the silicon foundry and which the foundry would prefer to keep confidential. These characteristics of the process affect the relationship between the artwork and the fabricated geometry. PAT performs the translation of fabricated artwork to drawn geometry by bloating and shrinking the features on the individual mask layers before the format conversion is done.

PAT optimizes the ordering of the data on the PG tapes to minimize the time required by the PG machine in making the reticles. The optimization algorithm is easily modified to accommodate variations in different machines. The PG output data produced by

PAT can be visually checked by plotting it with CHKPLT. This provides a picture of the final data as it is sent to the mask house.

3.0 Conclusions

The structured design methodology and algorithmic approach to design tools appear to be a good fit and are working well. The emphasis on structures which allow implementation of the desired function in a manner suitable to integrated circuit implementation has been and continues to be a great success. A great advantage has been gained by building tools closely related to the design methodology. These tools are built with assumptions about the general form of their input. These assumptions lead to simpler implementations of tools and more powerful design systems. Powerful design systems have allowed us to quickly produce complex chips.

3.1 Successes

The major successes have been in four areas: methodology, chip assemblers, standards, and silicon foundry software.

There is no doubt that adopting the structured design methodology and rigorously structuring a design produces great benefits. These benefits include shorter design times, better design management, easier checking and increased probability of correctness. Over a period of time, work at Caltech has produced some crude measurements of improvements. For example, it is possible to achieve an order of magnitude speed-up in delivery of artwork over conventional designs.

The invention of the chip assembler has contributed significantly to reduction in design time. Paradoxically, there is even evidence to show that compiled layouts consume less area than hand-packed designs. This could be a result of the compiler inexorably applying area optimizations where a designer may be less careful. Additional savings come from global optimizations made possible by the quicker

turnaround of the assemblers. These area reductions may be as large as twenty percent.

The somewhat radical view at Caltech which promotes helpful, yet unpretentious standards in lieu of an all-encompassing database has allowed nearly unconstrained work to progress on a number of tools independently, yet has allowed the tools to be interfaced in a well-defined manner. This has contributed significantly to the productivity of the group as a whole.

Although superficially less interesting, the development of the basic CIF software (foundry software) has been a notable success. It is in continued use at Caltech and other institutions.

3.2 Difficulties

No major difficulties have been found with the structured design methodology, although refinements continue to be made. The major criticism of correctness by construction is that it interferes with optimization of the circuit. It is true that work at Caltech has addressed the issue of correctness at the expense of area and time optimization. It seems clear that some area or performance penalty should be incurred in structuring a design. After all, the largest determinant in systems performance is design at the highest level of abstraction. However, it is difficult to make convincing arguments without quantifiable evidence. At this point, we are generating measures and collecting data.

The Caltech methodology moves the responsibility for correctness to the construction programs. The feedback paths traditionally provided by verification programs have been replaced by restrictions on the design structures. This trading of verication feedback for the simplicity gained by automating details of the design task, is precisely the tradeoff made in the software community for compiled languages over assembly languages. Elimination of feedback paths may diminish confidence in the resulting designs. It is not yet known if verification tools can be eliminated totally or if they can be replaced by simpler verification tools.

The major difficulties in tools have been in those areas not compatible with the directed thrust of the department, specifically, checking tools. The difficulties with

checking tools appear to stem from the current state of tools at Caltech, which only recently embraced the correctness by construction philosophy for all phases of design. Previously, tools existed only for selected parts of the design process, and much checking was necessary for the remaining design tasks.

3.3 Future Work

To date, emphasis has been on physical and structural design. In the future, we will explore temporal and functional approaches to design. Inevitably, this will center around models and notations appropriate to each design domain.

Compilation without regard to the physical aspects of the implementation medium appears to lead to poor use of the medium. This may be due to the low-level target representation for the circuit, transistors. Preliminary work has shown that it is much easier to convert a behavioral description of a circuit into a physical device by compiling to a known set of low-level structures, somewhat analogous to re-defining the low-level primitives for a programming language compiler. Compiling to transistors appears to be extremely difficult, compiling to datapaths, memory arrays, PLAs and decoders may be more easily implemented. Future work will explore reasonable goal structures for compilers along these lines.

Many of the techniques in use at Caltech produce a great savings in design time. However, there are, as yet, no good measures of that savings. Future work will be done to test the cost/performance of the design methodology in general, as well as specific instances of it. Subject to test in the near future are Sticks diagram systems and self-timed logic.

3.4 Open Questions

There are many areas of computer aided design which have not been addressed at Caltech. Some of these may come under investigation in the near future, some may not.

The Caltech standards work has proved to be very successful. However, it is not immediately apparent what further standards are needed. It is not reasonable to

define standards in the hope that tools will arise that need those standards. More experience will reveal the need for further standards as well as needs for changes in those currently in use.

Many books and papers have been written on the organization of design teams for software. These are usually part of a software design methodology. Caltech has done no work on the organization of design teams using the structured design methodology for integrated circuitry, although it is becoming clear that that organization must be different than that currently in use in the integrated circuit industry. The design team organization will be influenced by the role of the computerized design aids to be used, which, of course, are ultimately dependent on the design methodology.

4.0 Acknowledgements

We are extremely grateful to the sponsors of the Silicon Structures Project: Burroughs Corporation, Digital Equipment Corporation, Fairchild Corporation, Hewlett-Packard Company, Honeywell Incorporated, International Business Machines Corporation, Intel Corporation, Motorola Corporation, Sperry-Univac Corporation, Xerox Corporation, and the National Science Foundation for their support of our computer aided design effort. We would also like to express additional thanks to the individual visitors from those sponsors who have contributed immensely to the success of the project.

The work on design aids has benefitted from complementary research in the "Submicron Systems Architectures Project" under Defense Advanced Research Projects Agency sponsorship.

We are also grateful to the good people at Caltech who helped put this paper together: Dave Johannsen, Chris Kingsley, George Lewicki, Richard Mosteller, Richard Segal, Chuck Seitz and Telle Whitney.

5.0 References

[Baker 1980] C.M. Baker and C. Terman, "Tools for Verifying Integrated Circuit Designs", *Lambda Magazine*, vol. 1, no. 3, 1980.

[Bell 1970] G. Bell and A. Newell, "The PMS and ISP Descriptive System for Computer Systems", Proceedings of the AFIPS SJCC, 1970.

[Browning 1980] S. A. Browining, "The Tree Machine: A Highly Concurrent Computing Environment", PhD Thesis, California Institute of Technology, 1980.

[Buchanan 1980] I. Buchanan, "Modelling and Verification in Structured Integrated Circuit Design", PhD Thesis, University of Edinburgh, 1980.

[Chen 1977] K.A. Chen, M. Feuer, K.H. Khokhani, N. Nan and S. Schmidt, "The Chip Layout Problem: An Automatic Wiring Procedure", Proceedings of the 14th Design Automation Conference, 1977.

[Conway 1980] L. Conway, A. Bell and M.E. Newell, "MPC79: The Large-Scale Demonstration of a New Way to Create Systems in Silicon", *Lambda Magazine*, vol. 1, no. 2, 1980.

[Dahl 1972] O.J. Dahl, E.W. Dijkstra and C.A.R. Hoare, *Structured Programming*, Academic Press, New York, 1972.

[Dijkstra 1972] E. W. Dijkstra, "Notes on Structured Programming" from [Dahl 1972].

[Eichelberger 1977] E.B. Eichelberger and T.W. Williams, "A Logic Design Structure for LSI Testability", Proceedings of the 14th Design Automation Conference, 1977.

[Hsueh 1979] M-Y Hsueh, "Symbolic Layout and Compaction of Integrated Circuits" PhD Thesis, University of California at Berkeley, UCB/ERL M 79/80 Memo, 1979.

[Johannsen 1979] D. Johannsen, "Bristle Blocks -- A Silicon Compiler", Proceedings of the 16th Design Automation Conference, 1979.

[Ko 1979] D.C. Ko, "BUILD User's Manual", Burroughs Corporation - Mission Viejo, 1979.

[Koestler 1967] A. Koestler, *The Ghost in the Machine*, Chicago, Henry Regency Co., 1967.

[Lang 1980] C.R. Lang, "LAP User's Manual", California Institute of Technology Computer Science Department Technical Report #3356, 1979.

[Lattin 1981] W. Lattin, "A 32-Bit VLSI Micromainframe Computer System", ISSCC, New York, Feb 1981.

[McGrath 1980] E. J. McGrath and T. E. Whitney, "Design Integrity and Immunity Checking: A New Look at Layout Verification and Design Rule Checking", Proceedings of the 17th Design Automation Conference, 1980.

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

[Mosteller 1981] R. Mosteller, "REST -- Stick Diagram Editing System", MS Thesis, California Institute of Technology, 1981, in preparation.

[Persky 1976] G. Persky, D.N. Deutsch and D.G. Schweikert, "LTX - A System for the Directed Automatic Design of LSI circuits", Proceedings of the 13th Design Automation Conference, 1976.

[Rem 1981] M. Rem and C.A. Mead, "A Notation for Designing Restoring Logic Circuitry in CMOS", Proceedings of the Second Caltech Conference on VLSI, 1981.

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

[Segal 1980] R. Segal, "Structure, Placement and Modelling", MS Thesis, California Institute of Technology, 1980.

[Seitz 1980] C.L. Seitz, "System Timing", from [Mead 1980].

[Simon 1962] H.J. Simon, "The Architecture of Complexity", Proceedings of the American Philosophical Society, vol. 106, no. 6, 1962.

[Sproull 1980] R.F Sproull and R.F. Lyon, "The Caltech Intermediate Form for LSI Layout Description", from [Mead 1980].

[Sutherland 1977] I.E. Sutherland and C.A. Mead, "Microelectronics and Computer Science", *Scientific American*, September 1977.

[Trimberger 1980] S. Trimberger, "The Proposed Sticks Standard", California Institute of Technology Computer Science Department Technical Report #3880, 1980.

[Trimberger 1980a] S. Trimberger, "Combining Graphics and Layout Language in a Single Interactive System", California Institute of Technology Computer Science Department Technical Report #3794, 1980.

[Williams 1977] J.D. Williams, "Sticks -- A New Approach to LSI Design", MSEE Thesis, Massachusetts Institute of Technology, 1977.

[Whitney 1981] T. Whitney, "A Hierarchical Design Rule Checking Algorithm", *Lambda Magazine*, vol. 2, no. 1, 1981.

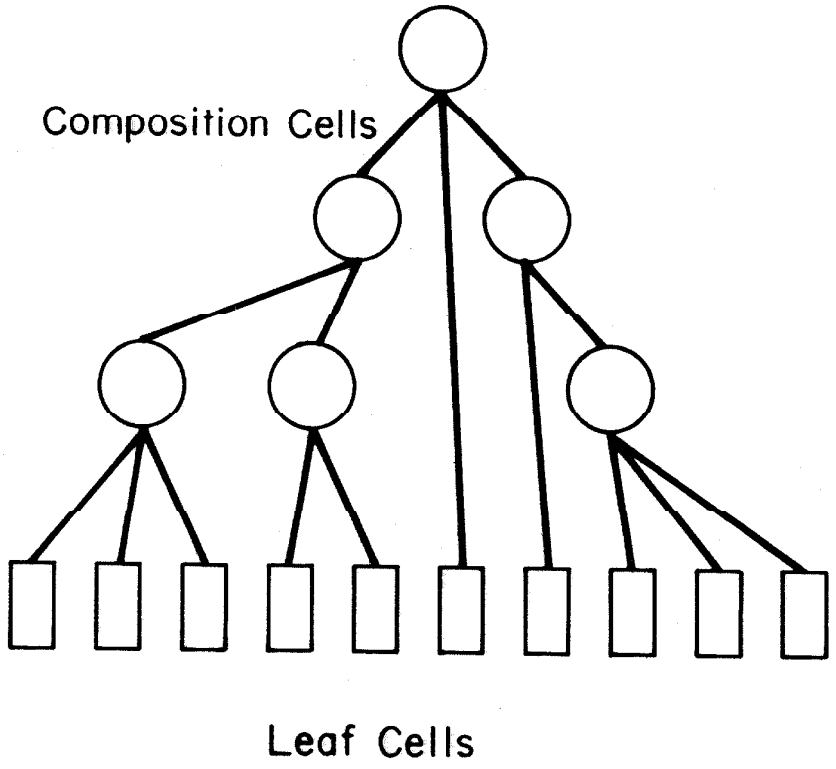


Figure 1.1.1 The Separated Hierarchy

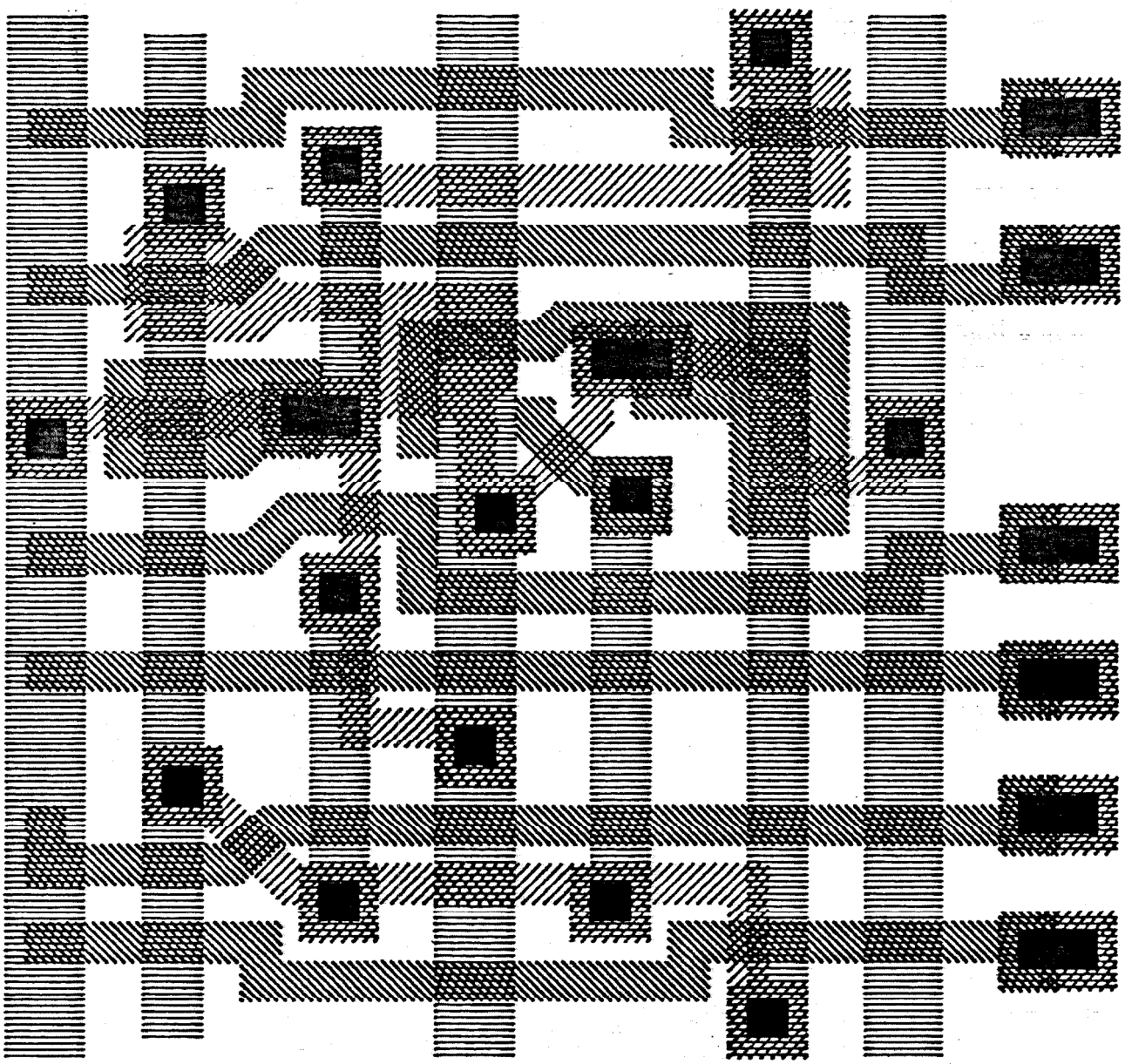


Figure 1.2.1a

Register Cell with All Functions

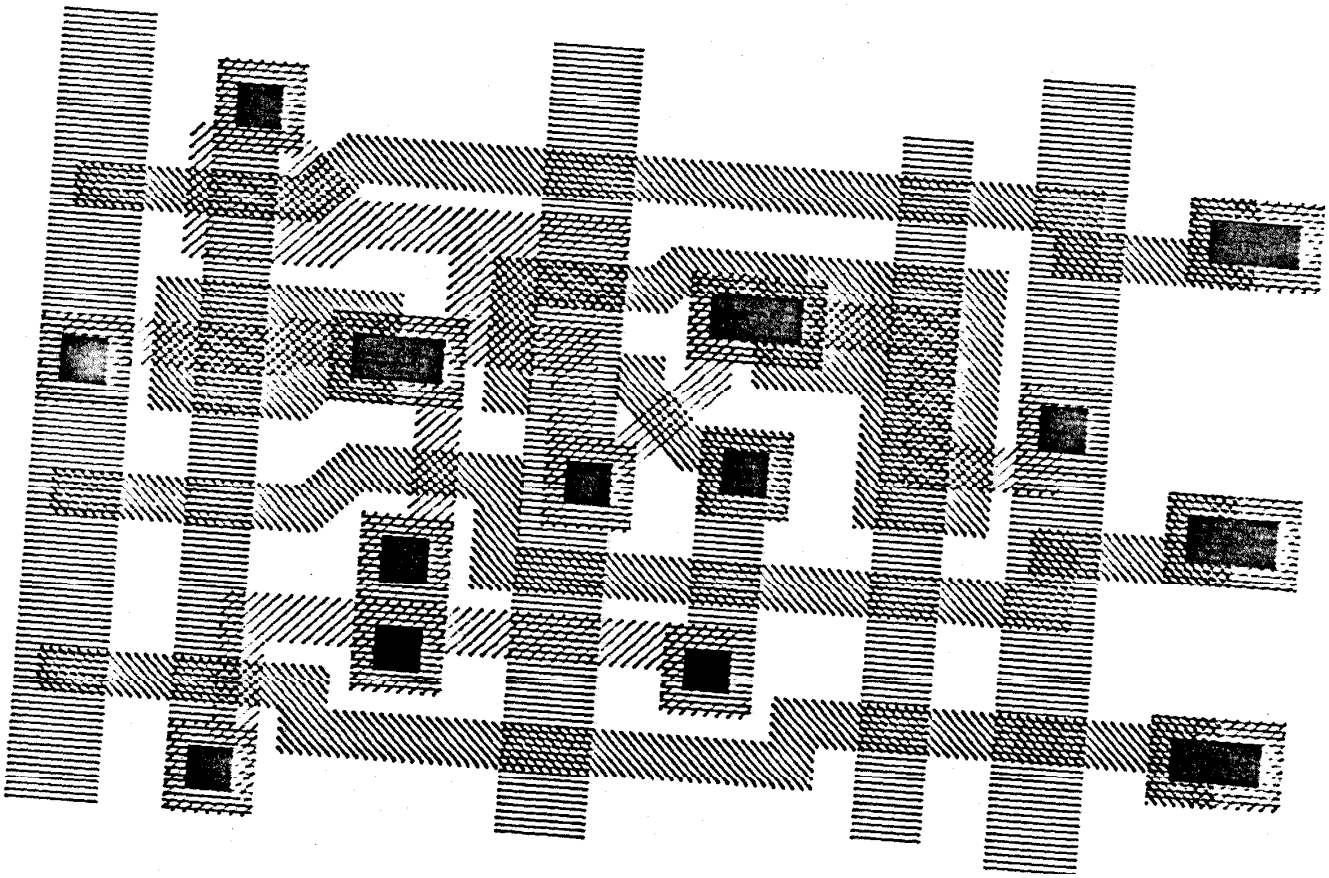


Figure 1.2.1.b

Minimal Register Cell

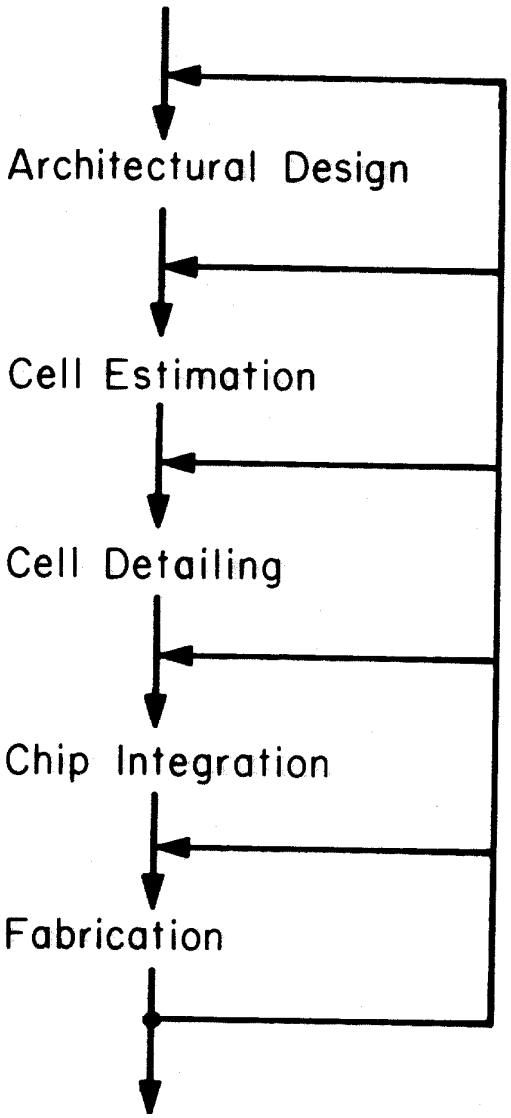


Figure 1.4.1 Design Process Flow

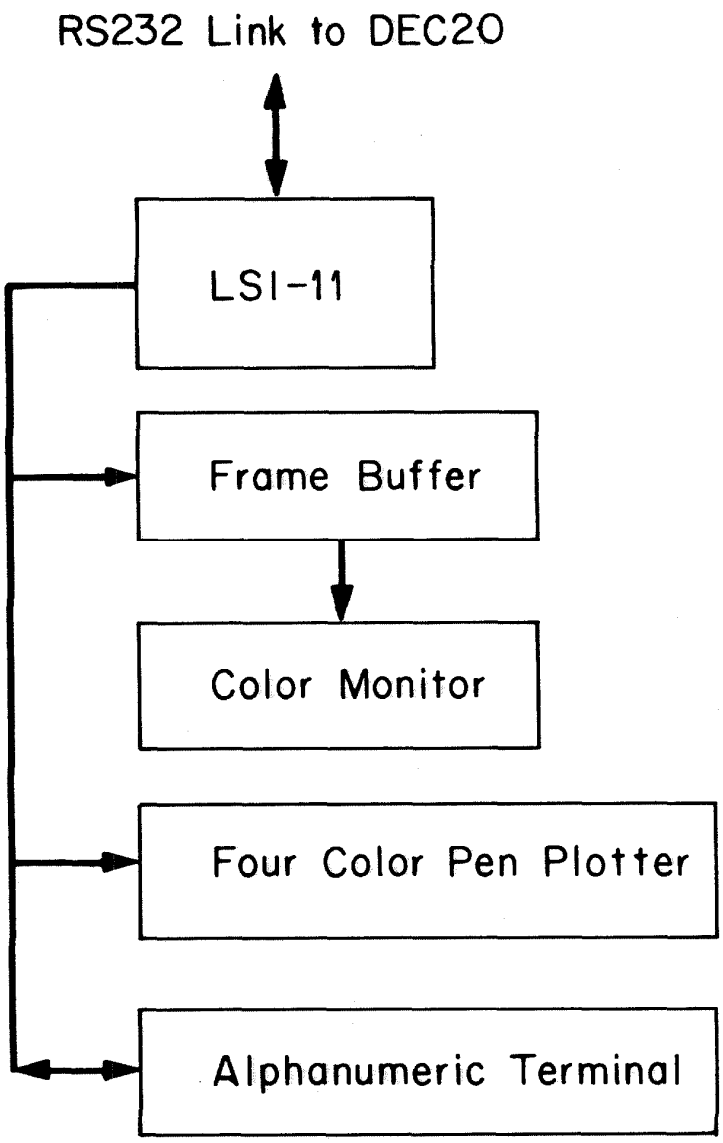


Figure 2.1.1 Graphic Workstation

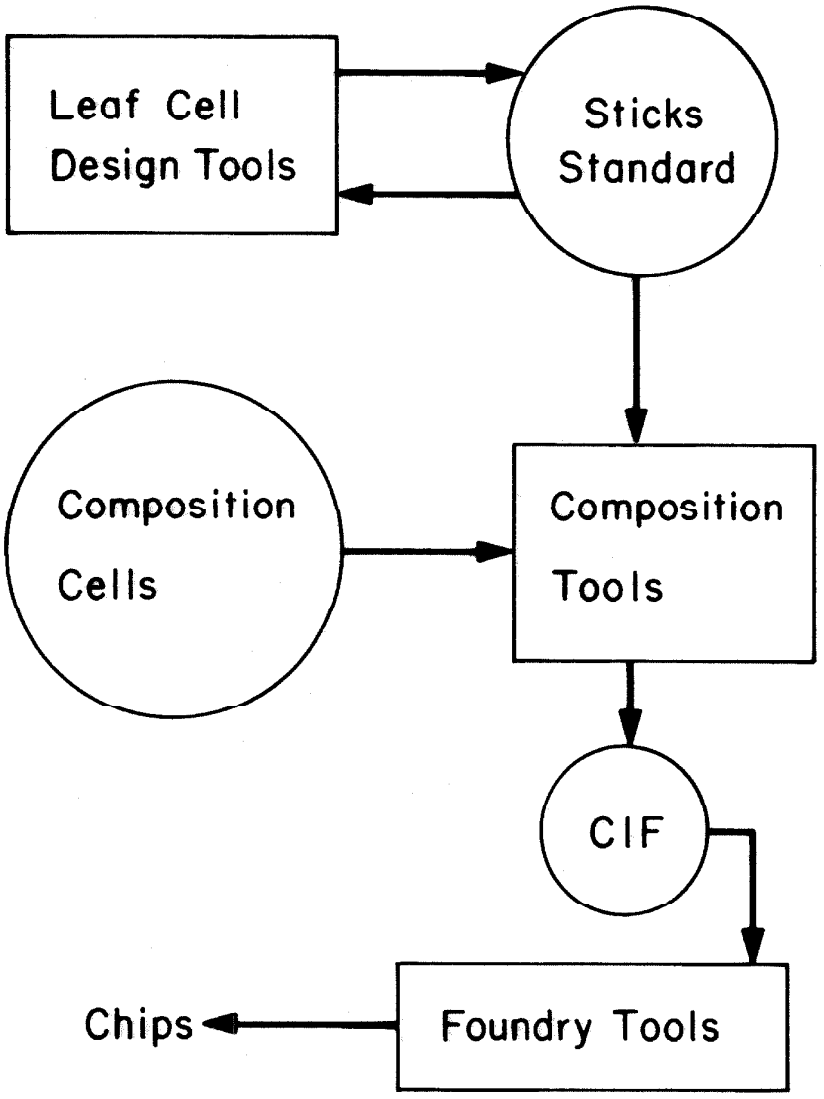


Figure 2.1.2 Tool Communication Overview

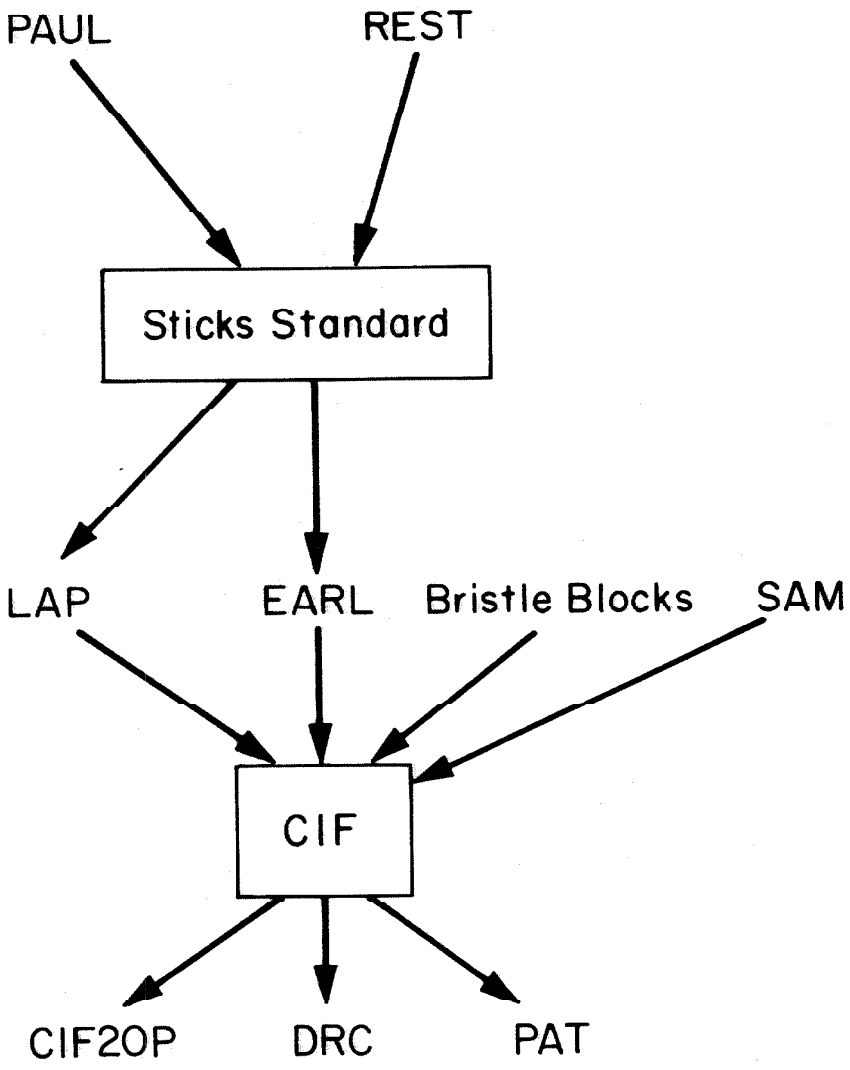
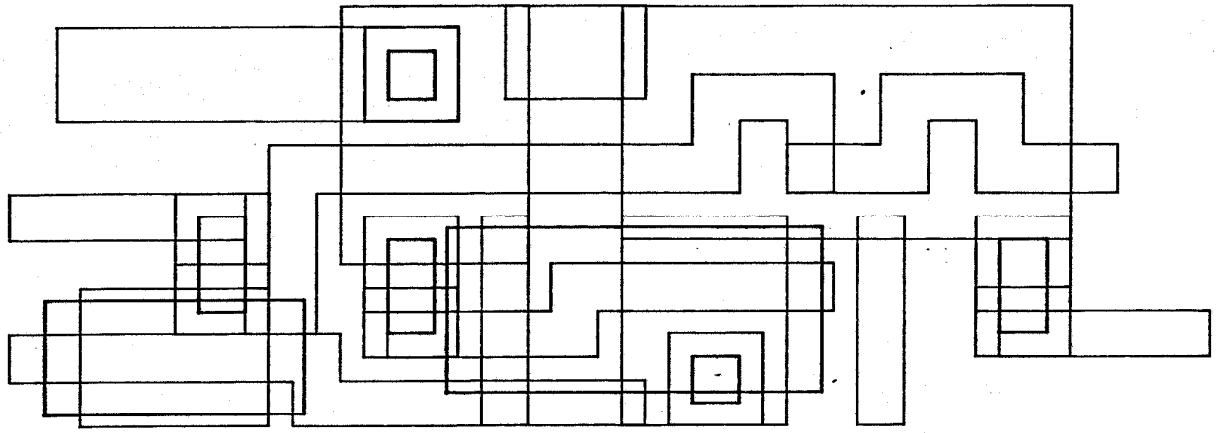
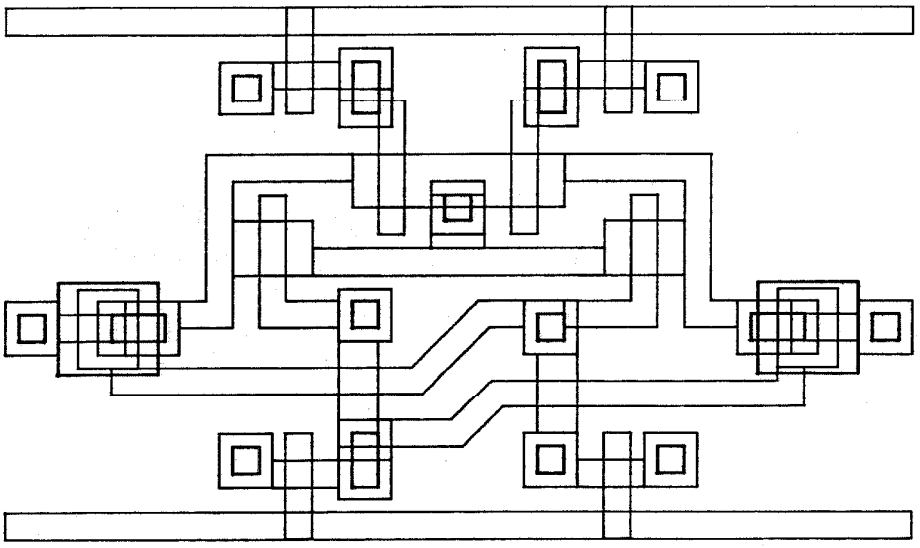


Figure 2.1.3 Tool Overview



Super Buffer Cell



Static Register Cell

Figure 2.2.1 Leaf Cell Examples

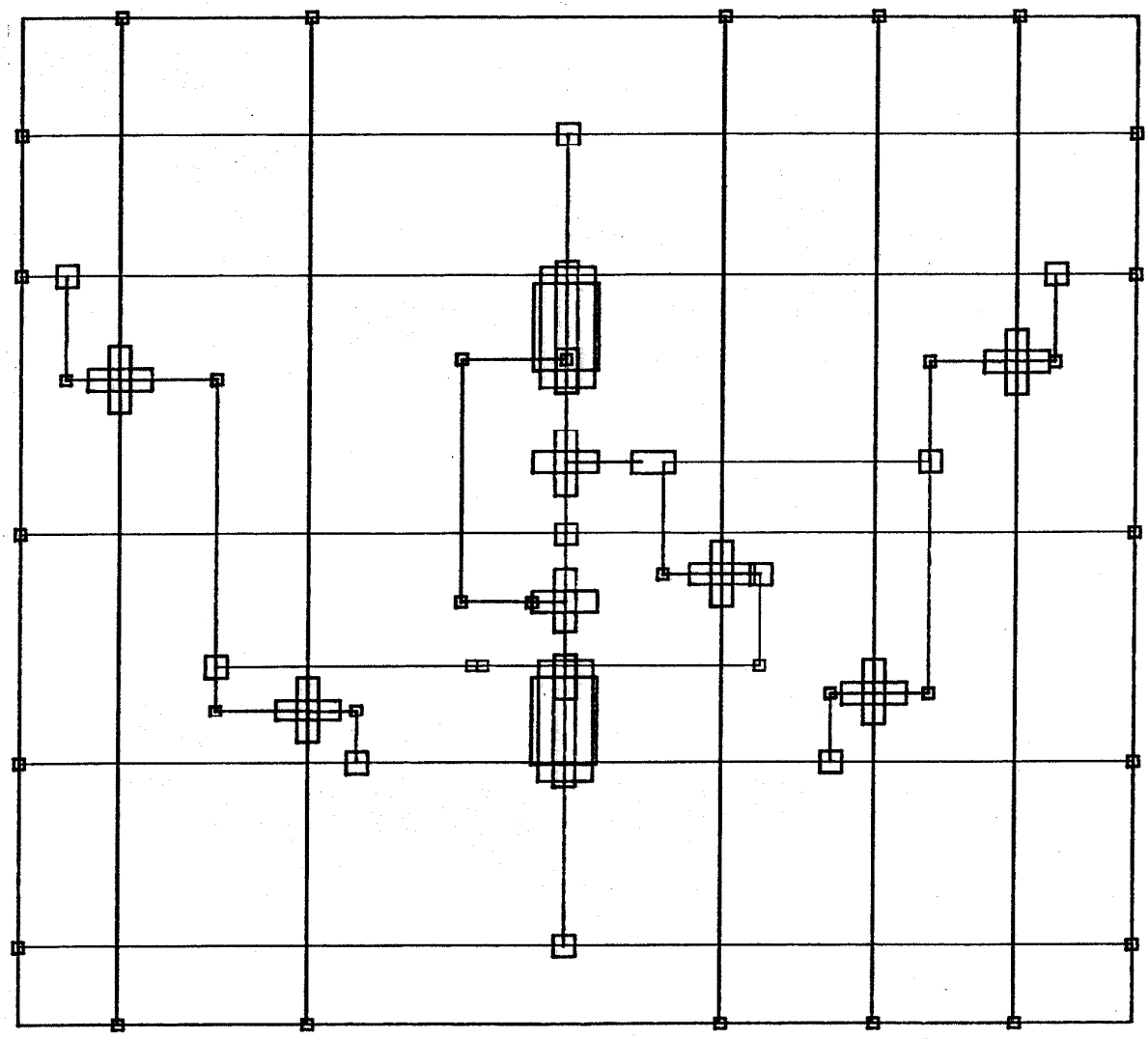


Figure 2.2.2a Uncompacted Stick Diagram

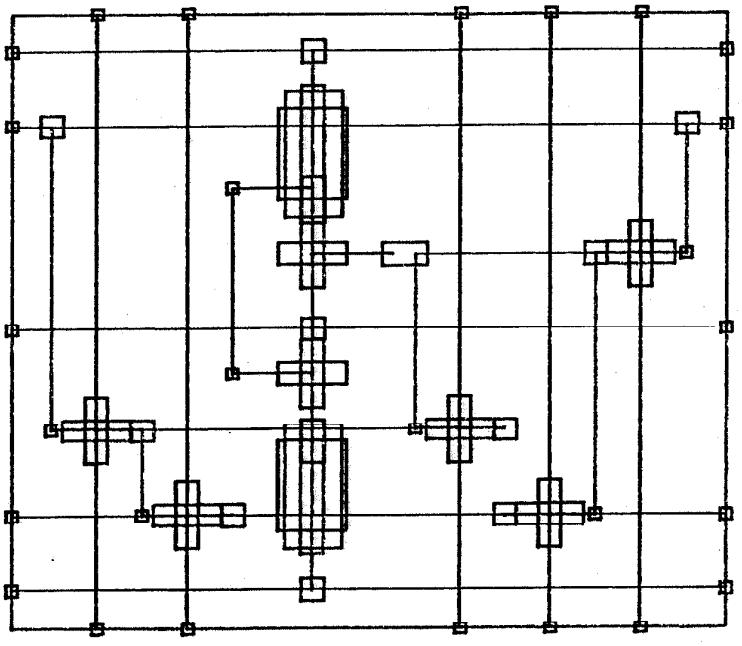


Figure 2.2.2b Compacted Stick Diagram

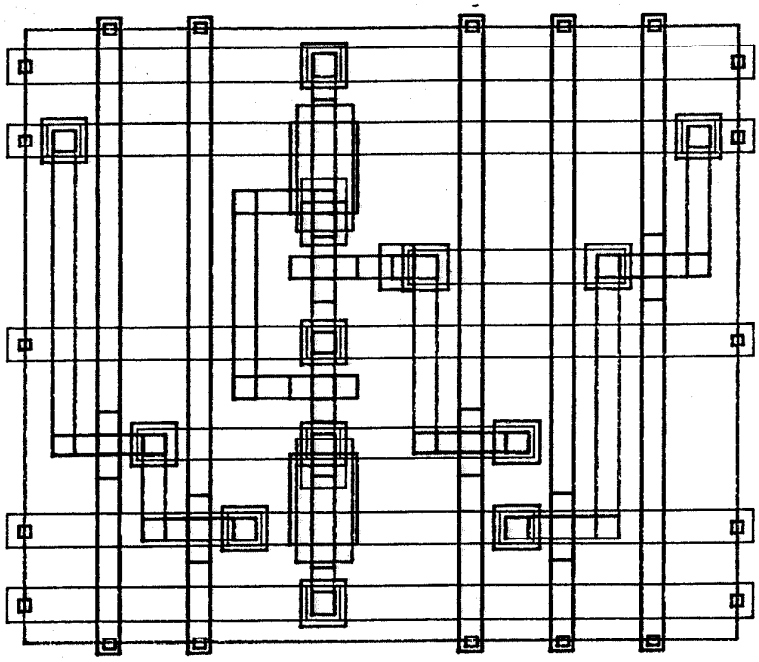


Figure 2.2.3

Layout Generated From Stick Diagram

```
Def SRcell | GNDy | VDDy | INPy | LEFTx |
RIGHTx |
    |Note: Default Note.
    |Box.  Layer: 5.  ll: ¯6+LEFTx,12+VDDy
ur: 13+RIGHTx,16+VDDy.
    |Box.  Layer: 2.  ll: ¯3,12+VDDy ur:
1,16+VDDy.
    |Box.  Layer: 4.  ll: ¯2,13+VDDy ur:
0,15+VDDy.
    |Box.  Layer: 3.  ll: ¯4,5 ur: 2,11.
    |Box.  Layer: 4.  ll: ¯2,3 ur: 0,7.
    |Box.  Layer: 2.  ll: ¯3,2 ur: 1,5.
    |Box.  Layer: 5.  ll: ¯3,2 ur: 1,8.
    |Box.  Layer: 1.  ll: ¯4,3 ur: 2,13.
    |Box.  Layer: 3.  ll: ¯6+LEFTx,¯1+INPy ur:
3,1+INPy.
    |Box.  Layer: 3.  ll: 5,¯6+GNDy ur:
7,16+VDDy.
    |Box.  Layer: 3.  ll: 5 + 4 + 2,¯1+INPy ur:
7+RIGHTx + 4 + 2,1+INPy.
    |Box.  Layer: 5.  ll: ¯6+LEFTx,¯6+GNDy +
2 + ¯2 ur: 13+RIGHTx,¯2+GNDy.
    |Box.  Layer: 4.  ll: ¯2,¯5+GNDy ur:
0,¯3+GNDy.
    |Box.  Layer: 2.  ll: 0,3 ur: 11,5.
    |Box.  Layer: 3.  ll: 9,¯1+INPy ur: 13,2.
    |Box.  Layer: 2.  ll: 9,1 ur: 13,5.
    |Box.  Layer: 4.  ll: 10,0 ur: 12,4.
    |Box.  Layer: 5.  ll: 9,¯1 ur: 13,5.
    |Box.  Layer: 2.  ll: ¯3,¯6+GNDy ur: 1,3.
    |Box.  Layer: 2.  ll: ¯2,¯6 ur: 0,16.
Inst SRcell | t11: 1 | t12: 0 | t21: 0 | t22: 1 |
tx: 0 | ty: 0 |
Params: | ¯7 | 0 | ¯6 | 0 | 4 |
```

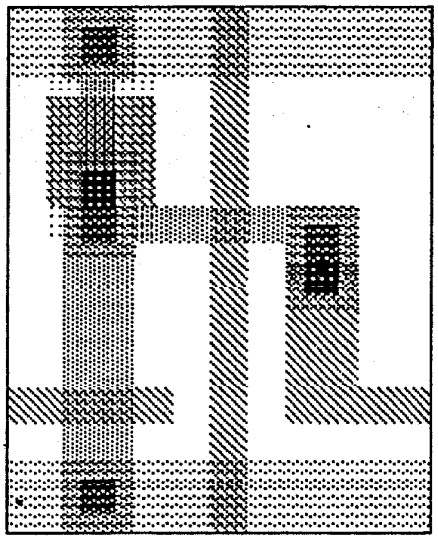


Figure 2.2.4 SAM Display

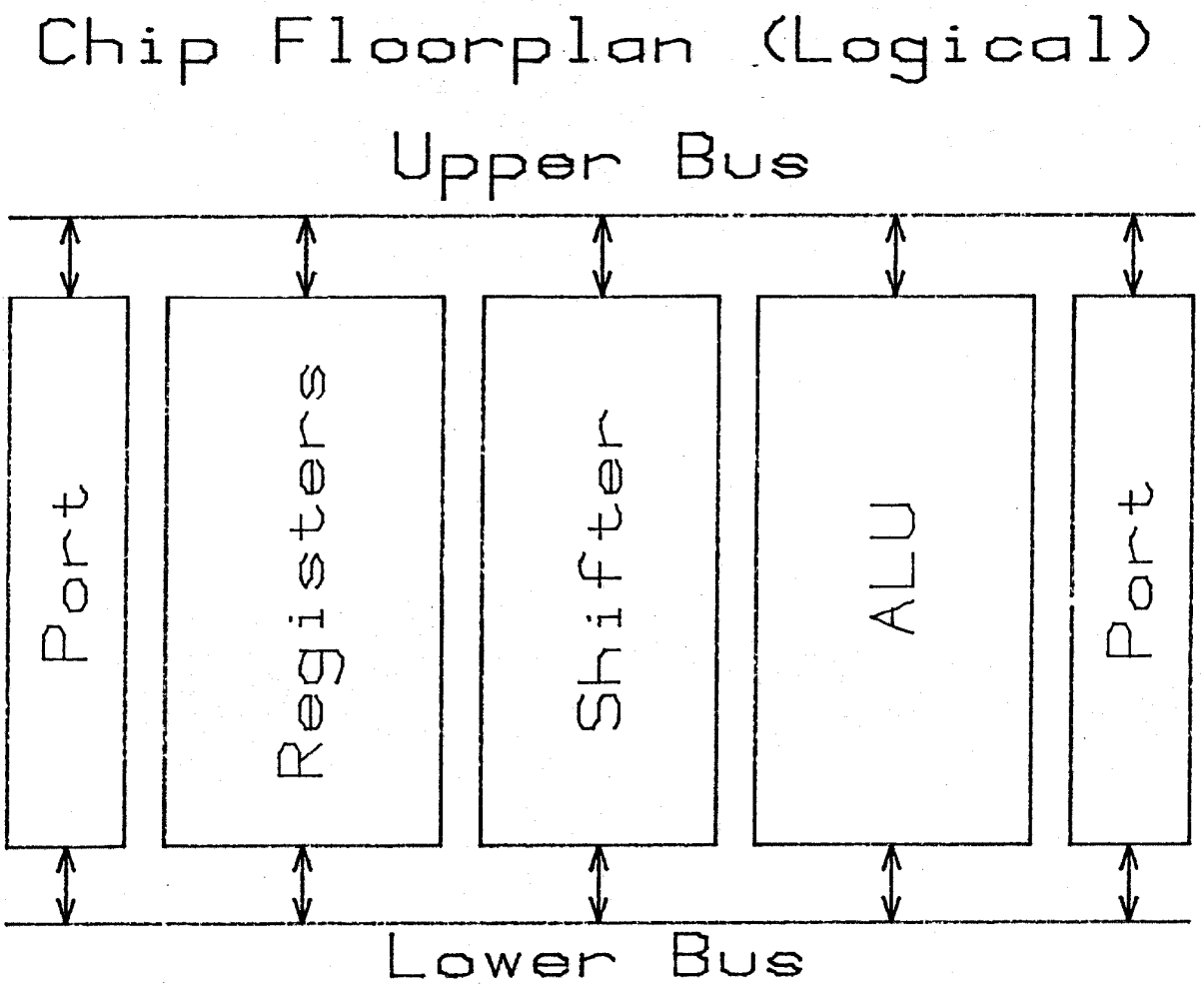


Figure 2.3.1a Chip Block Diagram

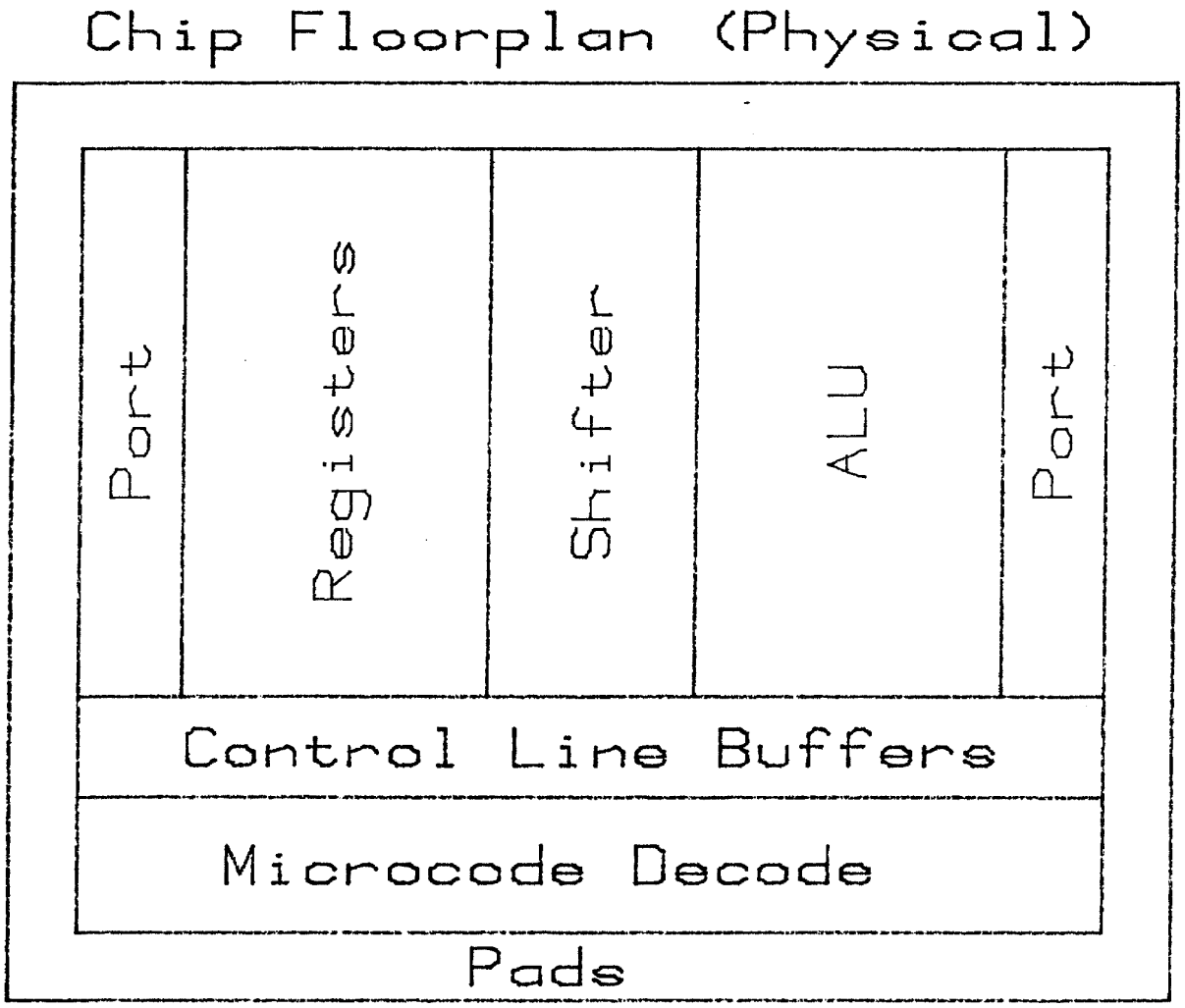


Figure 2.3.1b Chip Floorplan

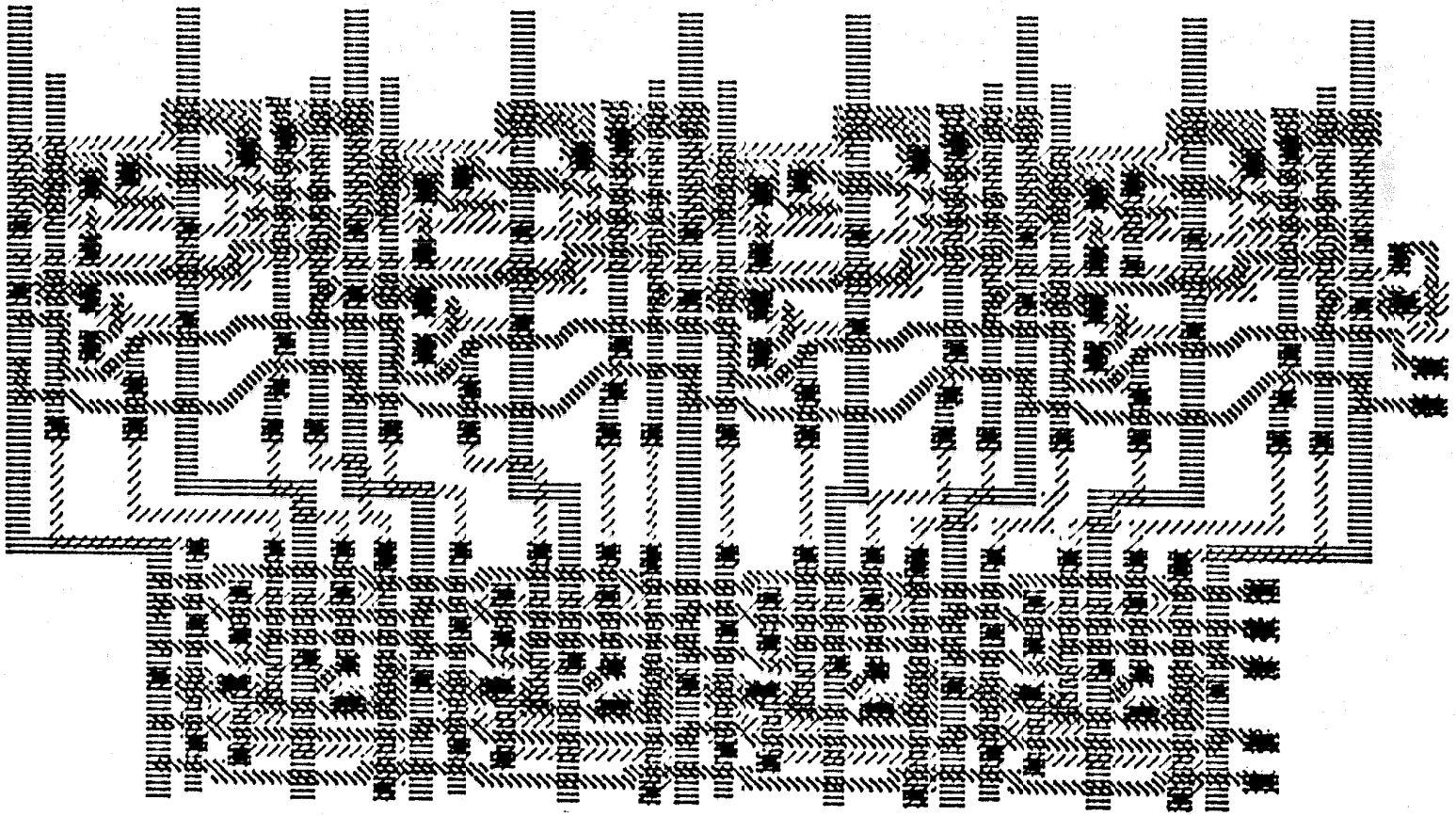


Figure 2.3.2a
Connections Made by Routing

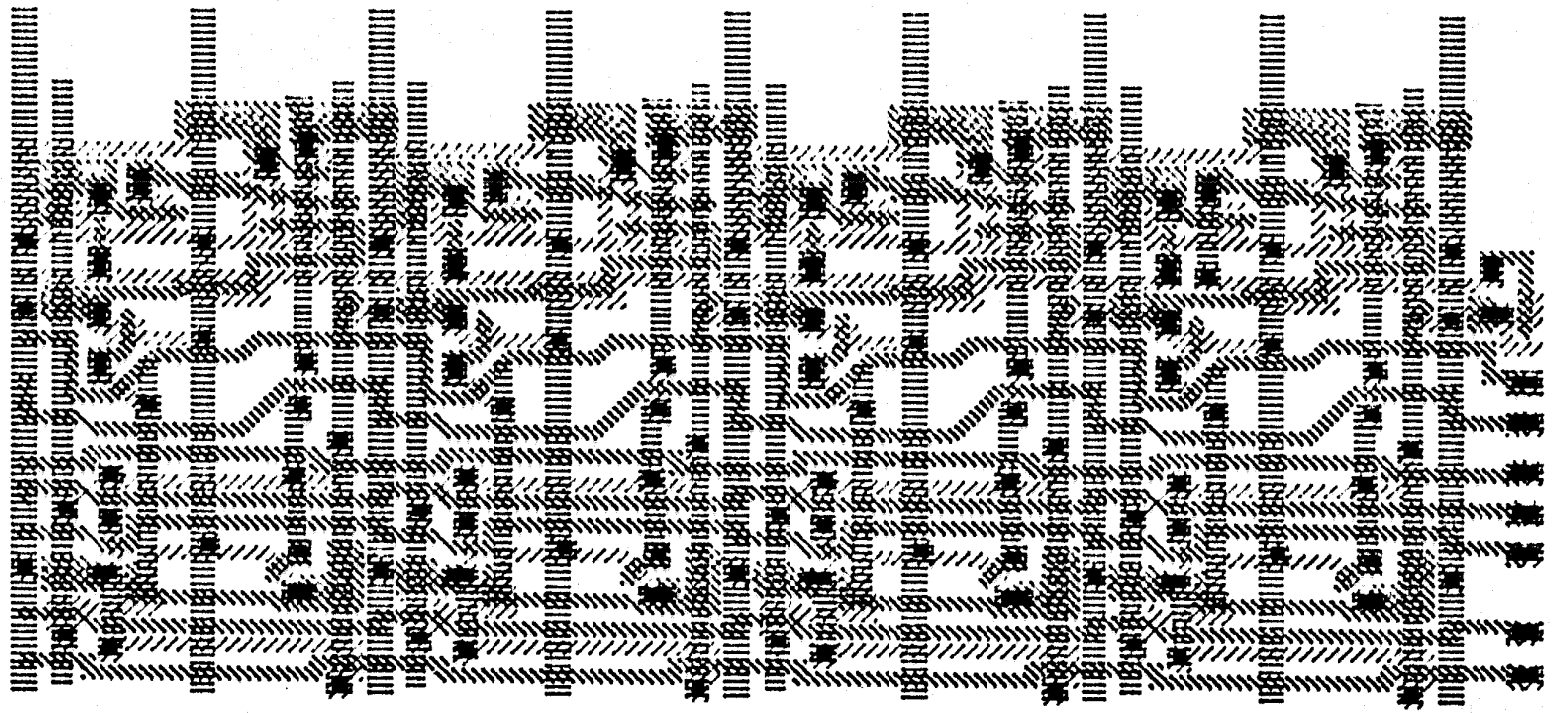


Figure 2.3.2b
Connections Made by Stretching

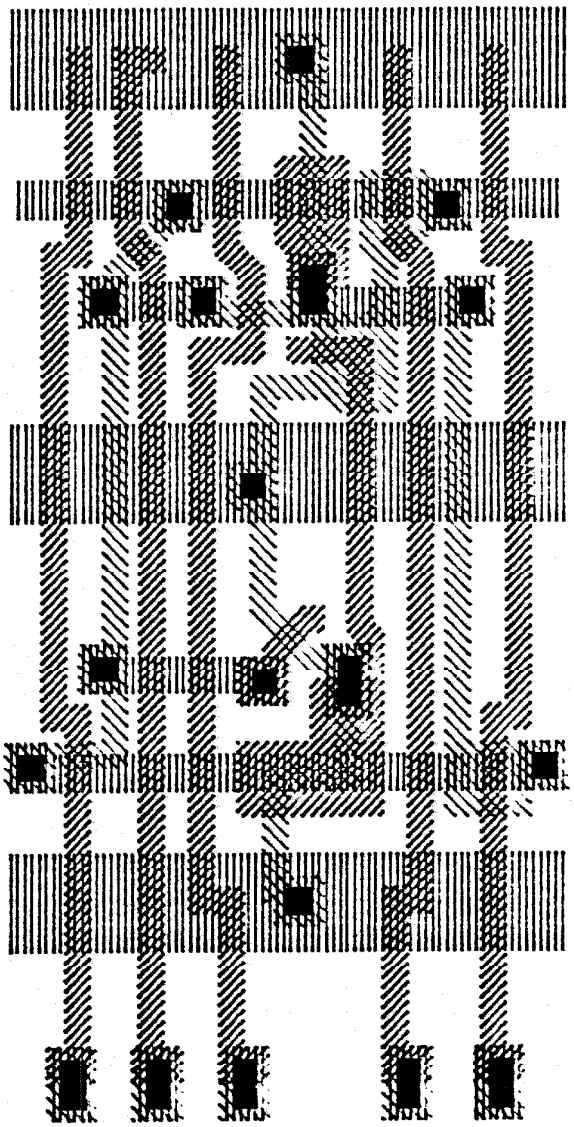
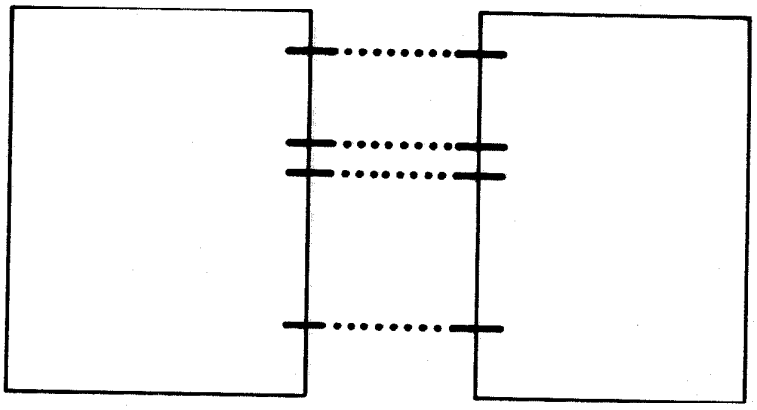


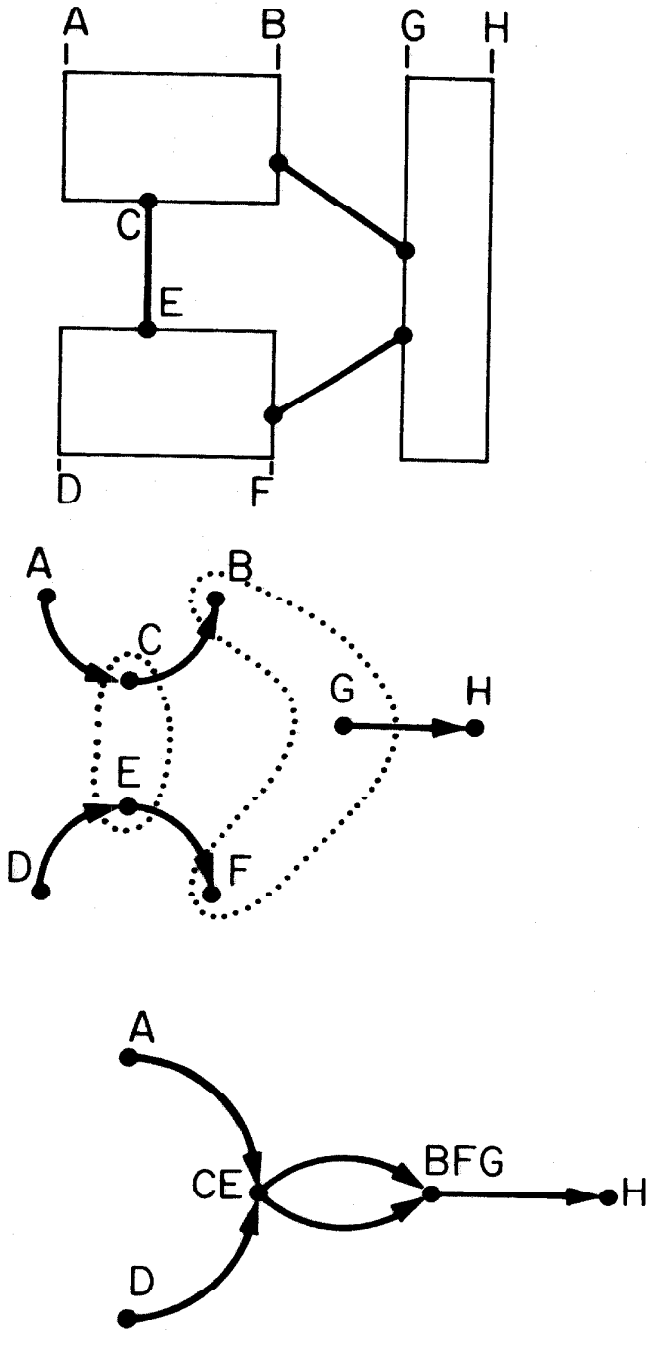
Figure 2.3.3

Stretched Register Cell

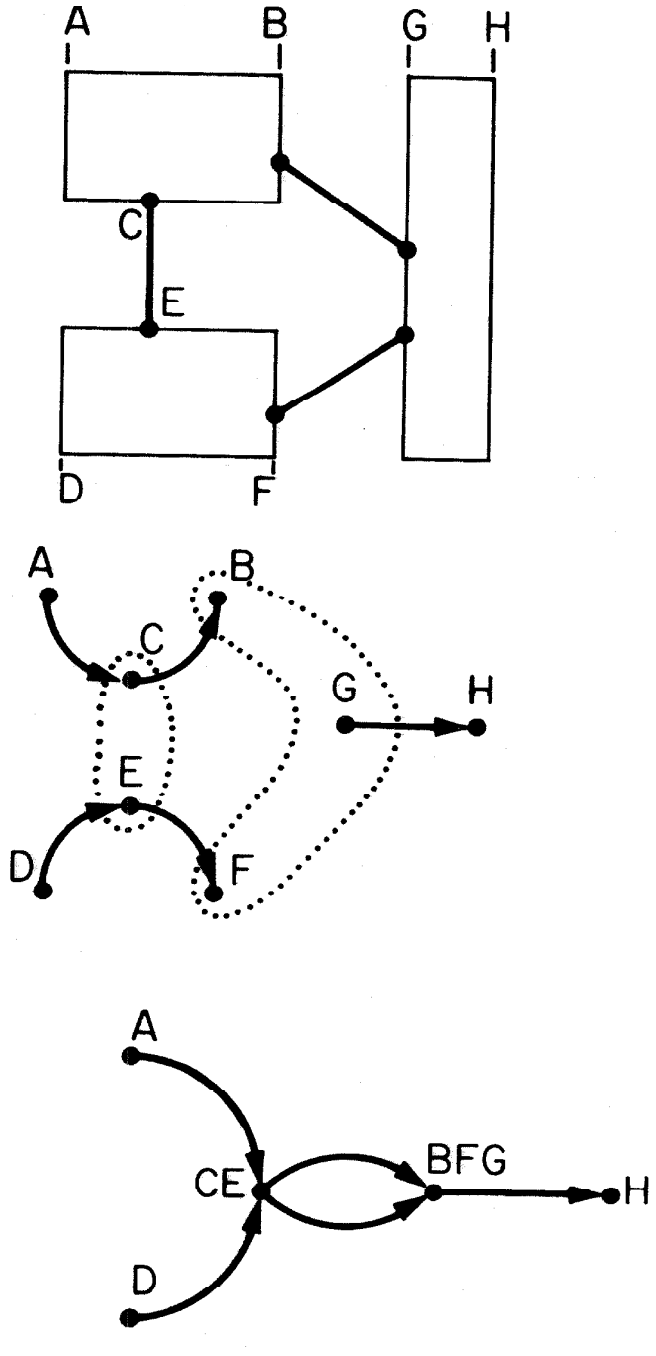


Connection by Abutment

Figure 2.3.4



Solution Graph for Horizontal Direction
Figure 2.3.5



Solution Graph for Horizontal Direction
Figure 2.3.5

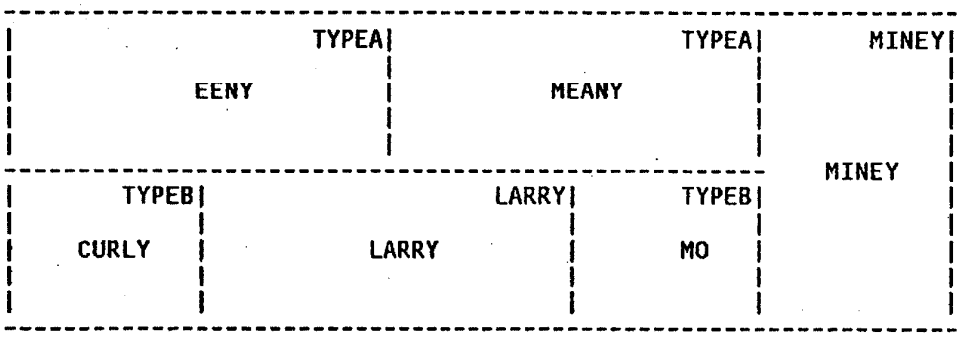


Figure 2.3.6

Example SPAM Floorplan Output

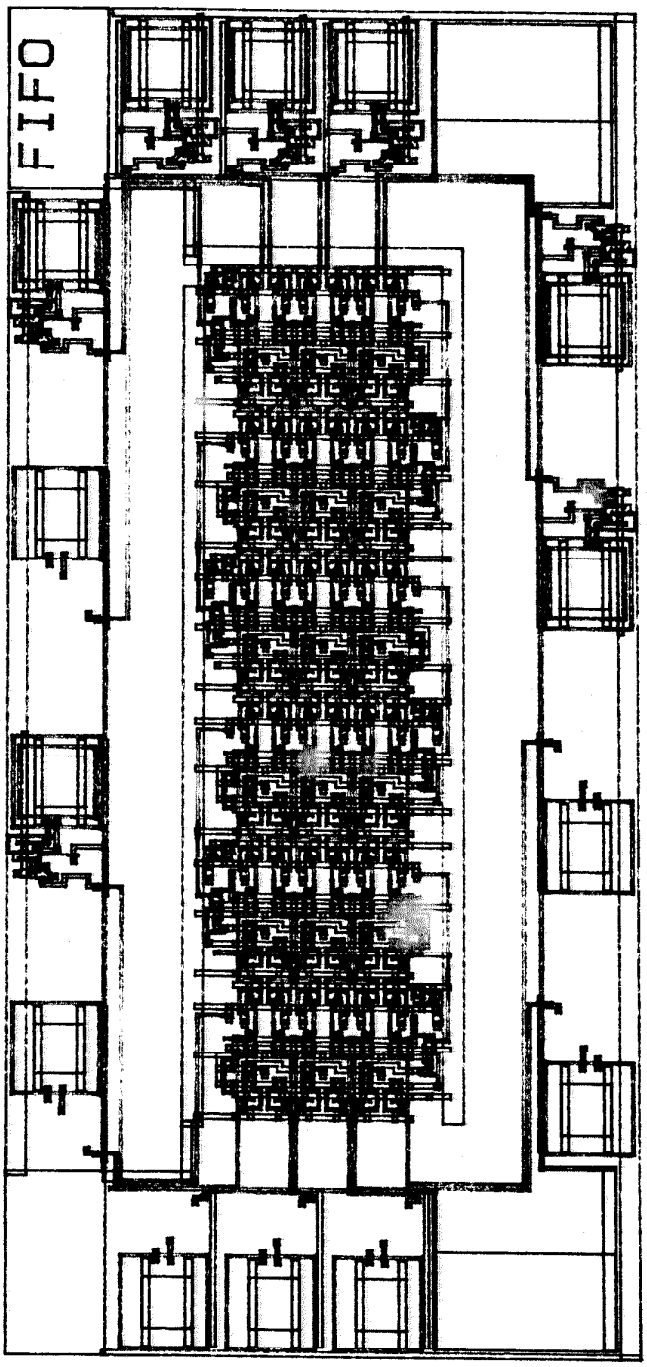


Figure 2.4.1
Design Rule Checker Example

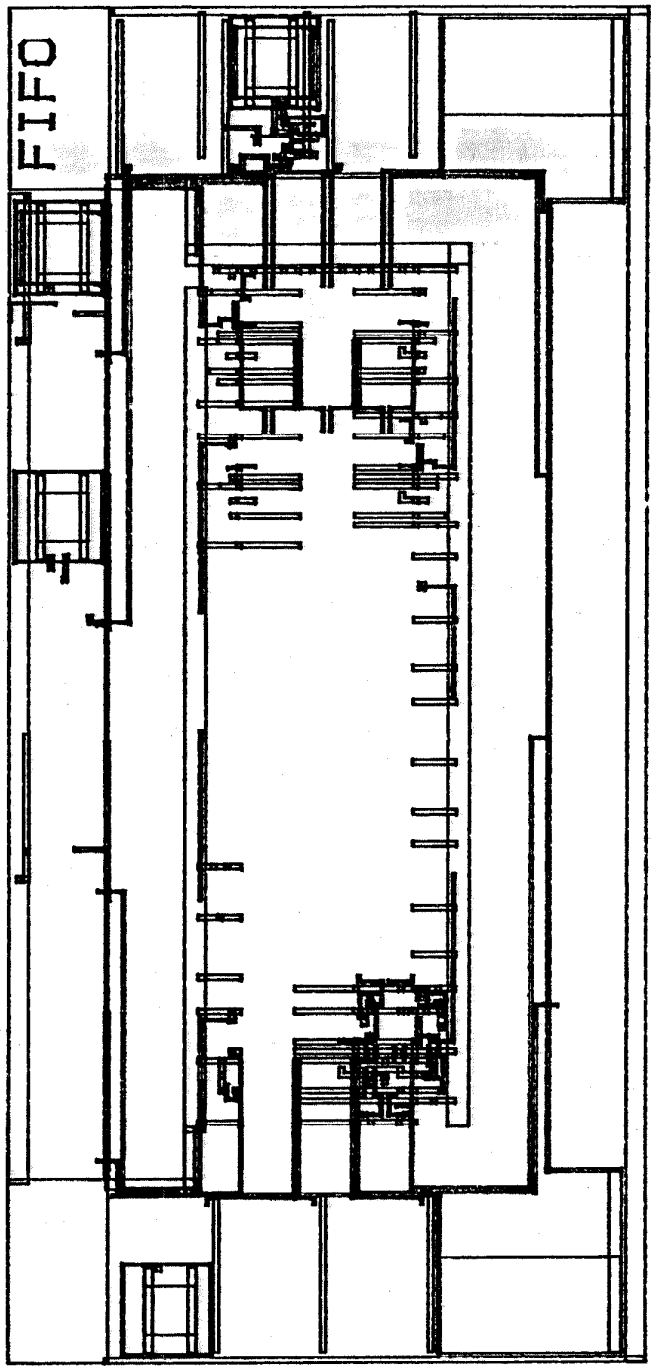


Figure 2.4.2
Shapes Actually Examined by DRC