8-1-1991

A Parameterized CMOS standard cell library and a full 8-bit grey scale morphological array processor

Lawrence H. Rubin

A Parameterized CMOS Standard Cell Library and a Full 8-Bit Grey Scale Morphological Array Processor

by

Lawrence H. Rubin

A Thesis Submitted
in
Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Computer Engineering

Approved by:

Prof. George A. Brown
(Thesis Advisor)

Prof. Tony Chang

Prof. _____
(Department Head)

DEPARTMENT OF COMPUTER ENGINEERING
COLLEGE OF ENGINEERING
ROCHESTER INSTITUTE OF TECHNOLOGY
ROCHESTER, NEW YORK
AUGUST, 1991

Title of Thesis: **A Parameterized CMOS Standard Cell Library and a Full 8-Bit Grey Scale Morphological Array Processor**

I, **Lawrence H. Rubin** hereby **deny** permission to the Wallace Memorial Library of Rochester Institute of Technology to reproduce my thesis in whole or in part.

Date: August 30, 1991

TABLE OF CONTENTS

TABLE OF CONTENTS [continued]

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ALU	Arithmetic Logic Unit
CAD	Computer Aided Design
CIF	Caltech Intermediate Form
CPF	Circuit Path Finder
DRC	Design Rule Checker
ERC	Electrical Rule Checker
HDL	Hardware Description Language
IBM	International Business Machines (Inc.)
LPE	Layout Parameter Extractor
LVS	Layout Versus Schematic
LSI	Large Scale Integration
MAP	Morphological Array Processor
MOSIS	Metal Oxide Semiconductor Implementation System
MSI	Medium Scale Integration
NSA	National Security Agency
RIT	Rochester Institute of Technology
SSI	Small Scale Integration
VLSI	Very Large Scale Integration

Special thanks to Jeff Correll for his help with the cell library and the incorporation of the REMEDI design rule checker into the Chipgraph macros, Shishir Ghate for his help with the cell library and incorporation of the library with Cellstation, Jeff Hanzlik for his help with the logical definition of the MAP, Chris Insalaco for the Morphology Theory section and his help with the Cadence tools and cell library, and Jens Rodenberg for the MAP Architecture section.

Abstract

The creation of a parameterized, full custom CMOS VLSI design library is discussed. This library consists of a schematic component library that is integrated with both logic and circuit level simulators, as well as a corresponding layout cell library that is integrated with automatic place-and-route tools as well as several layout verification tools.

The library enabled the design and implementation of a Morphological Array Processor (MAP). This VLSI chip fully implements the morphological operations of erosion and dilation using a 7x7 matrix. It will operate on a 512x512 image in real time (60 images per second). The chip is designed to be pipelined for multiple successive morphologic operations on a series of images. The MAP is implemented using an 2.0μm N-well CMOS process which can be fabricated through the MOSIS program.

I. Introduction

A significant amount of time is required to generate a standard cell library that will allow the design of VLSI microchips. Large semiconductor companies all have designated groups to generate the libraries for their proprietary processes. Many smaller companies, which have their chips manufactured at a foundry or foundry broker like MOSIS, do not have the resources necessary to generate a new library every time a new process is used. A solution is to have a fully scalable set of design rules, and to generate one library that can be modified as smaller feature sizes become available.

It would also be useful to have a parameterized component library that is designed to allow minimum changes (if any) to schematics to reflect the new technology. This would minimize the amount of time a designer has to rework a chip in order to get better performance by using a smaller fabrication technology. If a set of standard sizes are implemented, and their ratios are preserved from one feature size technology to another, little or no circuit design will need to be done to take advantage of the higher speeds, and layout is trivial.

Without a scalable cell library, a company would have to redo the layout, usually from scratch, as well as redo the circuit design to reflect the changes in technology. This can make the cost of shrinking to a smaller feature size approach the initial development cost. As the industry continues to produce smaller, faster technologies, a company can easily find itself changing over to a new technology every two years.

For logic or circuit simulation, the designer enters the schematic using elements from the component library. These have an advantage over the generic libraries that come with the Mentor Graphics package, as they support both logical and circuit simulation. Using only the Mentor Graphics package, the designer would have to enter a logical and a physical schematic, thus introducing a certain element of unnecessary human error into the design flow. The designer can change a components size by simply editing a text property attached to the component on the schematic.

The component library builds on itself. Except for the basic transistors, each component has a representative symbol (that appears on the schematic) and a schematic of its own. This schematic is composed of other, more primitive components.

The cell library consists of all the implemented sizes of all the components. The convention for naming a cell is the component name followed by the size parameter (which is alphabetic). Thus the "b" sized two input NAND gate is simply called "nand2b". Each cell is designed to work with the automatic place and route tools from Mentor Graphics Corporation.

The two ideas central to the library are the scalable design rules and the parameterizing of the components and cells. The reasons for using scalable rules are enumerated above. The components are parameterized alphabetically. Each letter corresponds to a numeric value in a lookup table located in single text file that is read by Expand, Mentor Graphics' netlister. Thus, by editing that single file, the schematics

can automatically represent a different feature size technology without any required schematic editing.

The overall objective is to create a complete, fully integrated, and flexible environment for the design, layout, and verification of CMOS VLSI microchips, using a scalable design rule set. Specific objectives are:

1. To create a component library: Each component will be a primitive that can be simulated both logically and electrically. The components will be designed in such a way as to ease future scaling or change in the design rules.

2. To create a cell library: Each cell will map to a specific size of a corresponding component. The design rules will be those of MOSIS' scalable N-well CMOS.

3. To Intimately integrate the component and cell libraries into the Mentor Graphics platform, including their schematic capture, automatic place-and-route, layout editing, and simulation tools: The cell layout will also support Cadence's Dracula II layout verification tools.

The library enables the design and implementation of significant CMOS VLSI projects. One such project is a morphological array processor, or MAP. The MAP is designed to perform the morphological image operations of erosion and dilation using a 7x7 mask over an image. The MAP is specified to operate on a 512x512 pixel image in real time (60 images per second), or 16MHz.

II. Historical Review

A. Design Before the Library

As stated before, without the library the designer would have to enter two different sets of schematics to do both logical and physical simulation. This is both inefficient and unreliable, as there is no mechanism to compare the logic level schematic to the equivalent transistor level schematic.

Another drawback of the generic logic level library (gen_lib), supplied by Mentor Graphics, is that every component has a default delay of zero when simulated. This is overly optimistic in most cases, and prevents operation in systems with feedback, such as an SR latch. Race conditions are not modeled, nor are settling glitches of the circuit's outputs. The library solves these problems by simply having all components default to at least one unit delay rise and fall time when being logically simulated.

Before the library was created, there existed no standardized mechanism to actually fabricate designs through the MOSIS program. The SPICE files were obsolete, and overly simplified (they didn't model leakage current or junction capacitance). There were no pad cells.

For the design and implementation of any VLSI projects, especially if they are intended to be fabricated, the compilation of a standard library is not only practical, but necessary and an inevitable evolutionary stage of any CAD center.

B. The First Test Chip

MOSIS currently supports twelve different processes, including 2.0, 1.6, and 1.2 micron CMOS and even a GaAs process. A standard cell library (no components) is available from MOSIS that was developed at the University of Mississippi for the National Security Agency. This cell library is publicly available. When the RIT library was proposed, a test chip was designed and fabricated to compare some prototype cells with the equivalent cells supplied by NSA to determine if there were any advantages to be had by generating a complete library from scratch. The alternative would be to create a matching component library for the NSA cell library.

The test chip consisted of MUXes, D flip-flops, and an 8-bit adder. The MUXes compared the RIT nMOS and CMOS transmission gate muxes against the NSA MUX called "dsel". A D flip-flop with asynchronous resets from each library was used. For overall speed comparison, two 41 stage ring oscillators were created.

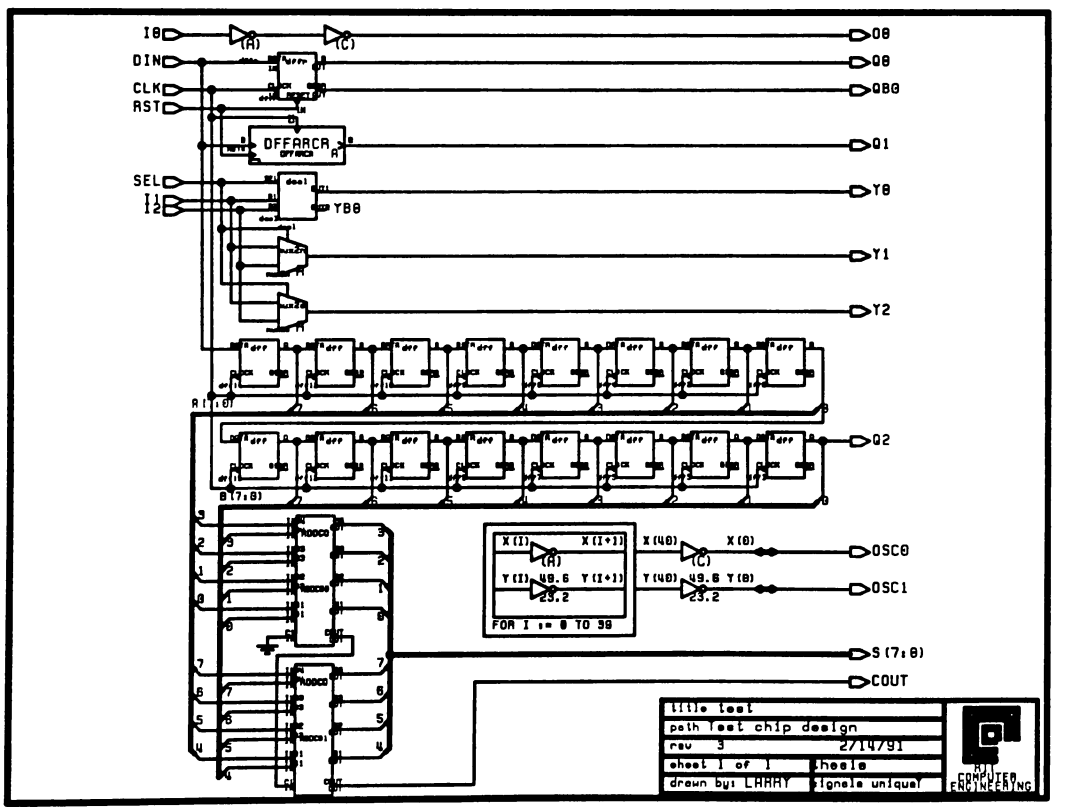

Figure 1 Test Chip #1 Schematic

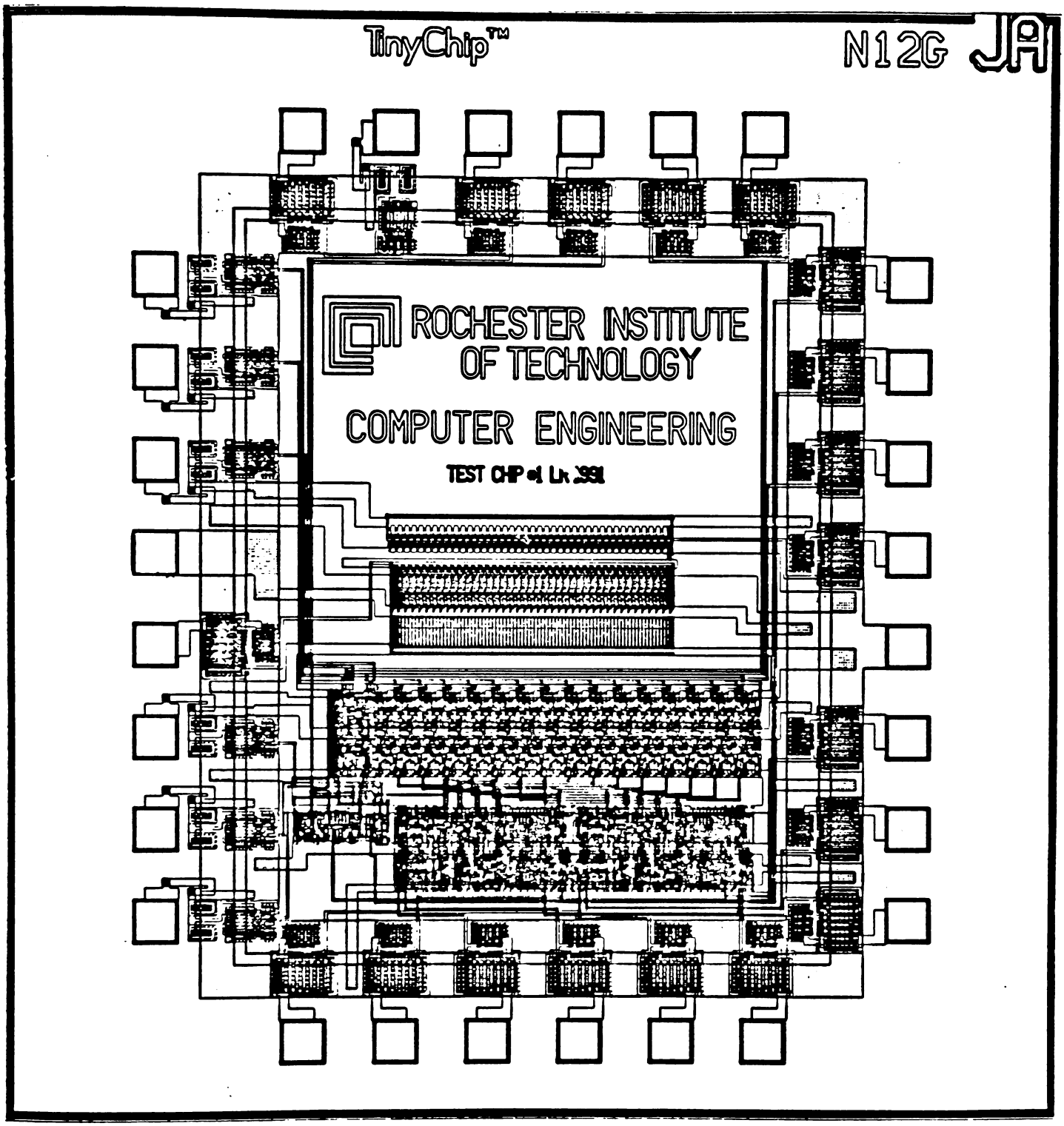

Figure 2 Photograph of Test Chip #1

The RIT cells required 1/3 to 1/2 of the area of the equivalent NSA cells, and had less input capacitance, while having the same drive capabilities. Because of this, the RIT cells were faster. Due to poor contacts and unnecessarily thin power routing in the NSA cells (causing unnecessary internal resistances), the RIT cells were calculated to consume less power.

Another advantage of the RIT cells is that there are several drive sizes available for each component. The inverter, for example has nine different drive sizes for the designer to choose from. The NSA library offers two sizes of inverter, and only one drive size was available for most of the cells. After all of these considerations were taken into account, it was obvious that a better library would result by developing from scratch.

The test chip has the distinction of being the first RIT design to be processed at MOSIS.

C. The Commission of a Morphological Chip Set

In April 1990 the Center for Imaging Science entertained faculty from the Computer Engineering Department to discuss morphology and the morphologic operations of erosion and dilations.

In June 1990 a Computer Engineering graduate student, Jeff Hanzlik, was selected as principal investigator for the exploration of a prototype morphologic platform. It was decided to implement a 7x7 morphological array processor on a printed circuit board which can be inserted into an IBM AT compatible machine and operate on 512x512 pixel 8-bit gray scale images.

In September 1990 another Computer Engineering graduate student, Jens Rodenberg joined the project, initially working with simulations of morphological operations. Based on these simulations, he developed a new architecture by November, whereby the invalid pixels (represented mathematically by $-\infty$) were implicitly defined via control bits, as opposed to the original architecture which required explicit $-\infty$'s in the data path. The new architecture also used enhanced pipelining, thereby increasing the efficiency of the processor.

An initial goal was to fit the prototype onto an AT standard card. This proved to be impossible, as the Actel gate array chips that were used did not allow a high enough level of integration. It was also originally desired to have the prototype operate in real time. Again, the technology prevented the desired performance from being achieved. In order to meet these goals, full custom VLSI was required. One chip would replace 23 of the gate arrays for the processor array itself, and a smaller, more integrated 2 chip set would replace 5 gate arrays of the control logic, while also increasing performance.

III. Theory

A. Design Automation

For the purposes of this thesis the methodology for a top-down design is partitioned into ten steps. These steps and their flow are shown in figure 3.

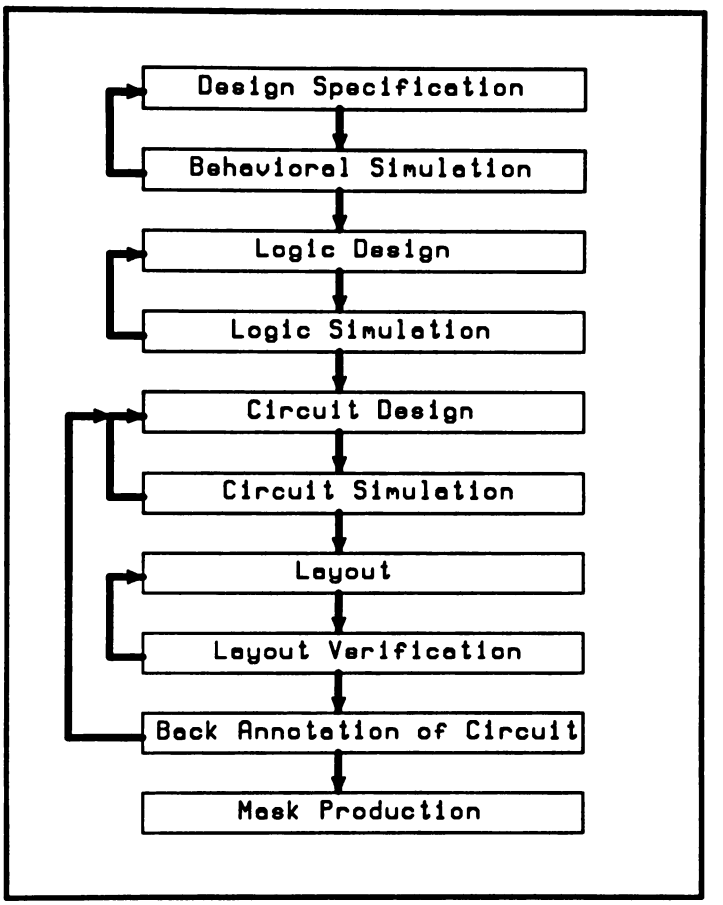

Figure 3 Design Flow

The design specification, or architectural step consists of creating a definition of what the chip is to do. This is accomplished by generation of block diagrams, timing charts, and behavioral models. Mentor Graphics supports models written in C, Pascal, FORTRAN, or VHDL.

For behavioral simulation, some simple test vectors can be applied to the behavioral model to verify that it performs as expected. Mentor Graphics' logic simulator, Quicksim is used for this.

Logic design consists of schematic capture of the gate level implementation of the design. Mentor's Neted is used for this.

During logic simulation the design is checked for functionality. The timing of the gates is in unit delays. Again, Mentor's Quicksim is used for this.

The circuit design sizes each gate to drive its load efficiently, and may cause minor alterations in the logic design in order to achieve timing requirements. Mentor's Neted is used to alter the schematics. Please note that the same symbols are to be used for both logical and circuit simulation. This is an enhancement to the libraries originally supplied by Mentor.

During circuit simulation the design is checked against timing requirements. Timing is based on transistor level models. This includes the loading capacitances caused by transistor gate and junction capacitances. After the layout line capacitances are extracted they are back annotated into the circuit database and re-simulated to provide a much more accurate timing analysis. Mentor's Accusim (a SPICE simulator) performs this step.

The layout of the chip consists of several steps; cell placement and routing for each block, block placement and routing for the chip, power line routing, and pad placement. A combination of tools contained in Mentor's Chipgraph and Cellgraph packages are used for these tasks.

Layout verification is divided into three parts; design rule checking (DRC), electrical rule checking (ERC), and layout vs. schematic comparison (LVS). DRC verifies that the design rules (line widths, spacings, etc.) are met. ERC checks the circuit extracted from the layout against electrical errors such as shorts and floating nodes. LVS verifies that the circuit extracted from the layout is the same as the circuit designed. Cadence's Dracula II is used for all these functions, as well as extracting the parasitic capacitances and diodes from the layout so that they can be back annotated in the circuit design to enable more accurate circuit verification.

Converting the Mentor Chipgraph database into a mask format accepted by MOSIS (Caltech Intermediate Form, CIF) is done by a combination of scripts to be written and Mentor's MCIF package.

B. Morphology

Morphological image processing refers to the analysis and processing of an image based upon a knowledge of its structure or form with the intent of modifying that form or structure. The theoretical basis for morphological image processing dates back to the work of H. Minkowski on spatial set algebra. Based upon this work G. Matheron and J. Serra of France and S. Sternberg of the U.S. have developed a set-theoretic approach to image processing. In this approach binary images are treated as sets in a background space which can then be acted upon by the usual set operations of union and intersection, in conjunction with another image set called a structuring element. The structuring element is used to probe or fit the image to extract information about its shape.

For gray scale images the set operations of union and intersection become the maximum and the minimum functions, respectively.

The two fundamental morphological operations are dilation and erosion. Dilation of an image yields an image which is uniformly larger than the original image while erosion of an image yields a uniformly smaller image. Building upon the basic morphological operations a large variety of morphological filters can be generated for such applications as edge detection, segmentation, and enhancement of images. Morphological filters can also be used instead of standard linear filters. The basic morphological operators will be derived using the Euclidean model then the digital and gray scale models will be presented.

EUCLIDEAN MORPHOLOGY

Morphological operations are built upon the set operations union and intersection as well as the translation and reflection operations. Given an image A in R^2 the translation of A by the point x in R^2 is defined by:

$T_x A = \{a+x : a \in A\}$ where the plus sign refers to vector addition

Considering the point x to be a vector in the plane, $T_x A$ is A translated along the vector x. A reflected image, $B^\wedge$ in R^2 is one that has been rotated 180^0 around the origin or alternatively one that has been flipped from left to right and from top to bottom. The first fundamental operation in morphological analysis is Minkowski addition. Given two images A and B in R^2, the Minkowski sum is defined as:

$$A \oplus B = \bigcup_{b \in B} T_b A = \bigcup_{b \in B} \{a + b : a \in A\}$$

$A \oplus B$ is constructed by translating A by each element of B and then taking the union of all the resulting translates. The second fundamental morphological operation is Minkowski subtraction. Given two images A and B in R^2, Minkowski subtraction is defined as:

$$A \ominus B = \bigcap_{b \in B} T_b A = \bigcap_{b \in B} \{a + b : a \in A\}$$

$A \ominus B$ is constructed by translating A by every element of B then the intersection of the resulting translates is taken.

In morphological processing the Minkowski sum is referred to as a dilation, and denoted as:

$D(A,B) = A \oplus B$

The morphological operation erosion has two different definitions in the literature. Serra defines erosion as Minkowski subtraction, denoted as:

$$E(A,B) = A \ominus B$$

whereas Sternberg and Matheron define erosion as:

$$E(A,B) = A \ominus \hat{B} = \bigcap_{b \in \hat{B}} T_b A$$

This second definition of erosion will be used for our purposes as it lends itself better to a digital implementation. The second image B is generally referred to as the structuring element. If $B = \hat{B}$, as it usually does, then erosion again becomes equal to Minkowski subtraction. Another form of the Minkowski subtraction equation is very useful in image processing, it states that the Minkowski subtraction of B from A is composed of all elements x in R^2 such that B translated by x is a subset of A, where $B \neq \varnothing$. This is denoted as:

$$A \ominus B = \{x : T_x \hat{B} \subset A\}$$

This equation can also be written as:

$$A \ominus \hat{B} = \{x : T_x B \subset A\}$$

An alternate definition of erosion can then be written as:

$$E(A,B) = \{x : T_x B \subset A\}$$

Another form of the Minkowski addition (dilation) is also very useful. It states that the Minkowski sum is composed of all elements x in R^2 such that $\hat{B}$ translated by x is not a subset of A, where $B \neq \varnothing$. This is denoted as:

$$D(A,B) = \{x : T_x \hat{B} \not\subset A\}$$

These are the forms of the erosion and dilation equations that are used in the digital implementation.

Erosion and dilation are often performed in succession on an image. A dilation followed by an erosion is called a "closing" operation and is denoted:

$$C(A,B) = E(D(A,\hat{B}), \hat{B}) = (A \oplus \hat{B}) \ominus B$$

An erosion followed by a dilation is called an open operation and is denoted by:

$$O(A,B) = D(E(A,B), B) = (A \ominus \hat{B}) \oplus B$$

Some of the properties of Minkowski addition, Minkowski subtraction, opening and closing will now be presented.

$A \oplus B = B \oplus A$ addition is commutative

$A \ominus B \neq B \ominus A$ subtraction is not commutative

$(A \oplus B) \oplus C = A \oplus (B \oplus C)$ addition is associative, this allows structuring

elements to be decomposed and chained to achieve the effect of larger structuring elements from small structuring elements

$(A \oplus B)^c = A^c \ominus B$

$(A \ominus B)^c = A^c \oplus B$ dilation and erosion are dual properties, the dilation

of the background of an object is the same as the erosion of the object, and vice versa

if $A \subseteq B$ then $A \oplus B \subseteq B \oplus C$

if $A \subseteq B$ then $A \ominus B \subseteq B \ominus C$ addition and subtraction are both increasing operations

$A \oplus (T_x B) = T_x (A \oplus B)$

$A \ominus (T_x B) = (T_x A) \ominus B = T_x (A \ominus B)$ addition and subtraction are both translation invariant

$O(A,B) \subseteq A$ open is antiextensive, the open operation tends to decrease the spatial extent of an image

$C(A,B) \supseteq A$ close is extensive, the close operation tends to increase the spatial extent of an image

$O(O(A,B), B) = O(A,B)$

$C(C(A,B), B) = C(A,B)$ open and close are idempotent (repeated applications have no effect)

if $A \subset F$ then $O(A,B) \subset O(F,B)$

if $A \subset F$ then $C(A,B) \subset C(F,B)$ open and close are increasing operations

DIGITAL MORPHOLOGY

Let F (j,k) be a binary valued image matrix. A pixel at coordinate (j,k) is an element of image F (j,k) if and only if it is a logical one. An image B (j,k) is a subset of an image A (j,k) if for every logical one pixel in B (j,k), there is a logical one pixel in A (j,k). An image F^c (j,k) is the complement of F (j,k) if all the pixels in F^c (j,k) are the opposite logically of those in F (j,k). A reflected image $F^\wedge$ (j,k) is formed by flipping the image matrix F (j,k) from left to right and from top to bottom. Translation of an image consists of shifting the image by r rows and c columns from the origin. This is denoted as:

$$G(j,k) = T_{r,c}\{F(j,k)\}$$

For the following definitions assume an NxN image matrix F (j,k) and an LxL, where L is odd, structuring element matrix H (j,k). Minkowski addition is defined, similar to the Euclidean case, as:

$$G(j,k) = \bigcup_{(r,c)\in H} T_{r,c}\{F(j,k)\}$$

The resulting image size is MxM where M = N + L -1. The definition of dilation is then:

$$G(j,k) = F(j,k) \oplus H(j,k)$$

Minkowski subtraction is similarly defined as:

$$G(j,k) = \bigcap_{(r,c)\in H} T_{r,c}\{F(j,k)\}$$

and the definition of erosion is then:

$$G(j,k) = F(j,k) \ominus H^\wedge(j,k) = \bigcap_{(r,c)\in H^\wedge} T_{r,c}\{F(j,k)\}$$

Another definition of dilation, analogous to the alternate definition in the Euclidean model, based on the scanning and processing of H (j,k) over F (j,k) is:

$$G(j,k) = \bigcup_{(m,n)} \{F(m,n) \cap H(j-m+S, k-n+S)\}$$

where $\max[1, j-Q] \le m \le \min[N, j+Q]$ and $\max[1, k-Q] \le n \le \min[N, k+Q]$ and S = (L + 1)/ 2 and Q = (L -1)/ 2. The alternate definition for erosion is similarly defined as:

$$G(j,k) = \bigcap_{(m,n)} \{F(m,n) \cup H(j-m+S, k-n+S)\}$$

where the same limits apply as for dilation. The digital opening and closing equations can also be defined analogously to the corresponding Euclidean operators as:

$$O(F,H) = D(E(F,H), H) = [F(j,k) \ominus H\hat{}(j,k)] \oplus H(j,k)$$

$$C(F,H) = E(D(F,H\hat{}), H\hat{}) = [F(j,k) \oplus H\hat{}(j,k)] \ominus H(j,k)$$

GRAY SCALE MORPHOLOGY

Let F (j,k) be a gray scale image quantized to an arbitrary number of gray levels, n, whose pixel values can be between zero and $2^n - 1$ or undefined, *. Only defined pixels may be elements of the gray scale image. An undefined pixel may come about by a variety of methods. It may be due to a sensor problem or the pixel may simply be out of the bounds of the image. For gray scale images the union operation is interpreted as the maximum of the two input images taken point by point. The intersection operation is interpreted as the minimum of the two input images taken point by point. The usual maximum and minimum definitions can be extended to include undefined pixel values by treating * as if it were negative infinity. The max of two defined pixels is the highest value pixel, the max of a defined and an undefined pixel is the defined pixel value, the max of two undefined pixels is *. The min of two defined pixels is the lowest pixel value, the min of a defined and an undefined pixel is *, and the min of two undefined pixels is *. The complement F^c (j,k) of a gray scale image F (j,k) is found by replacing every pixel value, i, with $2^n - 1 - i$. The translation, and reflection of a gray scale image are defined and denoted the same as they are for a binary image. For an NxN image matrix F (j,k) and an LxL, where L is odd, structuring element matrix H (j,k). Gray scale Minkowski addition is defined as:

$$G(j,k) = \max_{(r,c)\in H} [T_{r,c}\{F(j,k)\}]$$

and gray scale Minkowski subtraction as:

$$G(j,k) = \min_{(r,c)\in H} [T_{r,c}\{F(j,k)\}]$$

The alternate form of the dilation operation is defined as:

$$G(j,k) = \max [F(m,n) + H(j-m+S, k-n+S)]$$

The alternate form of the erosion operation is defined as:

$$G(j,k) = \min\ [F(j,k) + H(j-m+S, k-n+S)]$$

where the same limits as above for binary images hold. It may be of interest to compare the above equation with the defining equation for convolution. In the dilation and erosion equations the max or min operations are analogous to the summations in the convolution equation and the point by point additions are analogous to the point by point multiplications in the convolution equation. Similar to convolution, dilation can be thought of as the scanning and processing of F (j,k) by H (j,k) rotated by 180^0. The gray scale opening and closing operations are once again defined as:

$$O(F,H) = D(E(F,H), H) = [F(j,k) \ominus H\hat{}(j,k)] \oplus H(j,k)$$

$$C(F,H) = E(D(F,H\hat{}), H\hat{}) = [F(j,k) \oplus H\hat{}(j,k)] \ominus H(j,k)$$

where now gray scale erosions and dilations are performed. Because the max and min operations involved in erosion and dilation can be performed sequentially the erosion and dilation operations can decomposed into a series of iterative erosions or dilations to accomplish the same effect as a larger structuring element. Three iterations of a morphological image processing operation using a 3x3 structuring element accomplishes the same effect as one operation using a 7x7 structuring element.

IV. Materials and Apparatus

The library is developed on and for HP/Apollo workstations operating the Aegis™ operating system. The CAD system with which that the library is integrated is that of Mentor Graphics.

The Mentor Graphics development platform that will be supported by the proposed library consists of a schematic capture tool, Neted; a logic simulator, Quicksim; a circuit simulator, Accusim; a cell place-and-route tool, Cellgraph; and a layout editing tool, Chipgraph. The design flow allows the designer to enter in the schematic and simulate it logically to verify functionality, and then to simulate it electrically to verify its timing. For automatic placement and routing of the layout, Cellgraph is used. For improved compaction or higher performance parts, the designer may use Chipgraph to hand edit the layout to clean up any of the automated tool's inefficiencies.

The layout is verified to conform to the design rules by Cadence's Dracula II design rule checker (DRC), and verified to map to the schematic by its layout versus schematic (LVS) function. After the layout is done, the designer can extract the layout capacitances and re-simulate the circuit for greater accuracy. Towards this end, the library will fully support Cadence's Dracula II layout parameter extraction (LPE). Some of these functions have recently been added by Mentor Graphics in their Checkmate program, and the library will eventually support those tools as well.

V. Method of Procedure

The overall procedure of this thesis followed these steps:

1. Create an initial component library.
2. Create an initial cell library.
3. Generate a tool set whereby the library may be used with the schematic capture, simulators, layout editors, and layout verifications systems of Mentor Graphics and Cadence, as described above.
4. Design a Morphological Array Processor using the library.
    a. Perform the logic design of the MAP.
    b. Perform the Circuit Design of the MAP.
    c. Layout the MAP.
    d. Repeat the circuit simulations of critical paths with back annotated capacitance data extracted from the layout.

A. Creation of the Component Library

1. Component Choice

The first obvious step to generating a component library is the choice of what components to implement. In order for the library to be practical, the most commonly used components were implemented first. The initial philosophy was that the component library was to be the equivalent of a small scale integration (SSI) library. With very few exceptions, each component had only a single output and represented a single, unique function.

At the very beginning, the library consisted of a few pass transistor gates, inverters, and NAND and NOR gates. The remainder of the components (as well as all future components) were built using these primitives.

In the beginning, special emphasis was given to generate a full set of transmission gates. These included nMOS, pMOS, and CMOS transmission gate cells, as well as resistive nMOS and pMOS cells. These were given a very large range of possible drive size values. This was to insure that the components that were to be created later would have a ready set of primitive building blocks from which to be made. The resistive pass transistors were to be used for weak feedback circuits. These were later used in the latch circuits.

2. Component Specifications

Each component consists of a symbol and a schematic. The symbol is what the users will use in their schematics. Within the schematics some components have logical and physical case frames. This will allow logical simulation to be done more efficiently. For example, rather than logically simulate ten transistors for a 4-input AND gate, a single AND primitive is used, thus cutting down the number of elements Quicksim must handle. Quicksim supports such primitives as inverters, AND, OR, NAND, and NOR gates.

Some symbols of components (NANDs, NORs and INVs) have a "bub" option. This is merely a different symbol to represent a DeMorgan version of the same logical component.

The components represent both logical and physical elements. Each element has a "size" or a "sfx" (suffix) property to represent it's size and drive strength. The size property is represented by a letter in parentheses, while the sfx property is represented by a single letter. All size properties are initialized to "(X)", and all sfx properties are initialized to "X". The size properties will not work if they are not kept within parentheses, nor will the sfx properties work if they are kept within parentheses.

For logical simulation, no altering of the size or sfx properties is necessary. The default rise and fall delays for logical simulation is one time unit (ns) per gate. Therefore a 2-input AND would have a default delay of 2 due to the two gates within (a NAND gate followed by an inverter). Default logical delay times are produced by adding the properties "rise" and "fall" to the output pins of the symbols. The values of these properties may be explicit (i.e. an integer), or implicit. Examples of implicit values would be "(prise)" and "pfall", where "prise" and "pfall" are parameters in the expansion file (please see the section on Netlisting Scripts on page 20).

When the circuit is expanded for circuit simulation, a lookup table is used to convert the size properties into numerical values. For a list of the drive sizes and the effective widths that they represent, please see table 7.

All transistor lengths are minimum (2.0 μm). The look up numbers represent widths. If a component with the size property is made of both nMOS and pMOS devices, then the size represents the effective width of the nMOS devices, with the width of the pMOS devices a constant multiple of that. This constant is called "pscale" and is found in the expand file, with the default value of 2. For example, a "(B)" sized inverter would translate to an nMOS transistor of width 16μm and a pMOS transistor of width 32μm.

Another example is a 3 input "C" sized NAND gate (figure 4). The effective width of the nMOS transistors are 24μm, and the effective width of the pMOS transistors are 48μm. The pMOS transistors are in parallel, and therefore their widths are equal to the effective width (as there is only one transistor in a worst case path from Vcc to the output). The nMOS transistors are in a series of three, and must then be 3 times as

wide to have the same effective width, which comes to 72μm. It would be unwise to use large drive versions of NANDs and NORs with many inputs as they consume huge areas and are very slow.

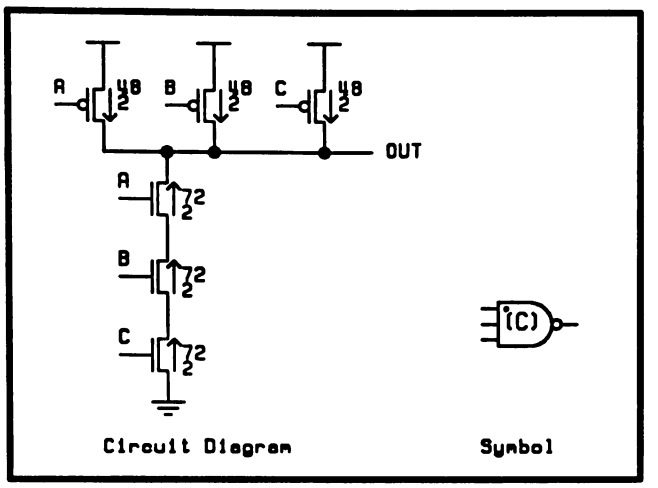

Figure 4 Three Input C Sized NAND

Some of the component symbols have a small circle next to one input (e.g. NANDs and NORs). This is to represent the fastest input. In the above example, this would be input A.

Support of devices with non-standard transistor sizes exists if the component has the size property. This is done by allowing the user to add width and length properties (nw, pw, nl, and pl) to these symbols. Not all the properties need to be added. The default lengths are 2μm, and the default widths are 8μm.

The sfx properties are handled differently. Instead of looking up the widths in a table at expand time, the circuits are sized within "case" frames on the component schematics themselves. This allows entirely different circuits to be represented by the same symbol and is useful for such things as large buffers. The only way to find out what drive strength a given sfx value of a given component has is to view the schematic of that component.

In general, the components with the size property are the more primitive, while the sfx components are the more complex. All component schematics, except for the primitive transistors, are made up of sized components, thus maintaining the standard size ratios even within the sfx propertied components.

An example of the use of logical, physical, and sfx frames is given in figure 5. Please note the rise and fall delay of 2 in the logical frame. The physical frame contains several sub-frames representing the supported sfx values. Further note that the frame for sfx value of "A" actually yields a drive size of "C".

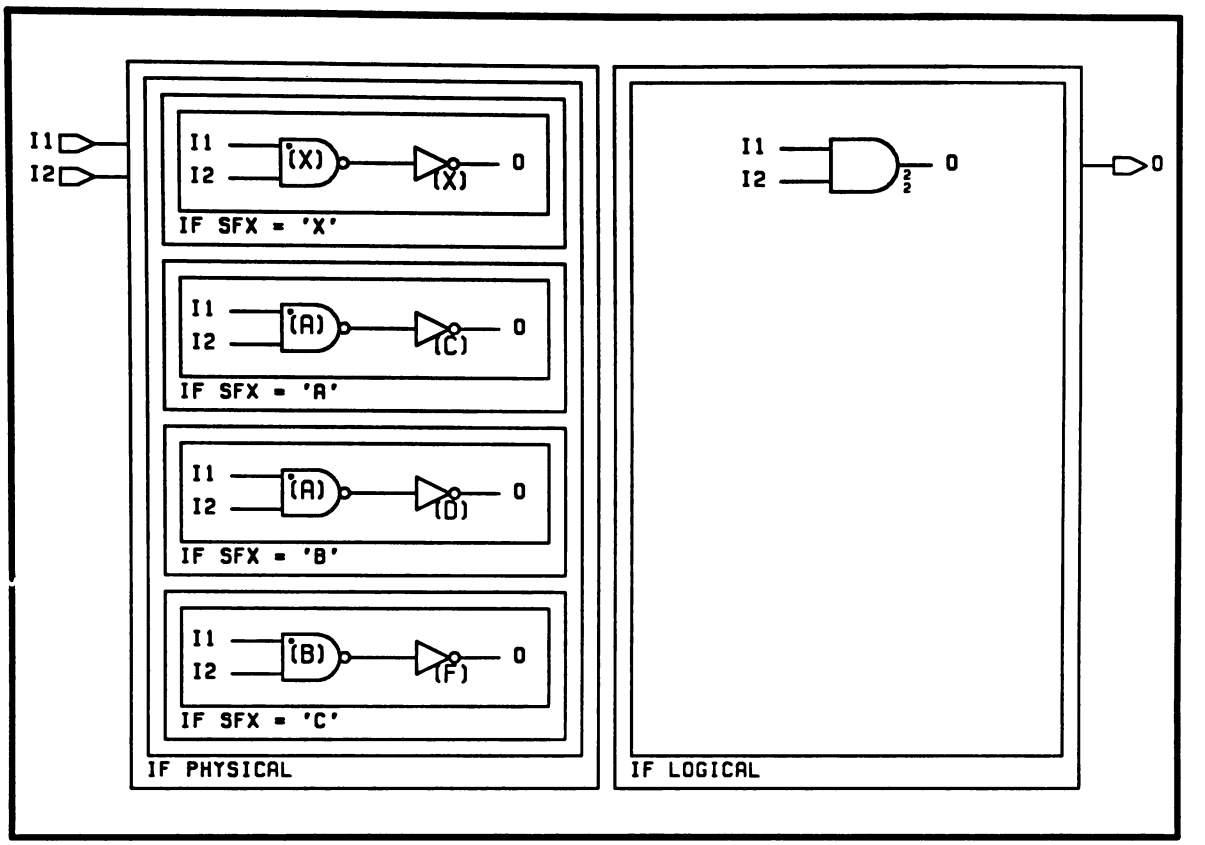

Figure 5 2-Input AND Gate Schematic

The actual physical simulation of the transistor level design can only be done by adding properties such that each transistor has several SPICE parameters. These allow accurate modeling (SPICE level 2), with leakage currents and junction capacitances. The parameters that are added to each sized nMOS transistor are:

Parameter	Description	Equation
NL	nMOS length	nl
NW	nMOS width	nw
AS	source area	nw * al
AD	drain area	nw * al
PS	source perimeter	nw + nw + al + al
PD	drain perimeter	nw + nw + al + al
NRS	resistance squares	al / nw
NRD	resistance squares	al / nw

TABLE 1 SPICEPAR Parameters

Where "nl" is the length of the transistor, "nw" is the width and "al" is the effective length of the active area. For a diagram demonstrating the derivation of these transistor parameters, please see figure 6. The default values for nl, nw and al are 2μm, 8μm, and 4μm respectively.

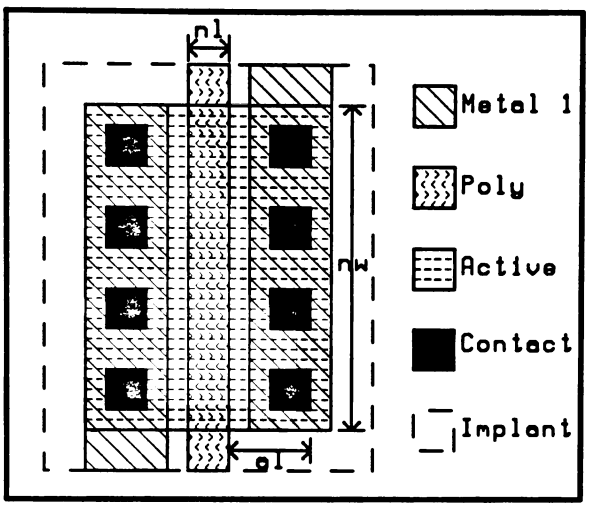

Figure 6 Derivation of Transistor Parameters

The equations are implemented within the properties attached to a component's symbol. For example, the inverter symbol has the property "nw" with the value "(size)", and the property "pw" with the value "(nw * pscale)". The parenthesis signify that a parameter is being passed down from the schematic where the component is placed.

The most primitive symbols, nmos4 and pmos4 (of which all else is made), contain the property "spicepar" which is used by Accusim for SPICE parameters. The spicepar of nmos4 has the value of:

"('TN L='&NL&'U W='&NW&'U AS='&(NW*AL)&'P AD ='&(NW*AL)&'P PS='&
(NW+NW+AL+AL)&'U PD='&(NW+NW+AL+AL)&'U NRS='&(AL/NW)&'NRD='&
(AL/NW))"

After expansion of a B sized nxfr, the SPICE deck will have an entry like the following:

MI$30/N 3 13 0 0 TN L=2U W=160U AS=640P AD=640P PS=328U PD=328U
+NRS=0.025 NRD=0.025

3. Neted Macros

Several macro routines and scripts were written into a startup file to be used by the designer to increase performance and simplify use of the schematic capture tool, Neted. The component menu was re-written to add the component library and these macros.

The first set of macros change the sheet size of the schematic and create a title block in the schematic's lower left corner. The available sheet sizes are 8.5"x11", 11"x17", and 22"x34". The title block includes information such as the schematic's name, its place in the hierarchy, the designers name, the revision number, and the date. The user is prompted for most of this information, but the revision number and date are automatically updated when the user checks and saves the schematic by typing the control-Y key (another added macro). This allows for automatic documentation control.

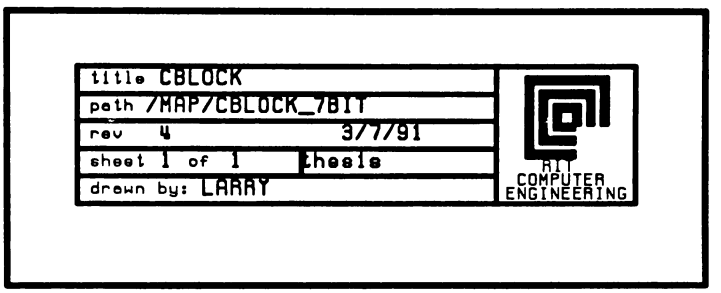

Figure 7 Sample Title Block

Other macros include automatic instance and net naming. These are for symbols of lower elements of the hierarchy. Every element is referred to internally by its instance number. The default numbers are "I$" followed by an integer (e.g. "I$21"). This is somewhat unreadable by the user. All instance numbers must be unique. By adding an "inst" property to the symbols generated automatically (see next section), every instantiation of the same component will have identical instance names. The "add inst prop" macro was written to take all the instance names of multiply instantiated components and append an incrementing integer (from 0) to their instance names. For example, if a schematic contained three "dff"s, their instance names would be "dff0", "dff1", and "dff2".

The user can automatically add a net name that is equal to a pin name. This can be done by placing the curser on the pin and typing control-P for individual pins, or can be done by the "add net name" macro to perform this operation on all pins of a symbol. These macros place the net name directly on top of the pin name, to minimize clutter of the schematic.

Many of the default function keys have been redefined to operate more efficiently or more user friendly. Many of the popup menu calls have been removed in order to allow faster operation on the slower nodes. The control-Q key has been redefined to prevent users from invalidly aborting an edit session and corrupting their databases.

4. Automatic Symbol Generation

To save the user the tedious task of symbol generation, as well as to standardize the hierarchial schematic symbols, an automatic symbol generator was written and named "gensym". What may have taken an experienced designer fifteen minutes to half an hour to draw by hand can now be done in under a minute. Gensym is a C program which reads schematics, using routines supplied by Mentor Graphics' sheet lister library.

Gensym recognizes the ports on a schematic and creates the equivalent pin on the symbol representing that schematic, distinguishing between input, output, and io pins. Gensym also adds the appropriate properties to the symbol. One such property is that of "inst", by which that symbol is labeled by the netlisters. The default label would be "/I$n" where n is some integer assigned by Neted, and is fairly unreadable. By adding the "inst" property, the netlist label becomes the name of the symbol (for example, "/half_adder"), which is more descriptive and thus helpful to the designer for debugging.

5. Netlisting Scripts

Netlisting is the act of extracting a formal description of the circuit from the schematic into a specific hardware description language (HDL) file. This file is then compiled into a form used by a specific software package. "Expand" is Mentor Graphic's general purpose netlister, and Extract then compiles the netlist. For different simulators, different parameters must be used by the expand program.

There are three netlisting operations supported by the library tool set. These consist of two expand scripts and one netlister that was written to create SILOS netlists (an industry standard HDL). The SILOS netlist is used by the Cadence's Dracula II package to perform ERC and LVS checking. It can also be used by Mentor Graphic's CPF utility, but that package is not available at the time of this writing.

The files lexpand and pexpand are scripts to run Expand in batch mode for logical and physical simulation, respectively. They contain the lookup tables for the size property, as well as several other parameters. The lookup sizes are not used for logical expansion, but are added anyway to prevent unnecessary warnings (see table 2).

Property	Value	Description
vplus	vcc	name of positive power node
vminus	vss	name of ground power node
physical	0	switch for logical or physical case frames
logical	1	switch for logical or physical case frames
prise	1	default rise time (logical simulation only)
pfall	1	default fall time (logical simulation only)
a	8	The lookup table for the size property
b	16	(effective transistor widths in microns)
c	24	
d	32	
e	40	
f	48	
g	64	
h	80	
i	160	
nsub	vss	the base of the nMOS transistors
psub	vcc	the base of the pMOS transistors
nw	8	the default nMOS width
pw	16	the default pMOS width
nl	2	the default nMOS length
pl	2	the default pMOS length
pscale	2	the default pMOS width / nMOS width ratio
pdrive	sss	the default logical drive strength

TABLE 2 Expand Parameters

Other primitive and property declarations are made. For more information on what these are, consult the Expand User's Manual and the Expand Reference Manual provided by Mentor Graphics.

The SILOS format netlister (also called the ECAD netlister because the products it supports, Dracula II and CPF, used to be owned by ECAD Inc.) is a rather efficient, hierarchical netlister. It creates macro files within each directory of each schematic of the design hierarchy called "ecad_mac". The final file it generates in the directory of the top level schematic is called "ecad.net". This is the final netlist file that is compiled for use by Dracula II, or is used directly by CPF.

For the library components, the netlister creates a different macro file for each size or sfx property (i.e. "ecad_mac.a", "ecad_mac.b", etc.). Each library component has macro files for all the supported sizes, as well as an unsized macro (which is simply called "ecad_mac"). For more concerning the netlister, see section 5.3.1 of appendix D.

B. Creation of the Cell Library

1. Cell Choice

It is unnecessary and impractical for the cell library to implement every size of every component. Several sizes of certain components are too large to be efficiently used. For example, if an F size 6 input NOR gate was desired, it would be better to use an A size 6 input NOR gate followed by a B sfxed BUF gate. The resultant circuit would have less input capacitance, less layout area, and operate at higher speeds.

Another reason not to implement certain sizes of some cells is that they would be no improvement over already existing cells. An example of this would be an A sized D flip-flop (dffa). The last stage of the layout of the dffc cell is a C sized inverter. The cell dffa would end with an A sized inverter instead. However, an A, B, and C sized inverter all require the same amount of layout area. Therefore a dffa cell would only have less drive strength than a dffc, but would have no advantage in area.

The choice of which cells to implement was based on which would be most likely to be used by the simple projects that were being assigned at the time. A strong precedence was given to cells required for the MAP and other thesis projects.

In the Spring of 1991 the Advanced VLSI course taught by Professor George A. Brown (EECC 631) was assigned to layout the pass transistors, inverters and buffers. This gave them the needed experience to layout the D flip-flops. These cells were then re-edited and cleaned up by the author of this thesis and other graduate students, Jeff Correll, Shishir Ghate, and Chris Insalaco, whose MS theses involved the maintenance of the library. This was a very successful assignment for everyone involved. The students had the opportunity to do original work that would be used in the future by them and by other classes, Professor Brown had a lecture taught and projects assigned and graded by the author of this thesis, who, in turn had a quantity of tedious work performed by the students.

2. Cell Specifications

The cell library was designed with the philosophies of low power and static design in mind. The cells would have to consume a minimum of area and fulfill all the requirements of the automatic place-and-router tool, Cellgraph.

The inputs of the cells have been drawn vertically on the poly and metal 2 layers, while the outputs have all been drawn vertically on the metal 2 layer exclusively. The top and bottom of each cell has horizontal metal 1 lines for Vcc and Vss, respectively. The internal circuitry of each cell was drawn to contain a minimum of metal 2. By using this scheme, the router is able to draw horizontal channels with metal 1 and vertical channels with metal 2. Many vertical channels are able to be drawn directly over circuitry, thus eliminating a need for inefficient feedthrough cells that other layout strategies might require.

The following are a list of rules that were followed in order to make each cell compatible with the router and the rest of the library. Most of these rules are to make the edges of one cell compatible with the edges of any other cell that could be placed next to it.

1. All cells have an 8μm tall VCC line on top and an 8μm tall VSS line on the bottom. The distance from the top of the VCC line to the bottom of the VSS line is 94μm.

2. Any metal2 is at least 2μm from the left and right edges of the cell.

3. Any metal1 (except for the VCC and VSS lines, which must go to the left and right edges) is at least 2μm from the left and right edges of the cell.

4. Any poly is at least 1μm from the left and right edges of the cell.

5. Any active_area is at least 2μm from the left and right edges of the cell.

6. The n_implant and p_implant layers go up to the left and right edges of the cell.

7. The N-well is drawn from 47μm from the top of the VCC line up to at least the top of the VCC line (it may go over), and is drawn to at least the left and right edges of the cell (it may go over by a maximum of 3μm).

8. All cells are surrounded by a perimeter of layer cell_outline.e which encompasses everything contained in the cell excluding the N-well. All cells also have a point of cell_txt.e in the center with the property cell_name added.

9. The origin of every cell is the lower left corner of the VSS line. This enables more efficient automatic placement.

10. All polygons are drawn on a 1μm grid. The router may, however, route to a 0.5μm grid.

11. All input and outputs have non-overlapping, straight line paths out the top and/or bottom of the cell.

12. All inputs and outputs are drawn as poly.e or metal2.e ports. Some input or outputs may have multiple ports to allow for more efficient routing. These ports have the property "selgroup" added to them with incrementing integers as values.

13. All cells have blockage layers. These are metal1.e, metal2.e and poly.e perimeters that are drawn around all objects of their corresponding internal layers that the router may not route over.

14. All transistors are maximally contacted, with the contacts being evenly distributed. This minimizes the internal resistance of the source/drain areas.

Please see figure 8 for the layout of two sfx values for the 2-input AND cells.

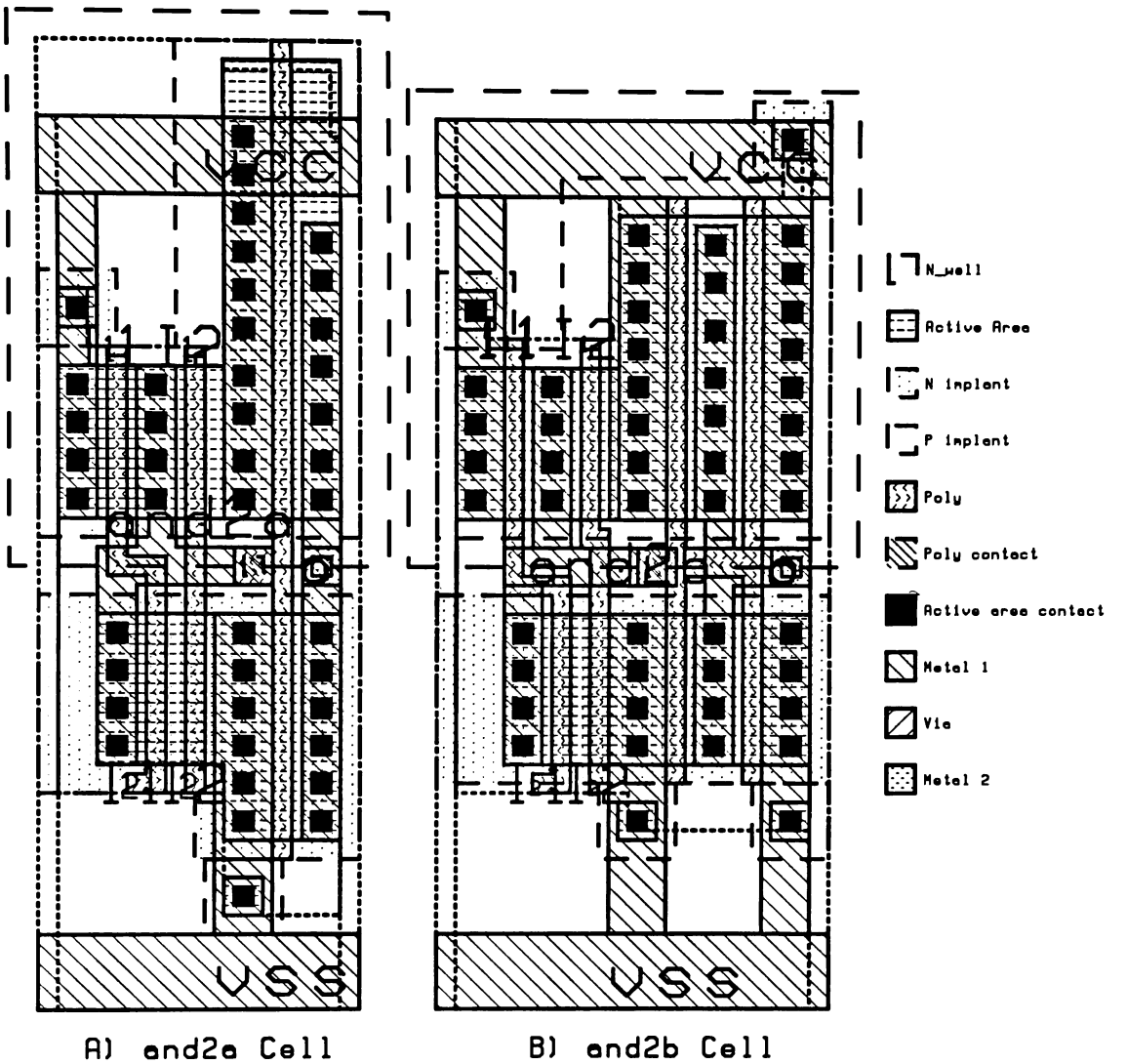

Figure 8 2-Input AND Layouts

3. Chipgraph Macros

Several macro routines and scripts were written into another startup file to be used by the designer to increase performance and simplify use of the layout editing tool, Chipgraph. Sub-menus were added to the main menu in order to add these macros.

One of these macros is the CMOS cell library. This sub-menu contains a list of each cell in the library and has been designed to perform similarly to the component library menu within Neted. The user clicks on a cell name, and the corresponding cell becomes the current active cell. The user then has to simply place the curser in the desired location of the layout and type the F5 key to instantiate that cell.

Another sub-menu that has been added is for the support of the on line design rule checker, REMEDI. The REMEDI checker is not as complete as the Dracula II DRC, and is therefore used for small cells and is intended to be run to find the simpler errors that the user may have. The advantages of REMEDI over Dracula II are that it executes much faster on smaller circuits and can run on any node, and may therefore be effectively used to minimize the time the user must spend on design rule checking. For more information on REMEDI, please see appendix E, The REMEDI User's Manual, written by Jeff Correll.

Other, simpler macros were added for operations such as the plotting of designs, and the viewing of selected layers (all, none, error only).

4. Dracula II Layout Verification Tools

Cadence's layout verification tool, Dracula II contains four major functions, design rule checking (DRC), electrical rule checking (ERC), layout versus schematic comparison (LVS), and layout parameter extraction (LPE). A drawback of Dracula II is that it is currently only licensed on two of the 15 nodes that are available, and that each license may only be used by one user at a time. If more than one user attempts to operate a copy of the software, the data becomes corrupted. Therefore a semaphore has been added to all of the calling scripts that have been written to prevent this from occurring. For information on the use of these tools, please see appendix C, the CMOS Standard Cell Library User's Manual.

The MOSIS scalable CMOS N-well (SCN) design rules were implemented. The initial feature size was chosen to be 2.0μm to match the cells. In the future, should another supported feature size be used, the design rule file will have to edited to reflect the new scale.

MOSIS supports two contact rule sets, a simple one and a dense one. The dense contact rule set was chosen so that the layout might be more compact. The disadvantage of this decision was that some of the rules were difficult to implement.

A script was written and placed in a public directory (/user/pub) to run the DRC. This script first translates the Chipgraph database into GDS2 format and then calls the Dracula II DRC. The results are then translated back from GDS2 format into error cells that the user may instantiate back into the layout. Similar scripts have been written for ERC, LVS, and LPE.

The ERC will flag shorts between labeled nodes (e.g. VCC and VSS), or open circuits between nodes with the same name property. This will avoid a lot of time spent running LVS to find these same errors, as LVS gets very confused by them. ERC also checks for nodes with no paths to VCC nor VSS, transistors with only one source/drain, or other malformed transistors.

The ERC creates a results file that contains a list of the bad transistors or gates, as well as error cells that the user may instantiate in the layout similarly to DRC.

Before running ERC, LVS, or LPE, the user must add "gdsii_text" properties to the input, output, and power nodes in the layout so that the tools will have a starting point with which to extract the netlist from the layout. These three tools also require that the schematics be netlisted and the netlist then compiled.

To provide that function, the ECAD netlister has been written (please see the previous section on Netlisting Scripts, starting on page 20). The results of the netlister are compiled by a routine supplied by Cadence called LOGLVS, which is written into another script in the public directory. The compiler also links in a SPICE format library of primitive macros (i.e. NANDSs, NORs, NXFRs, CXFRs, etc.) so that the error reports will recognize these gates and not simply report on the lowest transistor level.

The LVS will compare a netlist it extracted from the layout with the user supplied, compiled netlist and generate a discrepancy file containing any unmatched nodes and/or devices. It will also check that all of the devices are of the proper dimensions, within a given tolerance (currently set at 2%). The LVS will also generate error cells that the user can instantiate in the layout.

The LPE will create a SPICE deck based on the extracted layout, and will add all the parasitic capacitances caused by the layout lines. Only those capacitances above a defined threshold are added. Currently, this tool is impractical for large designs because it generates all the metal2 to metal1, metal2 to poly, metal2 to substrate, metal1 to poly, metal1 to substrate, and poly to substrate capacitors as it operates, and only when it is completed with this, does it "smash" the capacitors into the more useful node (sum) capacitors.

The problem with this algorithm is that for medium to large designs (over 2500 transistors), it lacks sufficient space in its internal memory to operate. It is the work of Shishir Ghate's thesis to develop a solution to this problem, as well as a practical method to back-annotate the resulting capacitances.

C. The Architecture of the MAP

To successfully perform a morphological operation, an image and a window composed of patterning elements are needed. The window is slid across the image such that the middle element of the window crosses over every pixel in the image. The image pixel that is under the center window element is the pixel that is being operated on, which will be referred to as the target pixel. Operations are performed using all of the window values and the image values that lie directly beneath the window values, with the resulting value taking the place of the original target pixel value.

When the construction of a Morphological Image Processor prototype was proposed, it was decided that the image would be 512 × 512 pixels using an 8-bit gray scale pixel representation, with an extra bit used to represent negative infinity (shown as * in the Morphological Theory section and as $-\infty$ in all future references) and negative pixel values. The meaning of $-\infty$ was discussed in the Morphological Theory section and since it was possible to use the extra bit to also represent negative pixel values, they were included in case a use is ever found for them. Table 3 shows the binary pixel values and their corresponding decimal values. Negative values use the two's complement representation, with $-\infty$ using the value that would normally represent -512. Since most image capture and processing systems can only deal with 8-bit gray scale images and have no concept of $-\infty$ and negative pixel values, the 9-bit gray scale system will be referred to as an extended 8-bit gray scale system. The window was chosen to be a 7 × 7 array of patterning elements, also using the extended 8-bit gray scale pixel representation.

binary	decimal
0 0000 0000	0
0 0000 0001	1
0 0000 0010	2
⋮	⋮
0 1111 1101	509
0 1111 1110	510
0 1111 1111	511
1 0000 0000	$-\infty$
1 0000 0001	-511
1 0000 0010	-510
⋮	⋮
1 1111 1101	-3
1 1111 1110	-2
1 1111 1111	-1

TABLE 3 Binary Pixel Values And Their Decimal Equivalents

While choosing an architecture for the Morphological Image Processor prototype, there were a few goals that were desirable to achieve. One of these goals was to allow future revisions of the processor to operate at a real-time rate (60 images per second) using the same 512 × 512 pixel extended 8-bit gray scale images and a 7 × 7 window. Implementing the design using a VLSI circuit is currently the best way to achieve the real-time rate, so a regular structure is desirable to facilitate a VLSI layout. Another goal was a simple control section to facilitate expansion to larger window sizes and larger images. Also, the processor should be designed such that it is easy to pipeline with identical processors and allow the inputs and outputs to be connected to real-time sources and destinations, possibly with some buffering to make the system compatible with interleaved scan line systems.

For the Morphological Image Processor prototype, a 512 × 512 image and a 7 × 7 window was used, so all examples and explanations will use these sizes, unless otherwise noted. All of the concepts discussed in this section will work with any size image and window as long as appropriate adjustments are made. All index values will start at 0 and end at the one less than the maximum value, for example, the top left of a 512 × 512 image will be referred to as $X_{0,0}$ and the bottom right of that image will be referred to as $X_{511,511}$, with the first index value referring to the row and the second one being the column.

As previously discussed in the Morphological Theory section, the morphological image processing operation is similar to convolution, with additions replacing the multiplications, and comparisons replacing the final additions. With a 512 × 512 image operated on by a 7 × 7 window, the values of every pixel in the image and its 48 neighbors are added to their corresponding window values (with the window centered over the pixel being operated on), and then either the largest or smallest of those values, depending on the operation, is the resulting pixel value. The general mathematical equation follows:

$$Y_{i,j} = compare\ (X_{i+3,j+3} + W_{6,6},\ X_{i+3,j+2} + W_{6,5}\ \cdots\ X_{i-3,j-3} + W_{0,0})$$

where the operands of the compare are 49 separate additions computed in parallel and the *compare* is either the largest or smallest of the 49 results, depending on the desired morphological operation. In the equation, X is the input image, Y is the output image, W is the window matrix, i is the row index of the pixel being operated on (the target pixel), and j is the column index. A better indication of the 49 separate additions that are performed can be seen in figure 9, where the left matrix denotes a portion of the image at any given time that will be added to the window matrix, which is shown on the right side of the figure, with $X_{i,j}$ being the target pixel. The matrices are rotated 180° with respect to conventional matrix representation to be consistent with future references. At the edges of an image, some of the values in the image matrix will not be valid (for example, when $X_{0,0}$ is the target pixel, all pixels to the bottom and to the right in the matrix will have negative indices making them invalid). Depending on the specific architecture used, there are different ways to handle that situation and two of those ways will be discussed.

$X_{i+3,j+3}$	$X_{i+3,j+2}$	$X_{i+3,j+1}$	$X_{i+3,j}$	$X_{i+3,j-1}$	$X_{i+3,j-2}$	$X_{i+3,j-3}$
$X_{i+2,j+3}$	$X_{i+2,j+2}$	$X_{i+2,j+1}$	$X_{i+2,j}$	$X_{i+2,j-1}$	$X_{i+2,j-2}$	$X_{i+2,j-3}$
$X_{i+1,j+3}$	$X_{i+1,j+2}$	$X_{i+1,j+1}$	$X_{i+1,j}$	$X_{i+1,j-1}$	$X_{i+1,j-2}$	$X_{i+1,j-3}$
$X_{i,j+3}$	$X_{i,j+2}$	$X_{i,j+1}$	$X_{i,j}$	$X_{i,j-1}$	$X_{i,j-2}$	$X_{i,j-3}$
$X_{i-1,j+3}$	$X_{i-1,j+2}$	$X_{i-1,j+1}$	$X_{i-1,j}$	$X_{i-1,j-1}$	$X_{i-1,j-2}$	$X_{i-1,j-3}$
$X_{i-2,j+3}$	$X_{i-2,j+2}$	$X_{i-2,j+1}$	$X_{i-2,j}$	$X_{i-2,j-1}$	$X_{i-2,j-2}$	$X_{i-2,j-3}$
$X_{i-3,j+3}$	$X_{i-3,j+2}$	$X_{i-3,j+1}$	$X_{i-3,j}$	$X_{i-3,j-1}$	$X_{i-3,j-2}$	$X_{i-3,j-3}$

W6,6	W6,5	W6,4	W6,3	W6,2	W6,1	W6,0
W5,6	W5,5	W5,4	W5,3	W5,2	W5,1	W5,0
W4,6	W4,5	W4,4	W4,3	W4,2	W4,1	W4,0
W3,6	W3,5	W3,4	W3,3	W3,2	W3,1	W3,0
W2,6	W2,5	W2,4	W2,3	W2,2	W2,1	W2,0
W1,6	W1,5	W1,4	W1,3	W1,2	W1,1	W1,0
W0,6	W0,5	W0,4	W0,3	W0,2	W0,1	W0,0

Figure 9 Image and Window Matrices

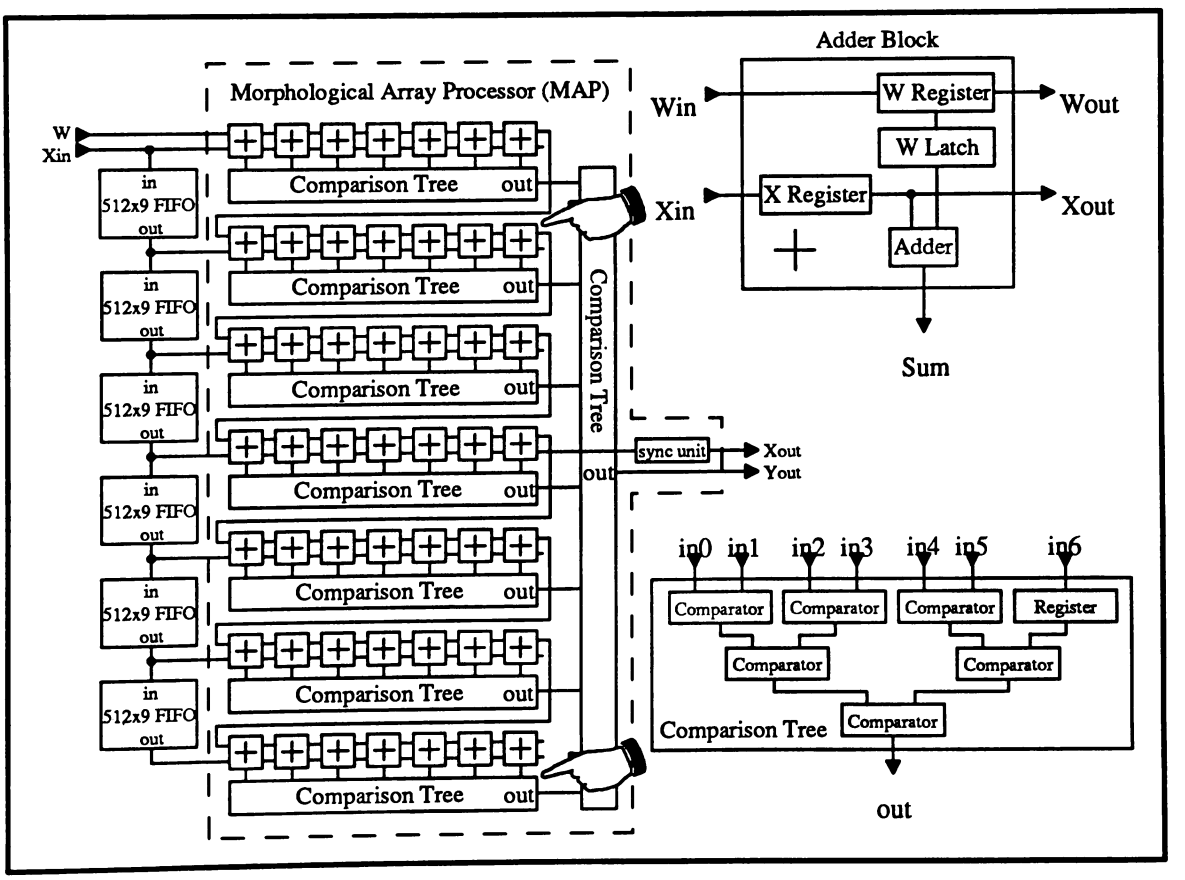

Figure 10 Morphological Array Processor Architecture

The architecture is suitable for pipelining, with the input format being the same as the output format. Figure 10 shows the general block diagram of the architecture, which includes the MAP (Morphological Array Processor) and six external 512 × 9 bit FIFOs. Since this figure is meant to illustrate data flow, control signals have been omitted. *X* refers to image values, *Y* refers to the resulting image, and W refers to window values. Both the window array and the adder array are rotated 180° with respect to the normal visual orientation of an array for imaging operations, with the window value $W_{0,0}$ being in the lower right corner, $W_{0,6}$ in the lower left corner, $W_{6,0}$ in the upper right corner, and $W_{6,6}$ in the upper left corner, as shown on the left side of figure 9. This orientation is used so that the data can be shown going into the top left of the array and coming out of the bottom right. The FIFOs are external to the MAP, as that configuration will allow any image size to be used with the same MAP, as long as the right size FIFOs are used and the control circuitry is appropriately modified.

Before processing begins, the window elements are serially loaded into the 49 W registers contained in the adder blocks. The pixels of the first row of the 512 × 512 image are sent into the MAP from left to right, at a rate of one pixel per clock cycle, immediately followed by the next row, also being sent in from left to right, continuing until all the rows have been sent into the MAP. The pixels are also simultaneously being sent into the 512 stage FIFOs, which are being used as line delays. Each of the FIFOs output image data that is exactly one image row above the output of the previous FIFO. For example, when FIFO #6 is sending the first row of image data into the last row of adder blocks, FIFO #5 is sending the second row of image data into the second to last row of adder blocks, FIFO #4 is sending the third row of image data into the fifth row of adder blocks, and so on. This will cause a 7 × 7 array of image data to be in the adder array at any given time, which will allow 49 simultaneous additions with the window patterning elements.

At any given time during processing, the pixel that is being operated on (the target pixel) is in the middle of the 7 × 7 array of adder blocks. All pixel values in the array are being added to the window values residing in the same adder block with the exception of pixels that are blanked. Blanking occurs when the edges of the image are being operated on, and unwanted pixels are in the 7 × 7 array, such as pixels from the opposite edge, pixels from the previous image, pixels from the next image, or unknown values when no image is directly preceding or following the image being processed. When a row and/or column is blanked, that row and/or column is not used for calculating the new target pixel value. For example, row blanking invalidates pixels from the right edge of the image when the left edge is being processed, and column blanking invalidates pixels from a previous image (or no image at all) when the first few rows of an image are being processed.

Figures 12 through 17 show how the blanking is implemented, with the figures showing 5 × 5 arrays and the image values contained in the corresponding adder blocks. 5 × 5 arrays have been used in place of 7 × 7 arrays for space considerations. The first and last rows and the first and last columns of a 7 × 7 array have been removed to create a 5 × 5 array, and the difference in the blanking procedure will be discussed

later. Each figure shows six consecutive cycles which surround a critical moment in the blanking process. Shaded rows and/or columns indicate which rows and/or columns are being blanked and the values in bold face indicates the target pixels. Any pixel value with a *prev* or *next* indicates a pixel from the previous or next image, respectively. Whenever a row and/or column needs to be blanked, the whole row and/or column must get blanked, as all of the pixels in that row and/or column are not valid for operating on the target pixel. In the example figures, each image is followed by another image without any delay, so it can be seen that in a real-time application no delay between images is needed. If desired, any delay greater than zero cycles can also be used, as any value that is not in the current image is always blanked.

To implement the blanking, control bits are needed to instruct each row and each column whether to blank or not. When a row and/or a column is blanked, the output of every adder in that row and/or column is set to $-\infty$. If a morphological erosion (minimum) is being performed, the $-\infty$ will propagate to the output, which is the desired effect for erosion. For a dilation (maximum), the $-\infty$ will have no effect on the output as $-\infty$ is the smallest defined value, which is also the desired effect. For a 7×7 window, no more than three rows and three columns can be blanked at any time, and the center row and center column of the adder array can never be blanked because the middle element of the array contains the target pixel. Figure 11 shows the bit numbers that will be used for both 5×5 and 7×7 windows to refer to the rows and/or columns that are to be blanked. Rows/columns 0 and 5 (the outer rows and columns) have been removed from the 7×7 array to make the 5×5 array. The elements that are missing from a 5×5 array that would make up a 7×7 can be determined by referencing the array on the left side of figure 9.

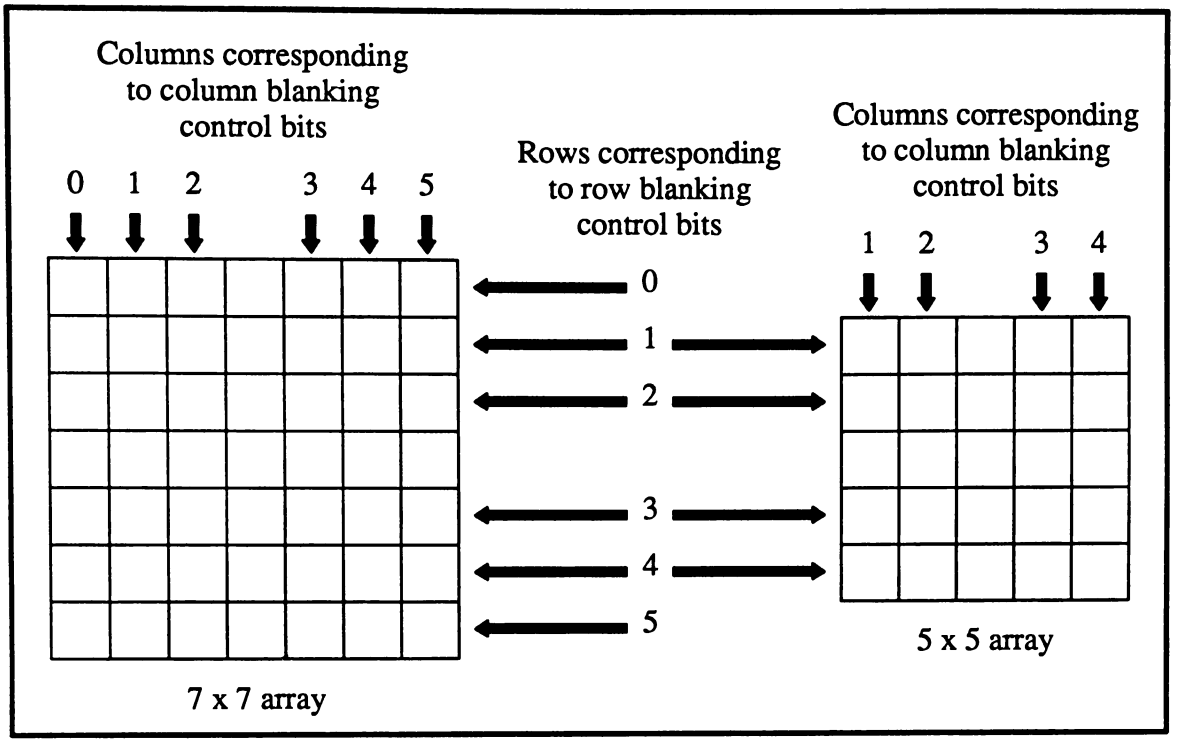

Figure 11 Blanking Bits

row or column index	7 x 7 blanking control bits 0 1 2 3 4 5	5 x 5 blanking control bits 1 2 3 4
0	0 0 0 1 1 1	0 0 1 1
1	0 0 0 0 1 1	0 0 0 1
2	0 0 0 0 0 1	0 0 0 0
3	0 0 0 0 0 0	0 0 0 0
⋮	⋮	⋮
508	0 0 0 0 0 0	0 0 0 0
509	1 0 0 0 0 0	0 0 0 0
510	1 1 0 0 0 0	1 0 0 0
511	1 1 1 0 0 0	1 1 0 0

TABLE 4 Blanking Control Values For All Index Values

Table 4 shows what the blanking control bits will be for all row and column index values of the target pixel. For a 7×7 window operating on a 512×512 image, no row blanking occurs between target pixel row index values of 3 and 508, as the rows in the adder array are far enough into the image to contain only valid pixels. Likewise, no

column blanking occurs between target pixel column index values of 3 and 508, as the pixels from the opposite edge (when near either the left or right edge) are no longer in the adder array. As can be seen in the table, a 7×7 window needs one more blanking control bit for each edge than a 5×5 window does (bit 0 on the top and left edges and bit 5 on the bottom and right edges). Each one of those four extra control bits (two for row blanking and two for column blanking) will be blanked one more cycle than its neighboring bit. For example, when the target pixel is $X_{0,0}$, $X_{0,1}$, or $X_{0,2}$, column blanking control bit 5 will specify that its corresponding column should be blanked.

target pixel (row,column)	row blanking bits 0 1 2 3 4 5	column blanking bits 0 1 2 3 4 5
0,0	0 0 0 1 1 1	0 0 0 1 1 1
0,1	0 0 0 1 1 1	0 0 0 0 1 1
0,2	0 0 0 1 1 1	0 0 0 0 0 1
0,3	0 0 0 1 1 1	0 0 0 0 0 0
⋮	⋮	⋮
0,508	0 0 0 1 1 1	0 0 0 0 0 0
0,509	0 0 0 1 1 1	1 0 0 0 0 0
0,510	0 0 0 1 1 1	1 1 0 0 0 0
0,511	0 0 0 1 1 1	1 1 1 0 0 0
1,0	0 0 0 0 1 1	0 0 0 1 1 1
1,1	0 0 0 0 1 1	0 0 0 0 1 1
1,2	0 0 0 0 1 1	0 0 0 0 0 1
1,3	0 0 0 0 1 1	0 0 0 0 0 0
⋮	⋮	⋮
1,508	0 0 0 0 1 1	0 0 0 0 0 0
1,509	0 0 0 0 1 1	1 0 0 0 0 0
1,510	0 0 0 0 1 1	1 1 0 0 0 0
1,511	0 0 0 0 1 1	1 1 1 0 0 0
2,0	0 0 0 0 0 1	0 0 0 1 1 1
2,1	0 0 0 0 0 1	0 0 0 0 1 1
2,2	0 0 0 0 0 1	0 0 0 0 0 1
2,3	0 0 0 0 0 1	0 0 0 0 0 0
⋮	⋮	⋮
2,508	0 0 0 0 0 1	0 0 0 0 0 0
2,509	0 0 0 0 0 1	1 0 0 0 0 0
2,510	0 0 0 0 0 1	1 1 0 0 0 0
2,511	0 0 0 0 0 1	1 1 1 0 0 0
3,0	0 0 0 0 0 0	0 0 0 1 1 1
3,1	0 0 0 0 0 0	0 0 0 0 1 1
3,2	0 0 0 0 0 0	0 0 0 0 0 1
3,3	0 0 0 0 0 0	0 0 0 0 0 0

TABLE 5 Blanking Control Values

target pixel (row,column)	row blanking bits 0 1 2 3 4 5	column blanking bits 0 1 2 3 4 5
508,508	0 0 0 0 0 0	0 0 0 0 0 0
508,509	0 0 0 0 0 0	1 0 0 0 0 0
508,510	0 0 0 0 0 0	1 1 0 0 0 0
508,511	0 0 0 0 0 0	1 1 1 0 0 0
509,0	1 0 0 0 0 0	0 0 0 1 1 1
509,1	1 0 0 0 0 0	0 0 0 0 1 1
509,2	1 0 0 0 0 0	0 0 0 0 0 1
509,3	1 0 0 0 0 0	0 0 0 0 0 0
⋮	⋮	⋮
509,508	1 0 0 0 0 0	0 0 0 0 0 0
509,509	1 0 0 0 0 0	1 0 0 0 0 0
509,510	1 0 0 0 0 0	1 1 0 0 0 0
509,511	1 0 0 0 0 0	1 1 1 0 0 0
510,0	1 1 0 0 0 0	0 0 0 1 1 1
510,1	1 1 0 0 0 0	0 0 0 0 1 1
510,2	1 1 0 0 0 0	0 0 0 0 0 1
510,3	1 1 0 0 0 0	0 0 0 0 0 0
⋮	⋮	⋮
510,508	1 1 0 0 0 0	0 0 0 0 0 0
510,509	1 1 0 0 0 0	1 0 0 0 0 0
510,510	1 1 0 0 0 0	1 1 0 0 0 0
510,511	1 1 0 0 0 0	1 1 1 0 0 0
511,0	1 1 1 0 0 0	0 0 0 1 1 1
511,1	1 1 1 0 0 0	0 0 0 0 1 1
511,2	1 1 1 0 0 0	0 0 0 0 0 1
511,3	1 1 1 0 0 0	0 0 0 0 0 0
⋮	⋮	⋮
511,508	1 1 1 0 0 0	0 0 0 0 0 0
511,509	1 1 1 0 0 0	1 0 0 0 0 0
511,510	1 1 1 0 0 0	1 1 0 0 0 0
511,511	1 1 1 0 0 0	1 1 1 0 0 0

Blanking Control Values (cont'd)

Table 5 shows more complete row and column blanking control bit values for a 7 × 7 window, with most of the critical areas of blanking directly corresponding to the 5 × 5 examples in figures 12 through 17. In the gaps between the left side and the right side of the image, no column blanking takes place and row blanking only occurs at the upper and lower edges of the image. In the gap between the upper and lower edges of the

image, no row blanking takes place and column blanking continues as usual at the sides of the image.

1,511	1,510	1,509	1,508	1,507
0,511	0,510	0,509	0,508	0,507
prev 511,511	prev 511,510	**prev 511,509**	prev 511,508	prev 511,507
prev 510,511	prev 510,510	prev 510,509	prev 510,508	prev 510,507
prev 509,511	prev 509,510	prev 509,509	prev 509,508	prev 509,507

2,0	1,511	1,510	1,509	1,508
1,0	0,511	0,510	0,509	0,508
0,0	prev 511,511	**prev 511,510**	prev 511,509	prev 511,508
prev 511,0	prev 510,511	prev 510,510	prev 510,509	prev 510,508
prev 510,0	prev 509,511	prev 509,510	prev 509,509	prev 509,508

2,1	2,0	1,511	1,510	1,509
1,1	1,0	0,511	0,510	0,509
0,1	0,0	**prev 511,511**	prev 511,510	prev 511,509
prev 511,1	prev 511,0	prev 510,511	prev 510,510	prev 510,509
prev 510,1	prev 510,0	prev 509,511	prev 509,510	prev 509,509

2,2	2,1	2,0	1,511	1,510
1,2	1,1	1,0	0,511	0,510
0,2	0,1	**0,0**	prev 511,511	prev 511,510
prev 511,2	prev 511,1	prev 511,0	prev 510,511	prev 510,510
prev 510,2	prev 510,1	prev 510,0	prev 509,511	prev 509,510

2,3	2,2	2,1	2,0	1,511
1,3	1,2	1,1	1,0	0,511
0,3	0,2	**0,1**	0,0	prev 511,511
prev 511,3	prev 511,2	prev 511,1	prev 511,0	prev 510,511
prev 510,3	prev 510,2	prev 510,1	prev 510,0	prev 509,511

2,4	2,3	2,2	2,1	2,0
1,4	1,3	1,2	1,1	1,0
0,4	0,3	**0,2**	0,1	0,0
prev 511,4	prev 511,3	prev 511,2	prev 511,1	prev 511,0
prev 510,4	prev 510,3	prev 510,2	prev 510,1	prev 510,0

Figure 12 Blanking part 1

2,511	2,510	2,509	2,508	2,507
1,511	1,510	1,509	1,508	1,507
0,511	0,510	**0,509**	0,508	0,507
prev 511,511	prev 511,510	prev 511,509	prev 511,508	prev 511,507
prev 510,511	prev 510,510	prev 510,509	prev 510,508	prev 510,507

3,0	2,511	2,510	2,509	2,508
2,0	1,511	1,510	1,509	1,508
1,0	0,511	**0,510**	0,509	0,508
0,0	prev 511,511	prev 511,510	prev 511,509	prev 511,508
prev 511,0	prev 510,511	prev 510,510	prev 510,509	prev 510,508

3,1	3,0	2,511	2,510	2,509
2,1	2,0	1,511	1,510	1,509
1,1	1,0	**0,511**	0,510	0,509
0,1	0,0	prev 511,511	prev 511,510	prev 511,509
prev 511,1	prev 511,0	prev 510,511	prev 510,510	prev 510,509

3,2	3,1	3,0	2,511	2,510
2,2	2,1	2,0	1,511	1,510
1,2	1,1	**1,0**	0,511	0,510
0,2	0,1	0,0	prev 511,511	prev 511,510
prev 511,2	prev 511,1	prev 511,0	prev 510,511	prev 510,510

3,3	3,2	3,1	3,0	2,511
2,3	2,2	2,1	2,0	1,511
1,3	1,2	**1,1**	1,0	0,511
0,3	0,2	0,1	0,0	prev 511,511
prev 511,3	prev 511,2	prev 511,1	prev 511,0	prev 510,511

3,4	3,3	3,2	3,1	3,0
2,4	2,3	2,2	2,1	2,0
1,4	1,3	**1,2**	1,1	1,0
0,4	0,3	0,2	0,1	0,0
prev 511,4	prev 511,3	prev 511,2	prev 511,1	prev 511,0

Figure 13 Blanking part 2

3,511	3,510	3,509	3,508	3,507
2,511	2,510	2,509	2,508	2,507
1,511	1,510	**1,509**	1,508	1,507
0,511	0,510	0,509	0,508	0,507
prev 511,511	prev 511,510	prev 511,509	prev 511,508	prev 511,507

4,0	3,511	3,510	3,509	3,508
3,0	2,511	2,510	2,509	2,508
2,0	1,511	**1,510**	1,509	1,508
1,0	0,511	0,510	0,509	0,508
0,0	prev 511,511	prev 511,510	prev 511,509	prev 511,508

4,1	4,0	3,511	3,510	3,509
3,1	3,0	2,511	2,510	2,509
2,1	2,0	**1,511**	1,510	1,509
1,1	1,0	0,511	0,510	0,509
0,1	0,0	prev 511,511	prev 511,510	prev 511,509

4,2	4,1	4,0	3,511	3,510
3,2	3,1	3,0	2,511	2,510
2,2	2,1	**2,0**	1,511	1,510
1,2	1,1	1,0	0,511	0,510
0,2	0,1	0,0	prev 511,511	prev 511,510

4,3	4,2	4,1	4,0	3,511
3,3	3,2	3,1	3,0	2,511
2,3	2,2	**2,1**	2,0	1,511
1,3	1,2	1,1	1,0	0,511
0,3	0,2	0,1	0,0	prev 511,511

4,4	4,3	4,2	4,1	4,0
3,4	3,3	3,2	3,1	3,0
2,4	2,3	**2,2**	2,1	2,0
1,4	1,3	1,2	1,1	1,0
0,4	0,3	0,2	0,1	0,0

Figure 14 Blanking part 3

511,511	511,510	511,509	511,508	511,507
510,511	510,510	510,509	510,508	510,507
509,511	509,510	**509,509**	509,508	509,507
508,511	508,510	508,509	508,508	508,507
507,511	507,510	507,509	507,508	507,507

next 0,0	511,511	511,510	511,509	511,508
511,0	510,511	510,510	510,509	510,508
510,0	509,511	**509,510**	509,509	509,508
509,0	508,511	508,510	508,509	508,508
508,0	507,511	507,510	507,509	507,508

next 0,1	next 0,0	511,511	511,510	511,509
511,1	511,0	510,511	510,510	510,509
510,1	510,0	**509,511**	509,510	509,509
509,1	509,0	508,511	508,510	508,509
508,1	508,0	507,511	507,510	507,509

next 0,2	next 0,1	next 0,0	511,511	511,510
511,2	511,1	511,0	510,511	510,510
510,2	510,1	**510,0**	509,511	509,510
509,2	509,1	509,0	508,511	508,510
508,2	508,1	508,0	507,511	507,510

next 0,3	next 0,2	next 0,1	next 0,0	511,511
511,3	511,2	511,1	511,0	510,511
510,3	510,2	**510,1**	510,0	509,511
509,3	509,2	509,1	509,0	508,511
508,3	508,2	508,1	508,0	507,511

next 0,4	next 0,3	next 0,2	next 0,1	next 0,0
511,4	511,3	511,2	511,1	511,0
510,4	510,3	**510,2**	510,1	510,0
509,4	509,3	509,2	509,1	509,0
508,4	508,3	508,2	508,1	508,0

Figure 15 Blanking part 4

next 0,511	next 0,510	next 0,509	next 0,508	next 0,507
511,511	511,510	511,509	511,508	511,507
510,511	510,510	**510,509**	510,508	510,507
509,511	509,510	509,509	509,508	509,507
508,511	508,510	508,509	508,508	508,507

next 1,0	next 0,511	next 0,510	next 0,509	next 0,508
next 0,0	511,511	511,510	511,509	511,508
511,0	510,511	**510,510**	510,509	510,508
510,0	509,511	509,510	509,509	509,508
509,0	508,511	508,510	508,509	508,508

next 1,1	next 1,0	next 0,511	next 0,510	next 0,509
next 0,1	next 0,0	511,511	511,510	511,509
511,1	511,0	**510,511**	510,510	510,509
510,1	510,0	509,511	509,510	509,509
509,1	509,0	508,511	508,510	508,509

next 1,2	next 1,1	next 1,0	next 0,511	next 0,510
next 0,2	next 0,1	next 0,0	511,511	511,510
511,2	511,1	**511,0**	510,511	510,510
510,2	510,1	510,0	509,511	509,510
509,2	509,1	509,0	508,511	508,510

next 1,3	next 1,2	next 1,1	next 1,0	next 0,511
next 0,3	next 0,2	next 0,1	next 0,0	511,511
511,3	511,2	**511,1**	511,0	510,511
510,3	510,2	510,1	510,0	509,511
509,3	509,2	509,1	509,0	508,511

next 1,4	next 1,3	next 1,2	next 1,1	next 1,0
next 0,4	next 0,3	next 0,2	next 0,1	next 0,0
511,4	511,3	**511,2**	511,1	511,0
510,4	510,3	510,2	510,1	510,0
509,4	509,3	509,2	509,1	509,0

Figure 16 Blanking part 5

next 1,511	next 1,510	next 1,509	next 1,508	next 1,507
next 0,511	next 0,510	next 0,509	next 0,508	next 0,507
511,511	511,510	**511,509**	511,508	511,507
510,511	510,510	510,509	510,508	510,507
509,511	509,510	509,509	509,508	509,507

next 2,0	next 1,511	next 1,510	next 1,509	next 1,508
next 1,0	next 0,511	next 0,510	next 0,509	next 0,508
next 0,0	511,511	**511,510**	511,509	511,508
511,0	510,511	510,510	510,509	510,508
510,0	509,511	509,510	509,509	509,508

next 2,1	next 2,0	next 1,511	next 1,510	next 1,509
next 1,1	next 1,0	next 0,511	next 0,510	next 0,509
next 0,1	next 0,0	**511,511**	511,510	511,509
511,1	511,0	510,511	510,510	510,509
510,1	510,0	509,511	509,510	509,509

next 2,2	next 2,1	next 2,0	next 1,511	next 1,510
next 1,2	next 1,1	next 1,0	next 0,511	next 0,510
next 0,2	next 0,1	**next 0,0**	511,511	511,510
511,2	511,1	511,0	510,511	510,510
510,2	510,1	510,0	509,511	509,510

next 2,3	next 2,2	next 2,1	next 2,0	next 1,511
next 1,3	next 1,2	next 1,1	next 1,0	next 0,511
next 0,3	next 0,2	**next 0,1**	next 0,0	511,511
511,3	511,2	511,1	511,0	510,511
510,3	510,2	510,1	510,0	509,511

next 2,4	next 2,3	next 2,2	next 2,1	next 2,0
next 1,4	next 1,3	next 1,2	next 1,1	next 1,0
next 0,4	next 0,3	**next 0,2**	next 0,1	next 0,0
511,4	511,3	511,2	511,1	511,0
510,4	510,3	510,2	510,1	510,0

Figure 17 Blanking part 6

After all the additions are performed in parallel, there are 49 results waiting to be compared against each other to determine which value is the largest or the smallest, depending on the operation. Since it is impractical to have a 49-input comparator, a comparison tree composed of 2-input comparators and synchronization registers is a much more feasible solution. See Figure 10 for the general block diagram of the comparison tree. Each comparison stage consists of a comparator and an output holding register, with all of the output holding registers tied to a common clock, so it will take a minimum of 6 clock cycles to complete all of the comparisons needed to generate an output pixel. Since a comparison tree is being used and all comparators are properly clocked and synchronized, the tree will output one resulting pixel value per clock cycle. Each level of the comparison tree contains intermediate results for successive pixels. Further references to intermediate results of an operation on any input pixel $X_{i,j}$ will be denoted as $I_{i,j}$, with $I_{i,j}$ having the possibility of referring to numerous intermediate results.

For example, when the first pixel of the image ($X_{0,0}$) is in the target pixel position (the middle of the adder array), all 49 additions for that pixel are performed simultaneously. On the next clock cycle, the 49 results are sent into the first level of the comparison tree and the results are ready before the and of the clock cycle. Meanwhile, the second pixel of the image ($X_{0,1}$) is in the target pixel position and the 49 additions for that pixel are taking place. On the next cycle, the intermediate results of $X_{0,0}$ ($I_{0,0}$) are sent into the second level of the comparison tree, and $I_{0,1}$ is sent into the first level, and $X_{0,2}$ is now the target pixel. This will continue until $I_{0,0}$ is in the sixth level of the tree and $X_{0,6}$ is the target pixel. On the next cycle, the output of the final comparator in the tree will be the first output pixel of the morphological array processor, $Y_{0,0}$. The proposed tree in Figure 10 is only one possible way to implement it, many other arrangements can be used as long as the intermediate results remain synchronized.

D. The Logic Design of the MAP

It was decided that the logic design of the MAP would be done based on the architectural description, rather than to translate the logic design used on the board designed by Jeff Hanzlik and Jens Rodenberg. This choice was made in order to produce a more efficient (both faster and smaller) design. It has been the experience of this author that whenever translating between media or technologies (i.e. gate array to custom VLSI or nMOS to CMOS) the designer always inherits all the drawbacks of the source medium or technology, while never inheriting any of the benefits.

The logic design falls directly out of the architectural schematics. See figure 18 for the top level schematic. The adder array was broken down into seven rows of seven adders each (see figure 19). Each adder contained D flip-flops to latch its inputs and outputs (see figure 20). The adders themselves were implemented with simple ripple carrys, as they were only nine bits wide.

The weight mask circuitry of each adder block is built with a D flip-flop and a D latch. The extra latch allows for a new mask to be serially shifted into place in the array, while the old mask is still being used. This allows for no latency cost when the user changes masks with a new image. Once the new image is fully loaded in the shift register and is needed for the next addition cycle, it is then latched in with the signal "W_LATCH".

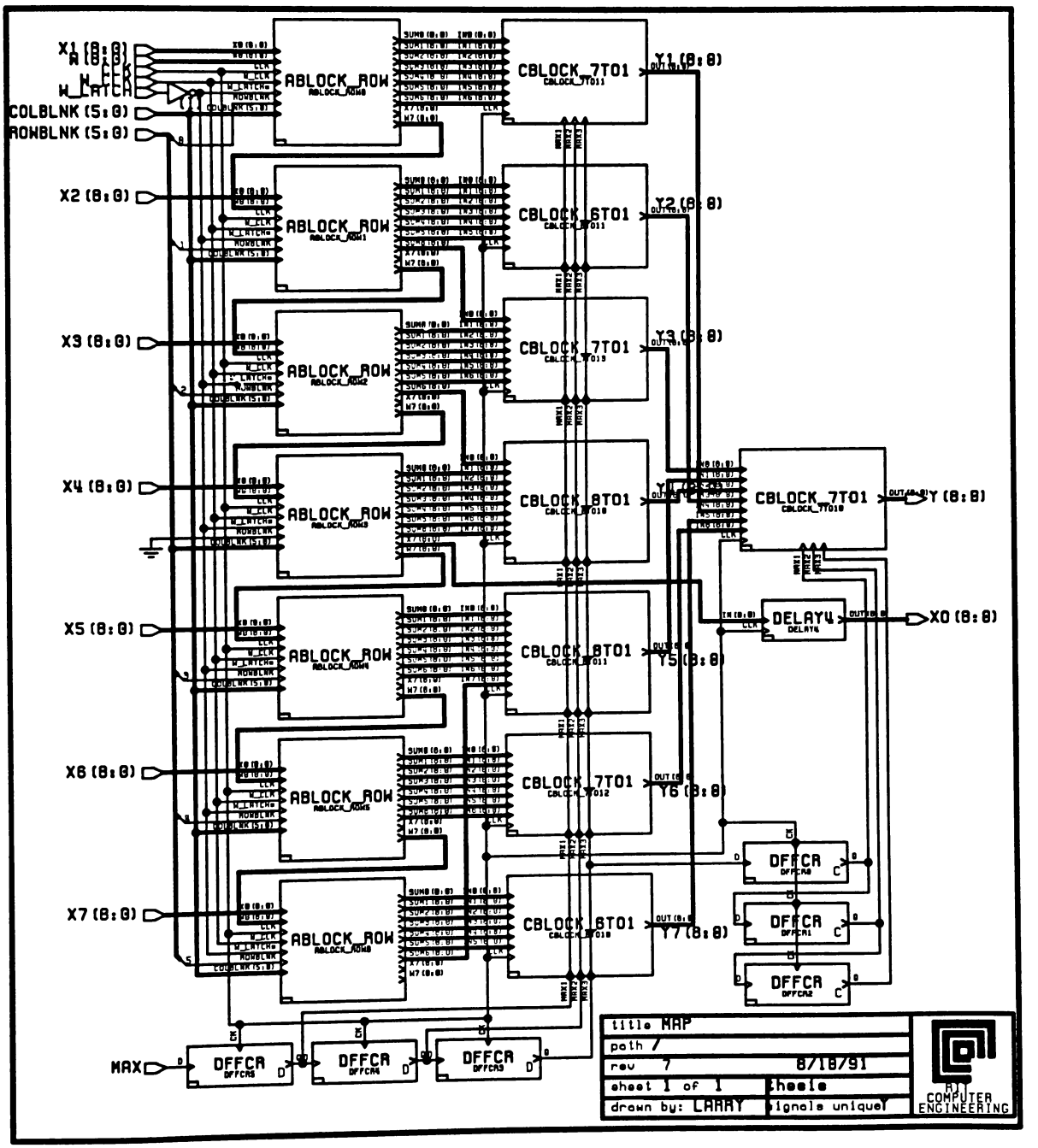

Figure 18 Top Level Schematic of MAP

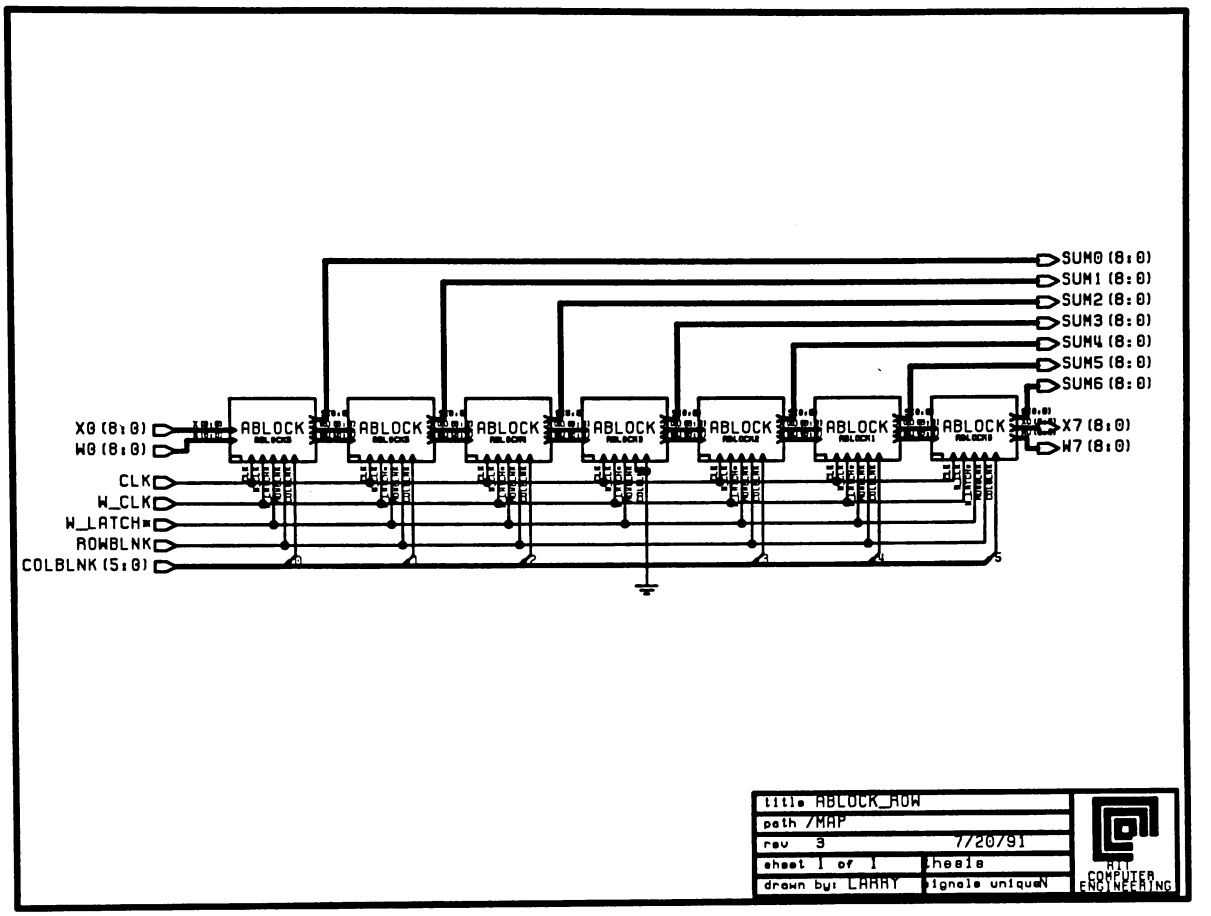

Figure 19 ABLOCK_ROW Schematic

Each of the forty nine adder blocks were identical. This made for simpler design and simplified the layout enormously. Each row of adder blocks had a unique blanking signal (to denote invalid pixel data), and a six bit column blanking signal. The middle adder block of every row did not have a column blanking signal, nor did he middle row have a row blanking signal. This was because the blanking signals represented the non-pixels beyond the edges of the image. The 7x7 mask could only go over the edge by three pixels in any direction, and therefore the center pixel could never be invalid unless explicitly represented as $-\infty$ in the data.

Exceptions that had to be handled by the adder block was that if either input was $-\infty$, or if the row or column blanking signal were active, then the result would be $-\infty$. This required added decode circuitry. Please see figure 21 for the circuitry that performs this task.

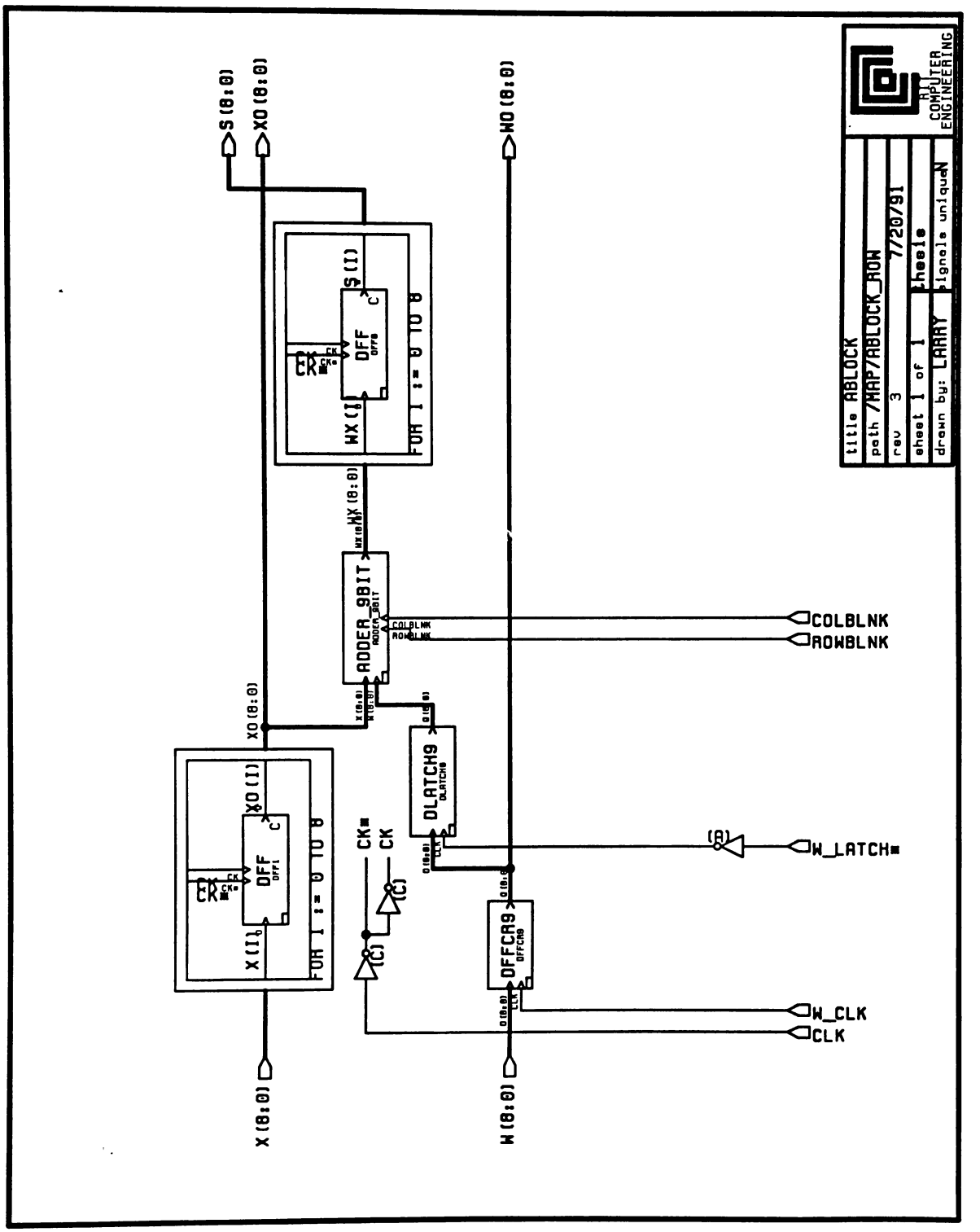

Figure 20 ABLOCK Schematic

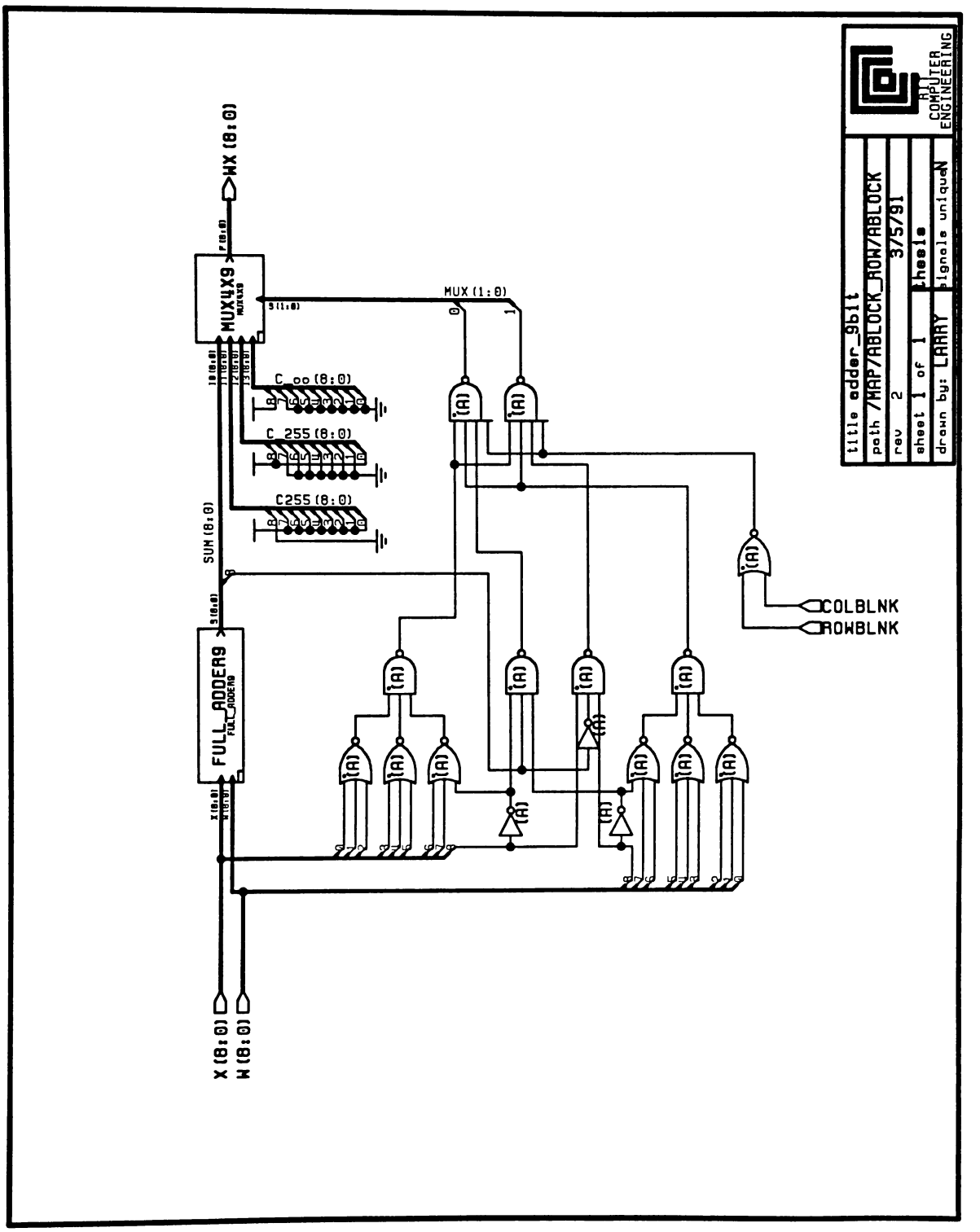

Figure 21 ADDER_9BIT Schematic

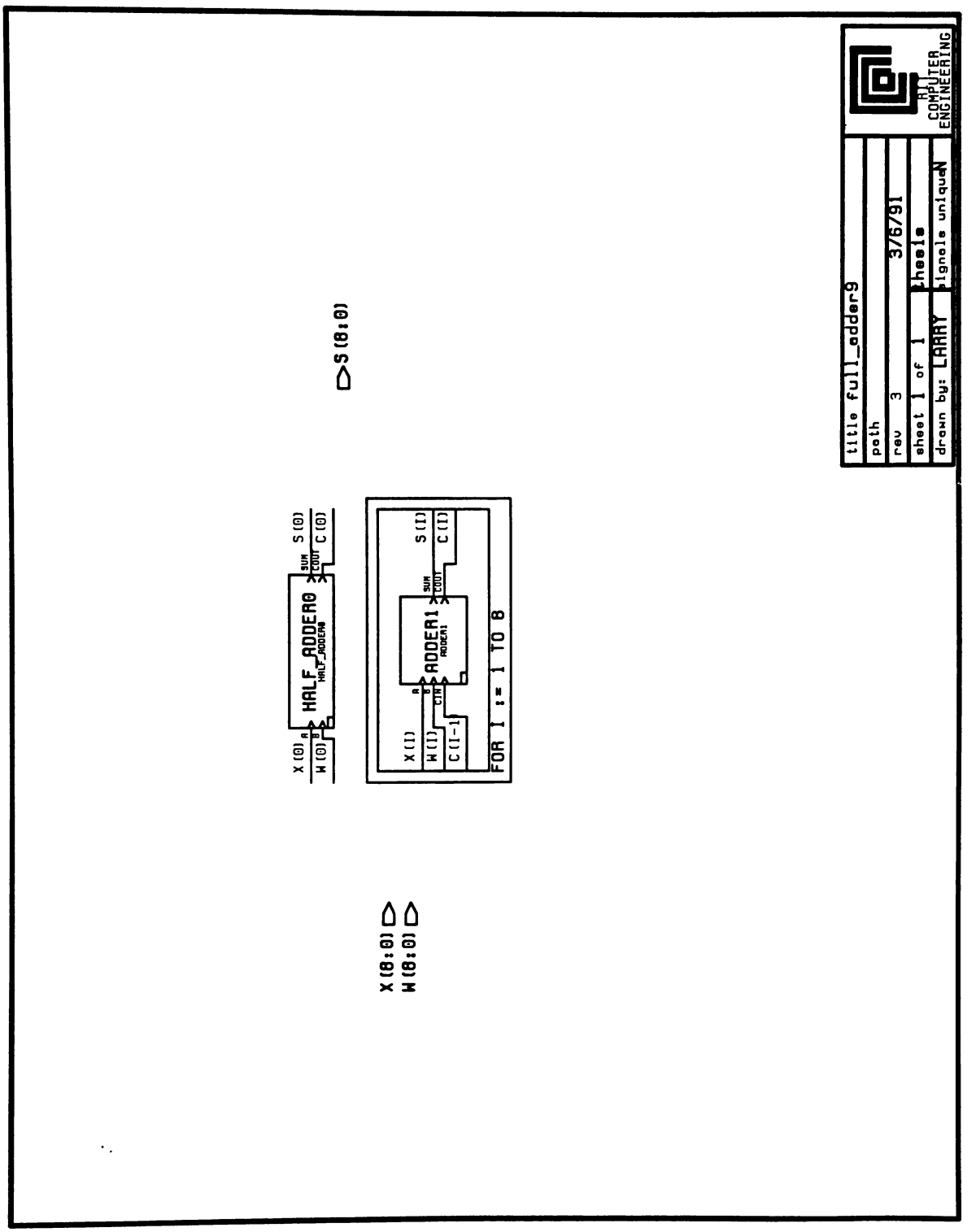

Figure 22 FULL_ADDER9 Schematic

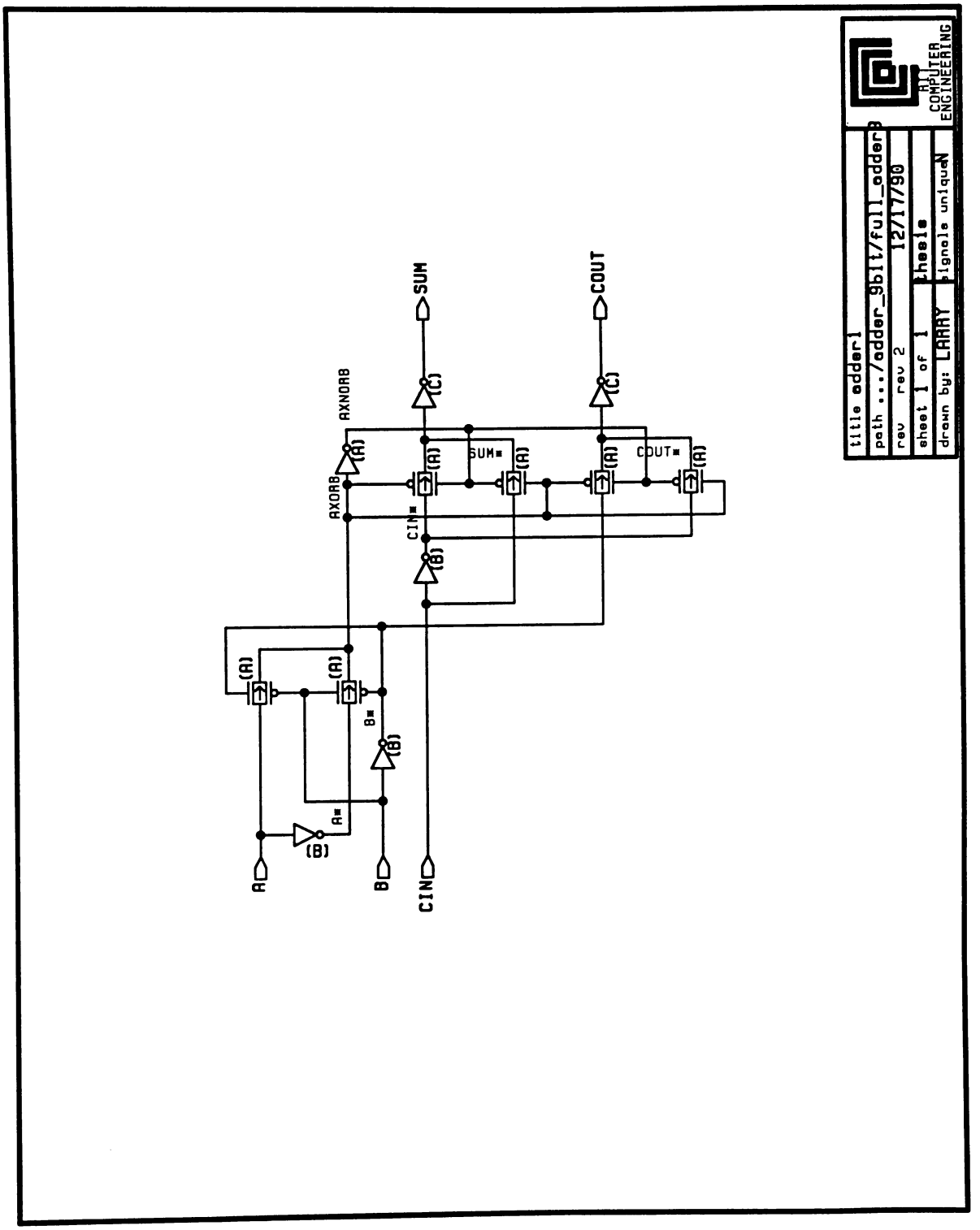

Figure 23 ADDER1 Schematic

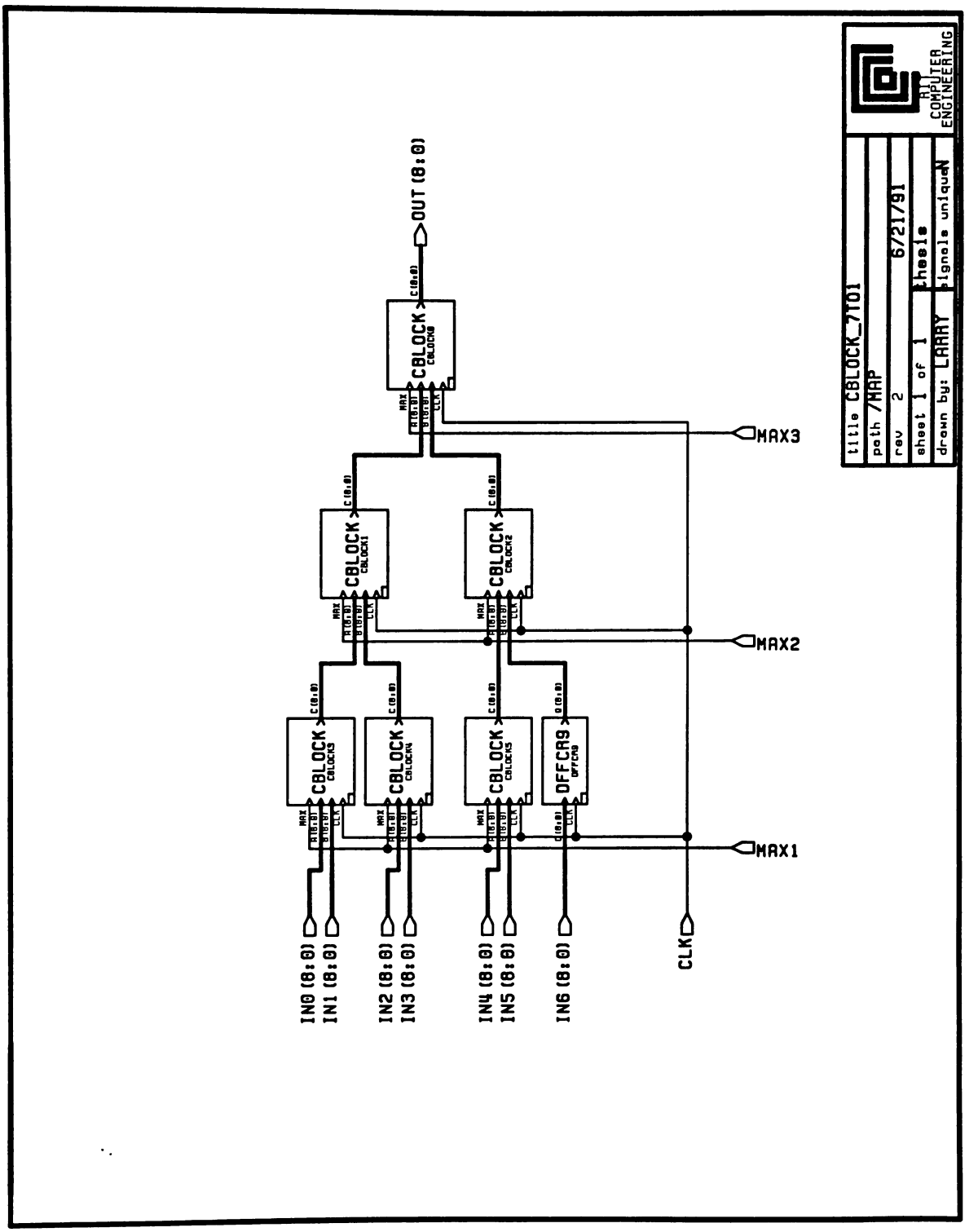

Figure 24 CBLOCK_7TO1 Schematic

The comparison tree of the 49 resulting sums was fairly straight forward. There were originally eight 7-to-1 comparators used (please see figure 24), with the last using the results of the first seven, and producing the final result. Each 7-to-1 comparator consisted of six comparison blocks and one nine bit register used to keep the timing correct.

While the layout was in its final stages, it became necessary to repartition the comparison tree in order to more efficiently fit on the maximum allowable die size. What was done was to move a single comparator block out of two of the 7-to-1 comparators into two others, thus making two 6-to-1 comparators, and two 8-to-1 comparators. This is a good example of the iterative nature of the design process.

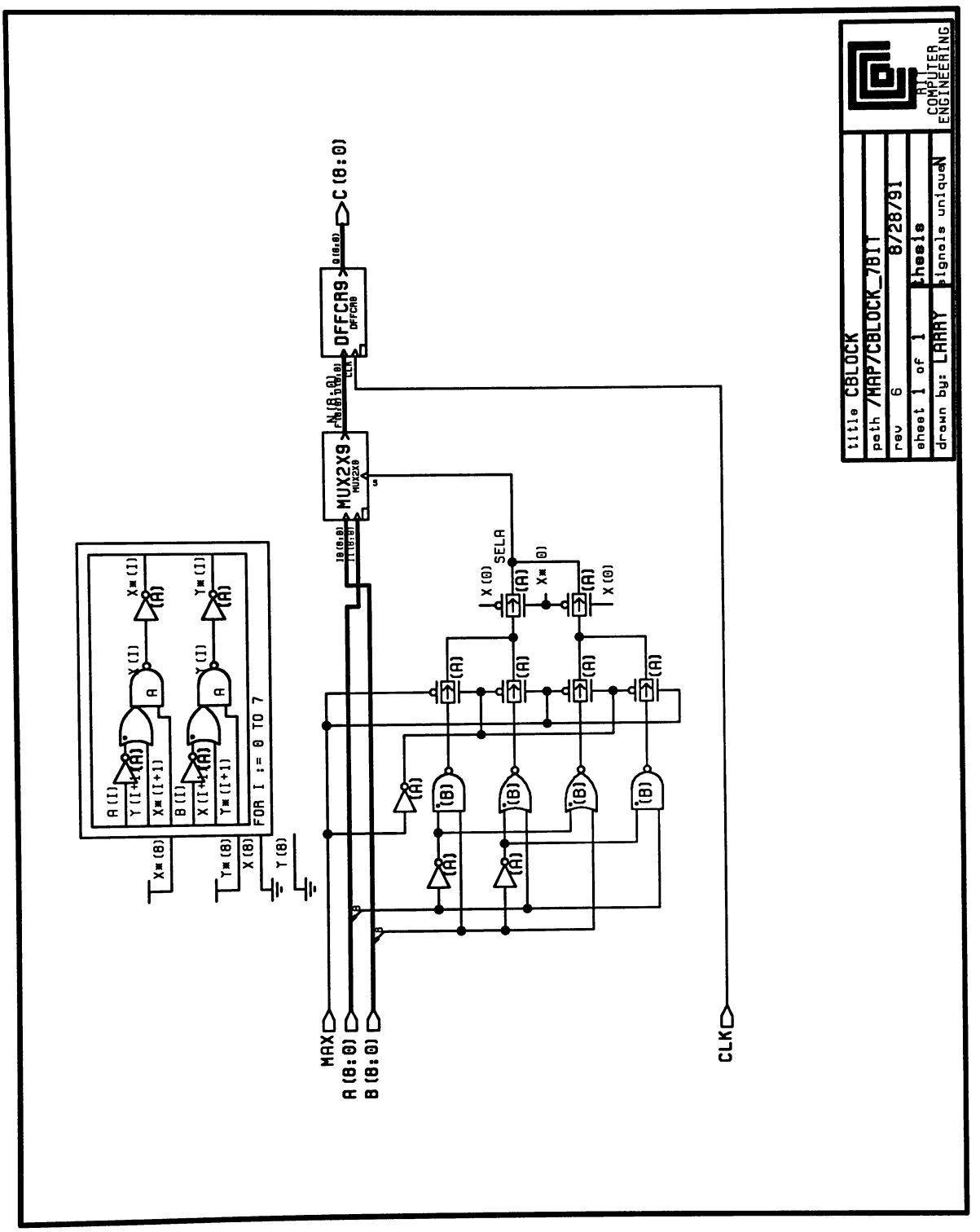

Figure 25 CBLOCK Schematic

In order to achieve maximum speed, the comparator block was implemented with an 8-bit ripple carry comparator and a multiplexer to select which input is the result. The exception that had to be handled was that if either input was $-\infty$ then the result would automatically be $-\infty$. Each stage of the comparator circuit itself (seen within the "FOR" frame in figure 8) consists simply of two inverters and a complex NAND gate, called "nd2_2". As its symbol indicates, the complex gate implements the function (A + B) * C. This is far more efficient in area, speed and power than to implement the same function with a NOR, inverter, and NAND gate.

Logical simulation was performed on the adder block and on the comparator block, and then on the top level of the design. For a listing of the Quicksim stimulus file and the results of the simulation of the comparator block, please see appendix A. As can be seen, the comparison block was functional.

It is interesting to note that while the architecture was defined top-down, the logic design and testing was performed bottom-up. This is for the somewhat obvious reason that one can not simulate a top level schematic successfully unless all its components are properly functional. Because the adder and comparison blocks were known to be repeated many times, particular care was taken to ensure that they would consume the least possible area, as well as satisfy the timing requirements.

E. The Circuit Design of the MAP

The actual circuit design of the MAP was minimal due to the simplicity of the logic. The chip was specified to operate at a frequency of at least 16 MHz. However, there exists very little circuitry between each D flip-flop, and therefore the speed paths were comparatively short.

To ensure that no delay paths longer than 63ns existed between clock edges (usually consisting of a D flip-flop driving through combinational logic to another D flip-flop), the worst case internal and external paths were examined for the adder and comparator blocks. The internal paths are defined as the worst case path within a single block, while the external paths are defined as the worst case paths which drive another block.

Because the adder blocks are logically arranged as a 49 deep 9-bit shift register for the image data (X buses) and the weight mask (W buses), the external paths have no combinational logic in them, but some have long layout lines. These had to be back annotated from the layout and simulated with representative capacitors added to simulation schematics. When this thesis was done, the library did not yet support Dracula II's layout parameter extractor (LPE), so some hand calculations had to be made. This was done by taking the longest layout path of each bus and hand calculating the capacitance based on process data received directly from MOSIS. The following calculation was made for the worst case X bus line:

8600μm Metal2 + 1200μm Metal1 = (.046 * 8600 * 3) + (0.45 * 1200 * 3) = 1.35 pF,

where both the Metal2 and Metal1 lines were 3μm wide, and 0.046 an 0.45 are the worst case capacitances (in F/μm^2) for Metal2 and Metal1 respectively.

The worst case path within the adder block was simulated using Accusim on the adder_9bit schematic (see figure 21). The stimulation file follows. Due to the complexity of the circuit, many initial conditions had to be added to the file in order for the simulator to converge on the initial DC operating point. Worst case conditions were used. Vcc was set to 4.3v, and the junction temperature was set to 125 °C.

The worst case speed path through an adder block was simulated to be approximately 25 ns. This is well within the 63 ns time that the 16 MHz specification requires. Please see figure 26 for a plot of the simulation results, and appendix B for the Accusim stimulus file.

The worst case timing from one adder block to the next was only 15 ns. The worst case timing from one comparator to the next was 13 ns. The worst case internal path of the comparator block was simulated to be approximately 21 ns. Therefore there is no problem with the adder or comparator blocks achieving the timing specifications of a 63 ns period clock.

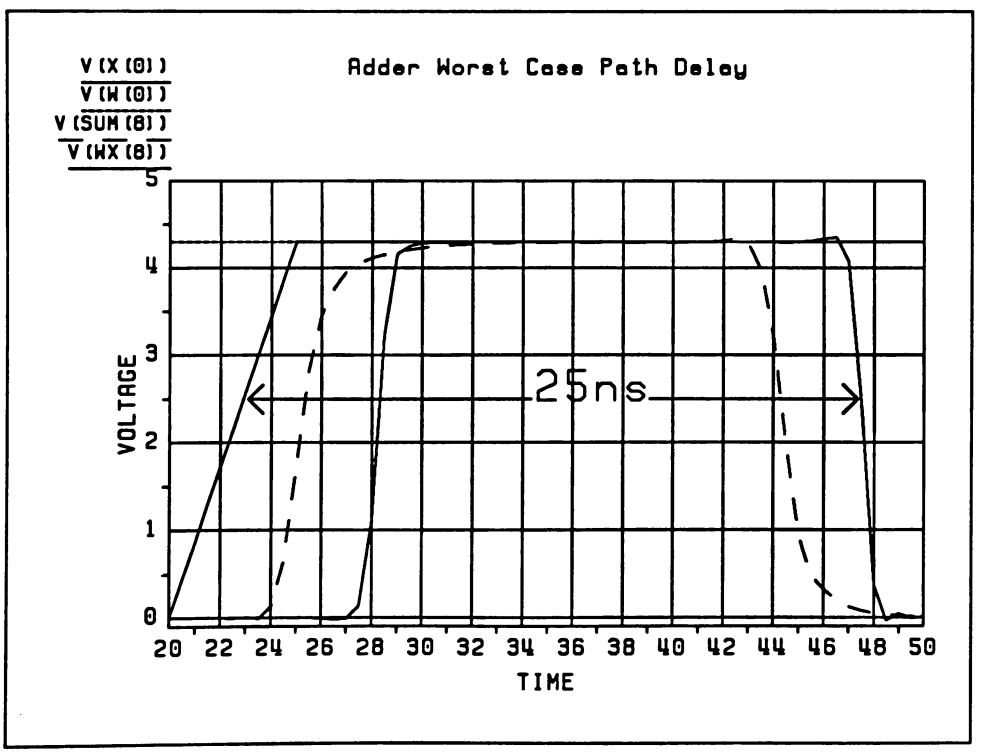

Figure 26 Adder Worst Case Path Delay

F. The Layout of the MAP

It is very common to underestimate the effort to lay out a full custom VLSI chip. The laying out of the MAP in particular required a full five months of full time work. By comparison, the logic and circuit design combined only required three months.

The major reason for the large time spent on the layout was that it had to undergo four major re-works in order to be compressed into the maximum allowable die size of 7900μm x 9200μm.

The layout was done "bottom up", with the basic cells consisting of the adder block (ablock), the comparator block (cblock), and a control block. The adder blocks were organized into rows of seven, and the comparator blocks were organized into 7-to-1 tree structures that aligned directly next to the adder block rows. Later, in order to fit into the required die size, some of the adder and comparator blocks were shuffled around. For the resulting floor plan, please see figure 27.

All layout was done completely by hand. It was chosen not to use any of the automatic place-and-route tools due to their inefficiency and the great need for very compact layout. Much time could have been saved using the automatic tools, but the result would never have fit in the allowable die size.

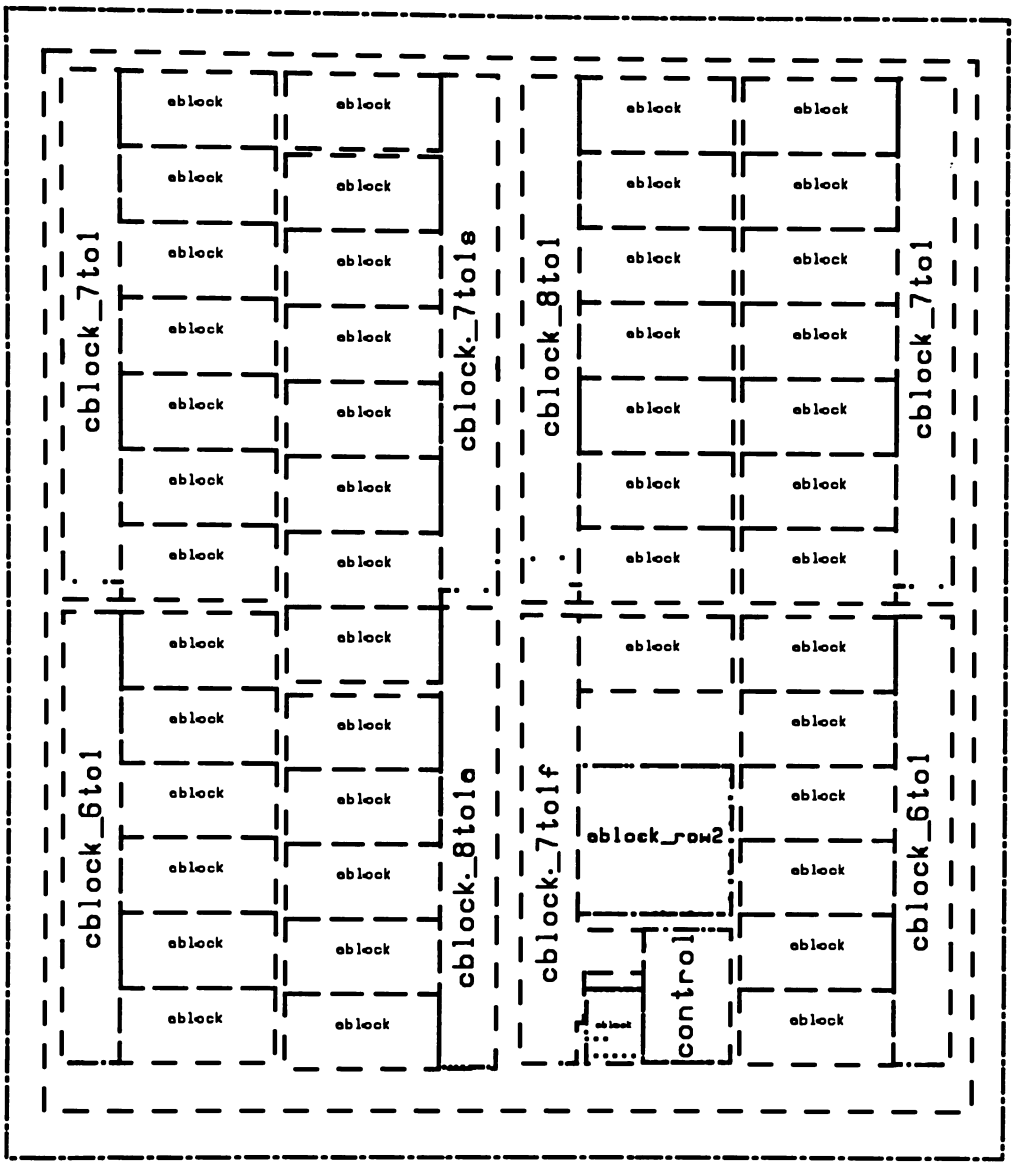

Figure 27 Floorplan of the MAP

The power routing was done using a mesh architecture. Each "strip" of logic has both VCC and VSS connected horizontally through metal 1 to a separate outer power ring which surrounds the chip. This outer power ring may be separate from the pad power ring (especially if there is a lot of electrical noise anticipated in the pad ring). Every 100 microns or so a vertical metal 2 line connects the VCC lines, and another for the VSS lines. This creates a mesh through which power can be distributed from four directions to any one point, as opposed to just one direction with a tree architecture. Please see figure 28 for examples of power routing architectures.

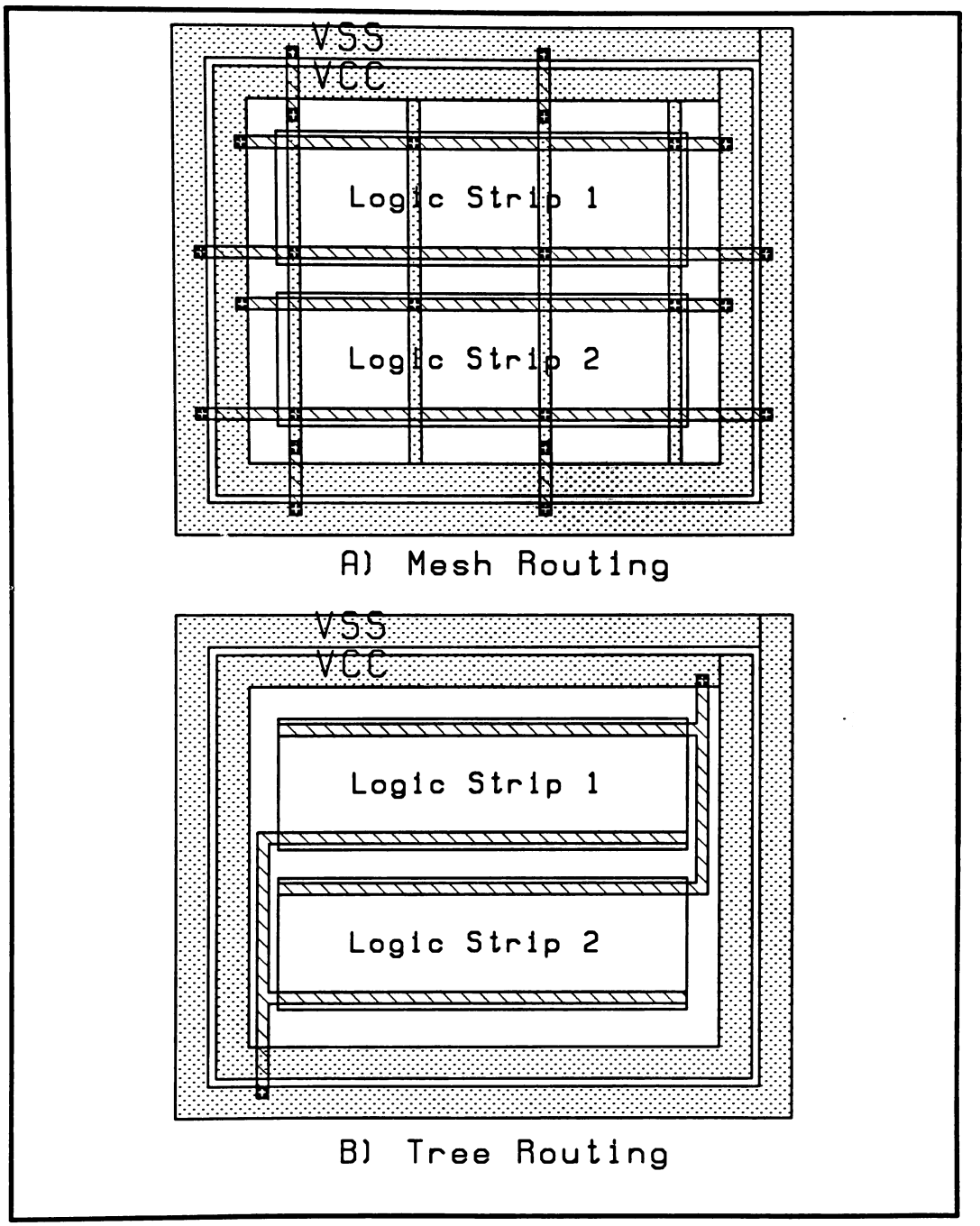

Figure 28 Power Routing Architectures

A mesh architecture has the added advantage of not requiring large internal power lines. This is because the power lines are evenly distributed throughout the layout, directly over the circuitry, and thus require no extra area. With the large amount of compaction that was required, this was a much needed bonus.

G. Suggested Testing Methodology for the MAP

There are two obvious methodologies for testing the MAP after it is fabricated. Each has its own advantages and disadvantages when comparing simplicity, cost, and completeness.

The first method would be to utilize the Tektronics LV500 testing system. The test vectors could be generated using the Tekwaves software on the HP/Apollos that is already in place. The principle advantage of this method is that is would be more complete. Frequency vs. failure (SCHMOO) plots could be made to determine the effects of frequency on failure rates, and accurate diagnostic test patterns could easily be generated.

The disadvantage of using the LV500 is the cost. Currently, the configuration only supports 64 channels, and at least 114 channels would be required to fully test the MAP. It would therefore be necessary to purchase expansion hardware for the extra channels, as well as a test fixture to support the 132 pin 1.4" PGA package.

The other obvious testing methodology is to create a testing platform. This could be done simply by hooking up the MAP to a bank of lamp/switch boards and cycling in the data by hand. More practically, a microprocessor emulation board such as the SDK-86 or the MC68HC11 boards could be programmed to run a set of predefined test vectors. This is the least expensive way to test the chip, but far from complete. Doing this would be tedious as specific programs would have to be written and debugged for these boards. It is unlikely that this would allow for testing at realistic speeds. Also, this strategy has greater potential for human error to enter into the test vectors.

With either testing methodology, it should be noted that to pass the entire 512x512 pixel images through the MAP is unnecessary, as the MAP only sees 7x7 of any sized image at a time. It would be far more efficient to therefore assume a minimum image size, which is at least larger than the size of the mask. It is therefore suggested to use an 8x8 pixel image size.

VI. Results

The following is a list of components/cells that currently comprise the RIT Standard Cell Library. A "D" indicates completion of the cell corresponding to that size of a given component. A "T" indicates that that size is listed as "To be done". Blank spaces indicate inappropriate or unnecessary cells that will not be implemented. The "N/A" indicates that the component only supports one size.

cell name	description	drive strength
		A B C D E F G H I J K L
and2	2 input AND	D D T T
and3	3 input AND	T T T T
and4	4 input AND	T T T T
and5	5 input AND	T T T T
buf	two inverter buffer	D D D D
cmux2	2 input CMOS pass gate MUX	D T T T T
cmux4	4 input CMOS pass gate MUX	T T T T T
cmux2nd	cmux2 with no decode	D T T T T
cmux4nd	cmux4 with no decode	D T T T T
conp	poly contact used by router	N/A
cxfr	cmos transmission gate	D D D D D D
crxfr	resistive cxfr	T

TABLE 6 Library Elements

Cell Name	Description	Drive Strength
		A B C D E F G H I J K L
dff	basic D flip-flop	D T
dffar	dff async reset	D T
dffarcf	async rst/falling clk	D T
dffarcr	async rst/rising clk	D D
dffarscr	async rst,set/rising clk	T T
dffas	async set	D T
dffascf	async set/falling clk	D T
dffascr	async set/rising clk	T T
dffcf	dff clk falling	D T
dffcr	dff clk rising	D T
dlatch	non-inverting latch	T T T T T T T T T T T T
full_adder	full adder	N/A
gnd_pad	ground pad	N/A
half_adder	half adder	N/A
ietri	tri-statable inverter	T T T
ilat	inverting latch	D T T T T T T T T T T T
in_pad	input pad	N/A
inv	inverter	D D D D D D D D D
inv_inv	inverter with feedback	T T T T
inv_pch	inverter with feedback nrxfr	T T T T T
invr	resistive inverter	N/A
io_pad	bidirectional pad	N/A
keeper	one pin inv_inv	T
nand2	2 input NAND	D D D D D
nand3	3 input NAND	D D D D D
nand4	4 input NAND	D D D D D
nand5	5 input NAND	D D D
nand6	6 input NAND	D D
nand7	7 input NAND	D D
nand8	8 input NAND	D
nd2_2	complex nand	D T T
ndlat	NAND latch	T T T T T T T T T T T
nmux2	nMOS pass gate MUX	D D D T T
nmux3	nMOS pass gate MUX	D D T T T
nmux4	nMOS pass gate MUX	D T T T T
nmux5	nMOS pass gate MUX	D D T T T

Library Elements [cont]

Cell Name	Description	Drive Strength
		A B C D E F G H I J K L
nmux2nd	nmux2 with no decode	T T T T T
nmux3nd	nmux3 with no decode	T T T T T
nmux4nd	nmux4 with no decode	T T T T T
nmux5nd	nmux5 with no decode	T T T T T
nor2	2 input NOR	D D D D D
nor3	3 input NOR	D D D D
nor4	4 input NOR	D D D D
nor5	5 input NOR	D D
nor6	6 input NOR	D D
nrlat	NOR latch	T T T T T T T T T T T
nrxfr	resistive nxfr	T
nxfr	nMOS transfer gate	D D D D D D D
or2	2 input OR	T T T
or3	3 input OR	T T T
or4	4 input OR	T T T
out_pad	tri-statable output pad	N/A
pxfr	pMOS transfer gate	D D D D D D D
prxfr	resistive pxfr	T
tribuf	tri-state buffer	T T T
tsg1	tri-state gate	T T T
tsg2	tri-state gate	T T T
vcc_pad	Vcc pad	N/A
via	metal1-2 via used by router	N/A
xor	exclusive OR	D T T T
xnor	exclusive NOR	T T T T

Library Elements [cont]

All of the above components are complete and have been netlisted to allow users to use them with the ECAD netlister. The cells have all been compiled into a library technology directory to be used by Cellgraph for automatic placement and routing. Menu options for the above components and their corresponding cells have all been added to the appropriate Neted and Chipgraph startup scripts.

In order to determine how much load a given drive size cell can drive, a simple schematic was drawn with single inverters driving capacitive loads. The acceptable rise and fall times were chosen to be 5 ns. This schematic was simulated by Accusim using worst case models. After some trial and error adjustment of the capacitor loads, the maximum capacitive loading capability for each drive size was determined.

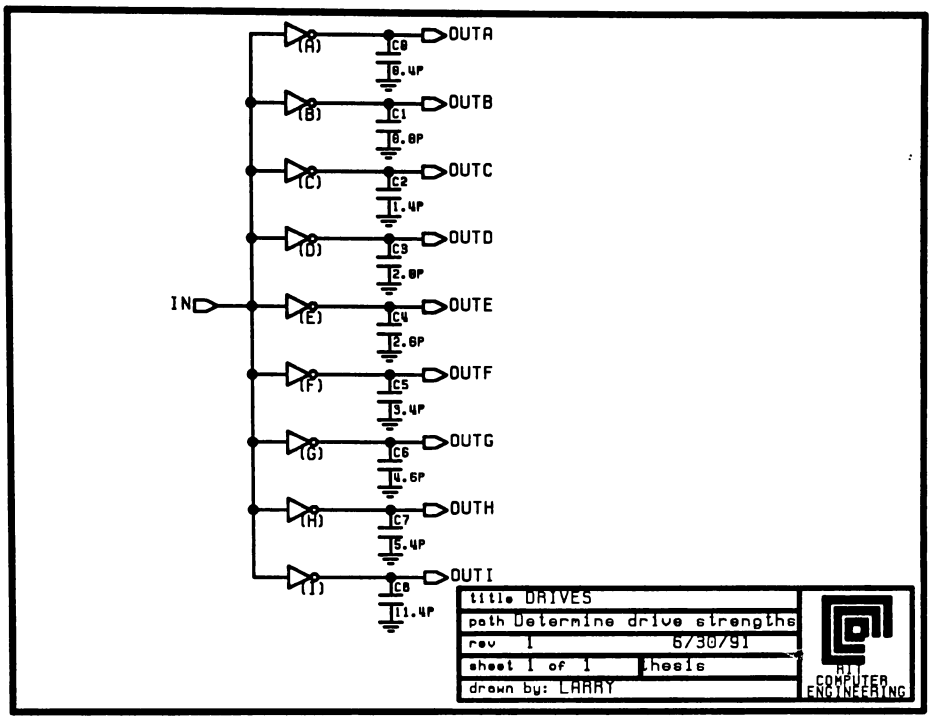

Figure 29 Maximum Loading Capabilities

Size	Width	Max. Load
A	8μm	0.4pF
B	16	0.8
C	24	1.4
D	32	2.0
E	40	2.6
F	48	3.4
G	64	4.6
H	80	5.4
I	160	11.4

TABLE 7 Drive Strengths

The MAP chip contains over 60,000 transistors. It will perform the morphological operations of erosion and dilation at the rate of one pixel per clock cycle (after initial latency). The chip is specified to operate at 16MHz, but has been simulated to operate at speeds up to 30MHz. For a copy of the final layout of the MAP, please see the fold out pages of appendix F. The layout has successfully passed DRC, and LVS checking. ERC and LPE were not performed on the MAP as they were not fully supported at the time of the MAP's design.

VII. Discussion

The library was built with the philosophy to be maximally flexible, user friendly for both the novice and experienced designer, upwardly compatible, and above all, useful for real projects. In many ways this has been achieved. The many drive sizes available for most of the components allows for flexibility. The designer may design circuits both at the logic gate or the more efficient transmission gate level. The library has already been used to generate chips that have been fabricated at MOSIS.

Great care must be taken for the maintenance of the library. If any component symbol or any cell's external layers are altered **in any way**, every user who uses that component or cell must update every schematic or layout where it was used. If the administrator simply calls Symed or Chipgraph in edit mode (the default) on the symbol or cell and saves the session when exiting (again, the default), the symbol or cell will be effectively altered, even if the administrator performed no edit function. The administrator must therefore take great care when even viewing the components and cells in order to avoid causing many users much trouble.

The library comprises mostly SSI parts. It is hoped that the library will be a dynamic thing, always growing and adjusting to its environment. Eventually larger, more integrated components should be added to enable larger, more complex projects.

For a new component to be added, a schematic and symbol for it must be created in the library directory and the specially written component menu for Neted should be edited to allow the users easy access to it. For a new cell to be added, it must be drawn with Chipgraph and a simple "build library" script must be run to re-compile the information for the place-and-route tools.

One of the unsatisfactory results of the library is the inefficiency of the automatic place-and-route tool, Cellgraph. The placement algorithms it uses are adequate, but the routing leaves much to be desired. Some routes will make loops and cross themselves, or go a long way around an empty area. This makes it necessary for the user to clean up much of the routing. A possible explanation for the inefficiency is the choice of several of the input parameters to Cellgraph. Some work will have to be done in the future to experiment with different values of these parameters to produce better routing.

The logic and circuit design of the MAP are not very complex by today's standards. The logic testing was done hierarchialy. Due to the simplicity of the design, most of the logic required little or no debugging.

As stated before, the circuit design partitioned all the possible timing paths into internal and external block paths. The internal block paths were simple to find and accurately simulate. They required very little back annotation of the layout capacitances. This was due to the fact that all the nodes within a block are relatively small with insignificant capacitive loading. The external worst case inter-block paths are somewhat more difficult to find. Mentor Graphics supplies a circuit path finder

(CPF) utility, but RIT's Department of Computer Engineering does not currently have that product. Incidentally, CPF also uses SILOS format netlists.

Back annotation of the top level of the MAP layout could not be done as the current configuration of the LPE software exceeds its limit of 500,000 capacitors. It seems to be making a separate capacitors for each metal2 to metal1, metal2 to poly, et cetera, rather than supplying simply one sum capacitance value for each layout node. Until this problem can be solved, the LPE will not be useful to the larger sized projects, which need it most.

Much effort had to be made in order to make the design fit onto the maximum allowable sized die. This effort was the largest single task during the entire implementation of the library or the MAP. It is very simple to under-estimate the work required to layout a chip, especially one of over 60,000 transistors. If most of the chip was not an array of identical elements, the layout of a comparably sized chip could easily require a full man-year of work or more.

VIII. Conclusions

RIT now has a complete, fully integrated, fully characterized and tested CMOS standard cell library. The library enables students and faculty to design low power, high speed chips with a minimum of design time taken up by layout. With the proper use of the simulators, the library can be used to generate chips with a reasonable expectation of working first silicon, thus saving time and expensive fabrication costs.

The library allows students, both undergraduate and graduate, to create bigger, more complex designs for their various assignments or research projects. Already, the part of the library that is complete has been used by three VLSI undergraduate classes, and is being used by four graduate theses, including projects like an image processing array (this thesis) and its controller, and an eight bit microprocessor.

Due to the scalability of the library, a larger scale (5 to 10 micron feature sizes) may be used by the Microelectronic Engineering department at RIT to fabricate designs made by undergraduates in the Center for Microelectronics and Computer Engineering.

By generating the component and cell library and having a few of the cells fabricated at MOSIS, the groundwork has been completed to allow a path to generate more complex, higher integrated cells and have them all physically characterized. This will be done by submitting a series of test chips to MOSIS.

Prototype cells from the standard cell library have already been manufactured by MOSIS on the same test chip as some cells from the National Security Agency's cell library and have compared favorably. The RIT cells are 1/2 to 1/3 of the size with the same output drives, have much less input capacitances, better internal connections, and are less noise prone. The RIT Standard Cell library also offers a much wider range of drive sizes, thus allowing the designer more flexibility, and hence more efficiency.

A morphological array processor has been implemented using an 2.0μm N-well CMOS process to be fabricated through the MOSIS program. This chip fully implements the morphological operations of erosion and dilation on a 7x7 matrix, and will operate on a 512x512 image in real time (60 images per second). The MAP is designed to be pipelined for multiple successive morphologic operations on a series of images. Multiple MAPs can also be used to perform multiple morphologic operations on a series of images in real time.

Once the initial latency of loading the image into the MAP is completed, every clock cycle processes another pixel. There is no latency cost between different images nor different weighing masks. A new weighing mask may be shifted in independent of what stage the image data is in.

The MAP is independent of the size of the processed image, but will only operate at an estimated worst case of 30MHz. In order to operate on images in real time (60 per second), a maximum size of approximately 700x700 pixel images must be used. Therefore, if it is desired to operate on 1024x1024 pixel images in real time, a smaller feature size must be used.

MOSIS currently supports a 1.2μm N-well CMOS process, which is a directly scaled version of the process used by the MAP design. The MAP could then be "flattened" and "magnified" (simple Chipgraph commands), and the resulting data base could be shipped off to MOSIS for fabrication. The total rework of the layout would require less than a half a day. It would be strongly advised, however, that some circuit simulation be redone to insure that the long layout lines will not exhibit destructive transmission line behavior at the higher speeds. A disadvantage to the smaller feature size is that the current pricing schedule has the 1.2μm projects costing about 4 times that of the equivalent project implemented in 2.0μm technology.

Future Work

A design library is never complete. It must always grow with the environment it is within, as well as the needs of its users. There are many additions that should eventually be added to the library, including components, cells, macros, and support for newer, more efficient tools.

More complex NAND and NOR components need to be generated. These allow for much better speeds and area reductions within combinational logic. An example of these is ND2_2, which is used in the comparison block to implement $\overline{(A + B) * C}$.

The automatic place-and-route tools need to be modified to produce better results.

As the library currently exists, it is composed of only Small Scale Integrated (SSI) parts. Medium Scale Integrated (MSI) components like registers and 8 bit adders should be developed next. Later Large Scale Integrated (LSI) components like ALUs and multipliers should be developed.

More design automation tools can be developed for the library. Very useful tools include a programmable logic array (PLA) or read only memory (ROM) generators where the user would enter equations or a truth table, and have the layout automatically generated. This could easily done by having standard sized arrays and simply have the tools program the array with the necessary connections, or "plugs".

User configurable LSI components might one day be implemented, where the user may answer a list of questions and have a custom designed component automatically supplied. For example, the user might give the customizing tool a request for an ALU with 16 bit inputs, 8 specific functions, and a bi-directional shift register on its output. The tool would then automatically generate the layout and possibly the schematic as well (or at the minimum a descriptive symbol).

When Mentor Graphics comes out with its version 8.0, all the netlisters, macros, and scripts will need to be re-written. The user interface language will no longer be supported and will be replaced by a C like language called "Ample". Particular care will have to be taken to preserve the data that is currently defined by the expand scripts. The 8.0 version software does not explicitly expand designs any longer, but seems to require the data to be in the design database when it is drawn. More understanding must be achieved of what is required to make this library fully compatible with the 8.0 software.

The Mentor Graphics layout verification tool set, Checkmate, would be a very good replacement of Cadence's Dracula II tools. Checkmate would be far more efficient of the system resources, produce results much faster, and (above all) be much less expensive for the department.

The MAP and its controller and support hardware need to be developed on a new board. The board should be designed initially to operate within an AT bus environment, and eventually ported to other platforms as any potential customer requires. The first board should have only one MAP, but future boards may have several MAPs arranged in series or parallel to provide for multiple morphologic operations on an image in real time, or to allow for multiple images to be processed simultaneously.

IX. References

An Introduction to CAD for VLSI, by Stephan M. Trimberger, Kluwer Academic Publishers, 1987

Physical Design Automation of VLSI Systems, by Bryan Preas and Michael Lorenzetti, Benjamin/Cummings Publishing, 1988.

IDEA Series Schematic Capture Reference Manual Vol. I Function Keys and Menus, Mentor Graphics Corporation 1989

IDEA Series Schematic Capture Reference Manual Vol. II Command Dictionary and System Functions, Mentor Graphics Corporation 1989

IDEA Series Schematic Capture User's Manual, Mentor Graphics Corporation 1989

EXPAND User's and Reference Manual, Mentor Graphics Corporation 1989

QUICKSIM Family Reference Manual, Mentor Graphics Corporation 1989

QUICKSIM User's Manual, Mentor Graphics Corporation 1989

Accusim User's Manual, Mentor Graphics Corporation 1989

Analog Simulators Reference Manual, Mentor Graphics Corporation 1989

Chipgraph Reference Manual Vol. I Command Dictionary and System Functions, Mentor Graphics Corporation 1989

Chipgraph User's Manual, Mentor Graphics Corporation 1989

CellStation Reference Manual, Mentor Graphics Corporation 1989

CellStation User's Manual, Mentor Graphics Corporation 1989

Digital Image Processing by William K. Pratt, John Whiley & Sons, Inc. 1991

Morphological Methods in Image and Signal Processing by Dr. Edward R. Dougherty and Dr. Charles R. Giardina, Prentice-Hall 1988.

Matrix Structured Image Processing by Dr. Edward R. Dougherty and Dr. Charles R Giardina, Prentice-Hall 1987

A. Logical Simulation Files

The following test vector file was run through the logic simulator, Quicksim to verify the functionality of the comparator block:

```
scale trace time 40
TRAce MAX CLK A B SELA C
#
# Setup the Clock
#
FORCe CLK 0 0
CLOck Period 40
FORCe CLK 0 18 -Repeat
FORCe CLK 1 38 -Repeat
#
# Test minimum checking
#
FORCe MAX 0
FORCe A 0
FORCe B 0
RUN 40
FORCe A 100
RUN 40
FORCe A 145
RUN 40
FORCe A 0ff
RUN 40
FORCe A 45
RUN 40
FORCe B 0ff
RUN 40
FORCe B 100
RUN 40
FORCe B 45
RUN 40
FORCe B 123
RUN 40
#
# Test maximum checking
#
FORCe MAX 1
FORCe A 0
FORCe B 0
RUN 40
FORCe A 100
RUN 40
FORCe A 145
```

```
RUN 40
FORCe A 0ff
RUN 40
FORCe A 45
RUN 40
FORCe B 0ff
RUN 40
FORCe B 100
RUN 40
FORCe B 45
RUN 40
FORCe B 123
RUN 40
```

The resulting vectors follow, along with comments to the right which explain the results as they occur.

```
  0.0 0  0  000  000  X  XXX
 38.0 0  1  000  000  0  XXX
 40.0 0  1  100  000  0  XXX
 80.0 0  1  145  000  0  000     min(0,0) = 0
120.0 0  1  0FF  000  0  100     min(-∞,0) = -∞
160.0 0  1  045  000  1  145     min(-187,0) = -187
200.0 0  1  045  0FF  1  000     min(255,0) = 0
240.0 0  1  045  100  0  000     min(69,0) = 0
280.0 0  1  045  045  1  045     min(69,255) = 69
320.0 0  1  045  123  0  100     min(69,-∞) = -∞
360.0 1  1  000  000  1  045     min(69,69) = 69
400.0 1  1  100  000  1  123     min(69,-221) = -221
440.0 1  1  145  000  1  000     max(0,0) = 0
480.0 1  1  0FF  000  1  000     max(-∞,0) = 0
520.0 1  1  045  000  0  000     max(-187,0) = 0
560.0 1  1  045  0FF  0  0FF     max(255,0) = 0
600.0 1  1  045  100  1  045     max(69,0) = 69
640.0 1  1  045  045  0  0FF     max(69,255) = 255
680.0 1  1  045  123  1  045     max(69,-∞) = 69
720.0 1  1  045  123  0  045     max(69,69) = 69

TIME  ^max  ~a       ^sela
        ~clk   ^b      ~c
```

B. Physical Simulation Files

The following is an input file for the Accusim simulation of the longest path within the adder block. It initializes the adder to all zeros and then adds 1 with 1FF, thus exercising the longest possible carry path.

```
# Accusim simulation file for adder_9bit
#
# start x and w at 0, then have x go to 1 and w go to 1ff
# this makes for the longest carry chain
# use rise time = fall time = 5ns

force pul v x(0) 0 4.3 15 5 5 40
force dc v x(1) 0
force dc v x(2) 0
force dc v x(3) 0
force dc v x(4) 0
force dc v x(5) 0
force dc v x(6) 0
force dc v x(7) 0
force dc v x(8) 0
force dc v rowblnk 0
force dc v colblnk 0
force pul v w(0) 0 4.3 0 5 5 40
force pul v w(1) 0 4.3 0 5 5 40
force pul v w(2) 0 4.3 0 5 5 40
force pul v w(3) 0 4.3 0 5 5 40
force pul v w(4) 0 4.3 0 5 5 40
force pul v w(5) 0 4.3 0 5 5 40
force pul v w(6) 0 4.3 0 5 5 40
force pul v w(7) 0 4.3 0 5 5 40
force pul v w(8) 0 4.3 0 5 5 40
Initial Condition c(0) 0
Initial Condition c(1) 0
Initial Condition c(2) 0
Initial Condition c(3) 0
Initial Condition c(4) 0
Initial Condition c(5) 0
Initial Condition c(6) 0
Initial Condition c(7) 0
Initial Condition c(8) 0
Initial Condition s(0) 0
Initial Condition s(1) 0
Initial Condition s(2) 0
Initial Condition s(3) 0
Initial Condition s(4) 0
Initial Condition s(5) 0
```

Initial Condition s(6) 0
Initial Condition s(7) 0
Initial Condition s(8) 0
Initial Condition /FULL_ADDER9/HALF_ADDER#1/A* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#1/B* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#1/AXORB 0
Initial Condition /FULL_ADDER9/HALF_ADDER#1/AXNORB 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#1/CIN* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#1/SUM* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#1/COUT* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#2/A* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#2/B* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#2/AXORB 0
Initial Condition /FULL_ADDER9/HALF_ADDER#2/AXNORB 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#2/CIN* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#2/SUM* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#2/COUT* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#3/A* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#3/B* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#3/AXORB 0
Initial Condition /FULL_ADDER9/HALF_ADDER#3/AXNORB 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#3/CIN* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#3/SUM* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#3/COUT* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#4/A* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#4/B* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#4/AXORB 0
Initial Condition /FULL_ADDER9/HALF_ADDER#4/AXNORB 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#4/CIN* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#4/SUM* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#4/COUT* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#5/A* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#5/B* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#5/AXORB 0
Initial Condition /FULL_ADDER9/HALF_ADDER#5/AXNORB 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#5/CIN* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#5/SUM* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#5/COUT* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#6/A* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#6/B* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#6/AXORB 0
Initial Condition /FULL_ADDER9/HALF_ADDER#6/AXNORB 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#6/CIN* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#6/SUM* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#6/COUT* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#7/A* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#7/B* 4.3

```
Initial Condition /FULL_ADDER9/HALF_ADDER#7/AXORB 0
Initial Condition /FULL_ADDER9/HALF_ADDER#7/AXNORB 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#7/CIN* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#7/SUM* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#7/COUT* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#8/A* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#8/B* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#8/AXORB 0
Initial Condition /FULL_ADDER9/HALF_ADDER#8/AXNORB 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#8/CIN* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#8/SUM* 4.3
Initial Condition /FULL_ADDER9/HALF_ADDER#8/COUT* 4.3

keep v x(0) x(1) x(2) x(3) x(4) x(5) x(6) x(7) x(8)
keep v w(0) w(1) w(2) w(3) w(4) w(5) w(6) w(7) w(8)
keep v sum(0) sum(1) sum(2) sum(3) sum(4) sum(5) sum(6) sum(7) sum(8)
keep v wx(0) wx(1) wx(2) wx(3) wx(4) wx(5) wx(6) wx(7) wx(8)

trans 0.5 40.0
```

Rochester Institute Of Technology's

CMOS Standard Cell Library

User's Manual

Lawrence Rubin

August 18, 1991

C.1 Overview Of A Design Flow

For the purposes of this manual the methodology for a top-down design is partitioned into ten steps. The flow of these steps are shown in figure 1.

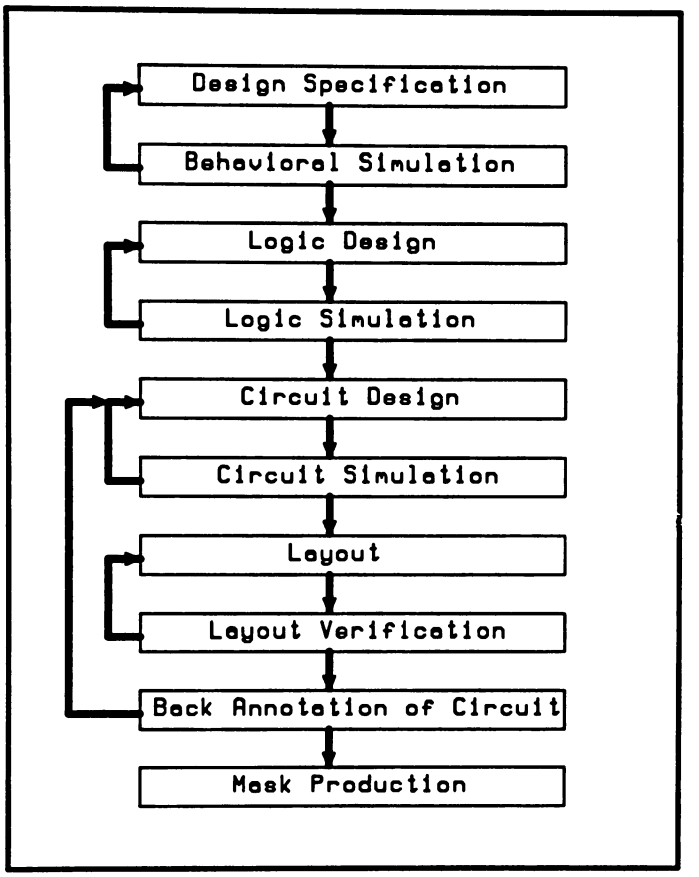

Figure C-1 Design Flow

The design specification, or architectural step consists of creating a definition of the chip's function. This is accomplished by the generation of block diagrams, timing charts, and behavioral models. Mentor Graphics Corporations' electronic design automation software supports models written in C, Pascal, FORTRAN, or VHDL.

For behavioral simulation, some simple test vectors are applied to the behavioral model to verify that it performs as expected. Mentor Graphics' logic simulator, Quicksim, is used for behavioral simulation.

Logic design consists of the schematic capture of a gate level implementation of the design. Mentor Graphics' Neted is used for this.

During logic simulation the design is checked for functionality. The timing of the gates is given in unit delays. Again, Mentor's Quicksim is used for this.

During circuit design each gate is sized to drive its load efficiently, and some minor alterations in the logic design may be required in order to achieve desired timing requirements. Mentor Graphics' Neted is used to edit the schematics. Please note that the same symbols are to be used for both logical and circuit simulation. This is an enhancement to the libraries originally supplied by Mentor Graphics.

During circuit simulation the design is checked against timing requirements. Timing is based on transistor level models. This includes the loading capacitances caused by transistor gate and junction capacitances. After the line capacitances are extracted from the initial layout, they are back annotated (added) into the circuit database and re-simulated to provide more accurate timing analysis. Mentor Graphics' Accusim (a SPICE simulator) performs this step.

The layout of the chip consists of several steps; cell placement and routing for each block, block placement and routing for the chip, power line routing, and pad placement. A combination of tools contained in Mentor Graphics' Chipgraph and Cellgraph packages are used for these tasks. At this point a schematic and a corresponding initial layout have been generated and need to be checked for consistency.

Layout verification is divided into three parts;
1) design rule checking (DRC),
2) electrical rule checking (ERC), and
3) layout vs. schematic comparison (LVS).

LVS verifies that the circuit extracted from the layout is the same as the circuit designed. DRC verifies that the design rules (line widths, spacings, etc.) are met. ERC checks the circuit extracted from the layout against electrical errors such as shorts and floating nodes. Cadence Corporation's Dracula II is used for all these functions, as well as extracting the parasitic capacitances and diodes from the layout so that they can be back annotated in the circuit design to enable more accurate circuit verification.

C.2 Setting Up Your Account

Before the RIT CMOS Standard Cell Library can be used, links must be created in the user_data directory of your account. Go to that directory and type:

```
$ : crl neted.startup3 /user/larry/thesis/neted/neted.startup
$ : crl chipgraph.startup3 /user/larry/thesis/chipgraph/chipgraph.startup
```

These pointers will allow Neted and Chipgraph to use the library startup files which redefine the menus and some of the function keys to make these tools more efficient. These links only need to be made once.

C.3 Entering The Schematic

Invoke Neted with the following syntax:

$: neted component -st

The "-st" tells Neted to use the startup file it will find in your user_data directory.

C.3.1 Initializing The Schematic

When Neted comes up, the component library menu will have three elements, "CMOS library", "set sheet size", and "macros". When you first create a schematic, select the "set sheet size" entry, and then the appropriate size of "A", "B", or "C". For the majority of the schematics you will be doing, size A should be appropriate. You will be prompted for certain information that will appear on a title block on the schematic. The suggested use for the fields follow:

schematic title:	The name of the schematic you're editing.
logic hierarchy pathname:	The path of the schematic within your design.
sheet number:	Usually 1
total sheets:	Usually 1

When you are ready to save the schematic, type cntrl-Y. This will automatically check the schematic, increment the revision number and update the date on the title block, and write the schematic to disc. An added safety feature is that cntrl-Q has been disabled to prevent corruption of the database.

C.3.2 Adding Components

After the title block has been made, you are ready to start editing. In the component libraries menu, select the "back" option and then go down into the "CMOS library" entry.

Select and place the various components as you would normally do. Some components (NANDs, NORs and INVs) have a "bub" option. This is merely a different symbol to represent a DeMorgan version of the same logical component.

A complete set of components is presented in this library, including some that are used only for circuit simulation (see the "spice only" entry). The library is designed to be self-contained, and while no problems are known to occur when using components from other libraries, it would be best to stick to using only these components whenever possible.

The components represent both logical and physical elements. Each element has a "size" or a "sfx" (suffix) property to represent it's size and drive strength. The size property is represented by a letter in parentheses, while the sfx property is represented by a single letter. All size properties are initialized to "(X)", and all sfx properties are

initialized to "X". The size properties will not work if they are not kept within parentheses, nor will the sfx properties work if they are kept within parentheses.

For logical simulation, no altering of the size or sfx properties are necessary.

When the circuit is expanded for circuit simulation, a lookup table is used to convert the size properties into numerical values. For the table containing the drives sizes and the effective transistor widths they represent, please see table C-1.

All lengths are minimum (2.0 μm). The looked up numbers represent widths. If a component with the size property is made of both nMOS and pMOS devices, then the size represents the effective width of the nMOS devices, with the pMOS devices two times that. For example, a "(B)" sized inverter would translate to an nMOS transistor of width 16μm and a pMOS transistor of width 32μm.

Another example is a 3 input "C" sized NAND gate (figure C-2). The effective width of the nMOS transistors are 24μm, and the effective width of the pMOS transistors are 48μm. The pMOS transistors are in parallel, and therefore their widths are equal to the effective width (as there is only one transistor in a worst case path from Vcc to the output). The nMOS transistors are in a series of three, and must then be 3 times as wide to have the same effective width, which comes to 72μm. It is unwise to use large drive versions of NANDs and NORs with many inputs as they consume huge areas and can be very slow due to the additional parasitic capacitances of the transistors.

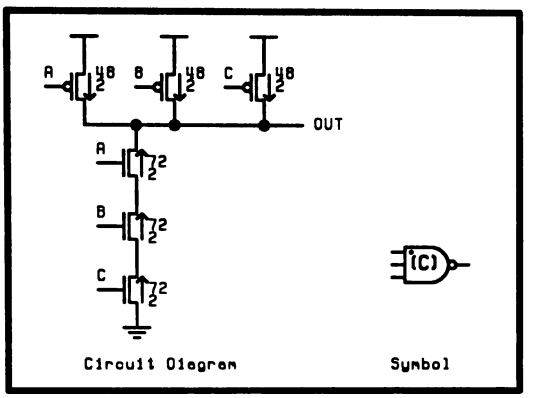

Figure C-2 Three Input C Sized NAND

Some of the symbols of the components have a small circle next to one input (e.g. NANDs and NORs). This is to represent the faster input. In the above example, this would be input A.

If you want to create devices with non-standard transistor sizes and they have size properties, then do the following:

1) select the component and type:

Neted> change text visibility size -hidden

This will hide the size property from the expander. Then place the cursor where you want the widths or lengths to appear and type:

Neted> add property pw desired_pMOS_width
Neted> add property nw desired_nMOS_width
Neted> add property pl desired_pMOS_length
Neted> add property nl desired_nMOS_length

Not all the properties need to be added. The default lengths are 2μm, and the default widths are 8μm. The schematic shown in figure C-2 was drawn using this method.

The sfx properties are handled differently. Instead of looking up the widths in a table at expand time, the circuits are sized within "case" frames on the component schematics themselves. This allows entirely different circuits to represented by the same symbol and is useful for such things as large buffers. To find out what drive strength a given sfx value of a given component has, that component's schematic must be viewed.

To view these schematics to see exactly what each sfx value resolves to, type:

```
$ : /idea/unrlsd/viewsh /user/larry/thesis/lib/component
```

Viewsh is a useful program to simply view the picture file generated by Neted. The commands are limited to scrolling left, right, up, and down via the page scroll keys. Zooming out is done by typing the "z" key, zooming in by typing the "x" key, viewing all by typing the "v" key, and quitting by typing the "q" or return key. It is much faster loading a picture than Neted, but does not allow any edits or selections.

C.4 Automatic Symbol Generation

To save the user tedious editing, and to make uniform symbols for the various schematics, an automatic symbol generator, "gensym" has been added. This program will work only if you have used the port symbols from the CMOS library in your schematic. To run it, you must not be editing the schematic at the same time, so exit Neted or edit another schematic, and at the shell prompt type:

```
$ : gensym schematic_name
```

If you want to add a "comp" property on your symbol for the Cellgraph routines, type:

```
$ : gensym schematic_name -c
```

This is only useful if you are creating new macros for a compiled cell library. For more information on creating a cell library, see the Administrators' Manual, as well as the Cellgraph manuals.

C.5 Simulation Methodologies

First and foremost: use the simulators for what they are designed for. The logic level simulator, **Quicksim is for FUNCTIONAL verification**. It will not tell you if your circuit's timing is within your specifications, except for possibly cycle-to-cycle timing. It may find some potential race conditions, but can not be relied upon to do so. However, it will tell you if your design is logically correct or not.

Circuit simulation is another matter entirely. **Accusim is for TIMING verification.** It should be used to verify that everything in your design is happening within the given timing specifications. It should be used to simulate only a few clock cycle at most. If you are running many clock cycles to verify timing, then you probably need to rethink your simulation methodology. You'll most likely have plenty of time to do so.

When given the task of logically simulating a 32 bit counter, a novice designer might reset the counter at 0 and then count up to 4.29 billion. The designer will then have a long wait in which to think up a better way to do that. What should be done, especially if the counter is implemented as chain of identical elements, is to simulate the reset, count the first 3 or 4 bits and then, if necessary, simulate the rollover condition by initializing the circuit properly (it's in the Quicksim Users Manual).

When doing circuit simulation, it is even more important to be efficient, as Accusim does much more number crunching than Quicksim. If you are only simulating a single path on a schematic (or sets of schematics), extract that path onto a dummy schematic which contains only the path elements and simulate that. It may save you many hours. Large test vectors are not supposed to be used with SPICE level modeling.

When doing any type of simulation, remember that ten minutes of careful thought as to how to properly approach a problem will often save you days or weeks of frustrating simulation and re-simulation!

C.6 Logical Simulation

To expand your schematic for logical simulation (Quicksim), type:

```
$ : lexpand schematic_name
```

Then run Quicksim as you would normally.

C.7 Circuit Sizing

A quick rule-of-thumb circuit sizing methodology is simply to remember that the optimal drive ratio from one stage to the next is about 3:1. To size circuits with this library, start from the output(s) of your circuit and work back. As all the lengths are the same (minimum of 2.0μm), we will neglect them in these calculations as they cancel out.

Sum the widths of the nMOS gates and pMOS gates that a device drives and divide by 9. The letter in the lookup table that corresponds to the smallest standard width that is larger than the result is the optimal drive size of the previous stage.

In the following example (figure C-3), the NOR drives a three input "B" sized NAND (32µm pMOS and 48µm nMOS), a "C" sized inverter (48µm pMOS and 24µm nMOS), and an "A" sized NXFR (8µm nMOS).

$$\frac{\Sigma \text{widths}}{9} = \frac{(32 + 48) + (48 + 24) + 8}{9} = \frac{160}{9} = 17.7$$

The smallest standard number from the lookup table that is greater than 17.7 is 24, or size "C".

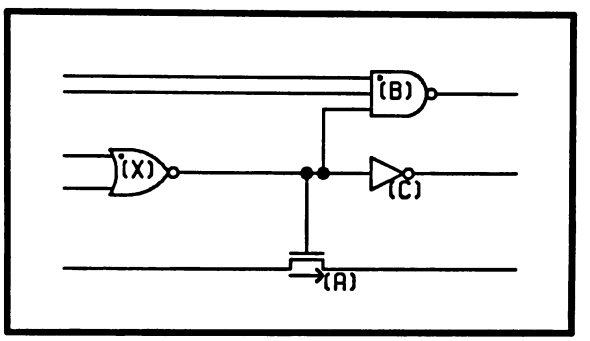

Figure C-3 Circuit Sizing Example

As another quick-rule-of-thumb, if a device drives many other devices, or if you anticipate a large layout net, then you might want to go up one size to try to take into account the layout capacitance. Please note that these sizing rules are only for a first approximation. After you have done circuit simulation (especially after you have back annotated the layout capacitances into the circuit simulations), you can size with more accuracy. A good philosophy to follow is to have the rise and fall times of your signals be roughly within 5 ns. This provides lower power dissipation, and more optimal speeds. A table of what each drive size will drive within 5 ns follows:

Size	Width	Max. Load
A	8µm	0.4pF
B	16	0.8
C	24	1.4
D	32	2.0
E	40	2.6
F	48	3.4
G	64	4.6
H	80	5.4
I	160	11.4

TABLE C-1 Maximum Loading of Drive Sizes

C.8 Physical Simulation

To expand your schematic for physical simulation with Accusim, type:

$: pexpand schematic_name

Then invoke Accusim as you normally would. When you enter Accusim, type:

Accusim> define ground //VSS
Accusim> force dc voltage //VCC 4.3
Accusim> temperature 125
Accusim> read modelfile /user/larry/thesis/lib/cmos2wc

This will set you up for military worst case conditions to simulate your circuit.

C.9 Automatic Placement And Routing of Layout

The Mentor Graphics software package, Cellstation contains many tools that have been integrated with the cell library to automatically place and route your designs. However, the resulting layout routing is generally of very low quality, and often needs to be cleaned up manually. Still, it does save the user many hours of mundane layout editing, so a script has been made to invoke it for you. Simply type:

$: ~/pub/cellstation schematic_name

This will generate the layout of your schematic in the same directory. If you look in your schematic directory, you will now see mask.* files and links of layout as well as the sheet1.* and symbol.* files and links created by Neted and Symed.

C.10 Layout Editing

To invoke Chipgraph, type:

$: chipgraph -st

This startup will automatically load the cmos process for you. Some new features that have been added include the on-line REMEDI design rule checker, and the CMOS cell library. The REMEDI design rule checker is limited as to what it can do and still finds some errors that aren't real. However, it is very useful as it is fast and can run on all nodes with Chipgraph, thus avoiding the long executions time of the Dracula software. It is **strongly recommended** that you use REMEDI to find the "simple" errors, and then do the final detailed checking with Dracula as you normally would.

Entering the CMOS cell library menu and selecting a cell performs just like activating a component in Neted. Simply select a cell and hit the F5 key to instantiate that cell. Note that when you place a cell, it will seem to be a large block of poly and metal1. These are merely the blockage layers added as external perimeters to the cells so that the router will not route over them and cause shorts. To see within the cells, simply select them and peek into them as you would normally.

Several of the cells have more than one place for an input or output connection. This allows more flexibility (and hopefully efficiency) for the router. If a pin appears more than once and has the property "selgroup" (numbered from 1), only one of those pins needs to be connected. To check what properties any item may have in Chipgraph, select it and type:

Chipgraph> status object -property

To get the Dracula LVS software to work, you must add text to your layout. This is done by adding the property "gdsii_text" with the appropriate value. Note that the only valid layers for text are poly.i, metal1.i, and metal2.i shapes. Also note that the origin point of the gdsii_text property MUST lie within the boundaries of the shape. Dracula is very sensitive to text and will discard text if anything is incorrect (e.g. shorted nodes, ambiguous origin points, inappropriate layers). There must be at least one VCC and VSS label. Multiple labels on the same node (of the same value) are acceptable, but you can NOT logically connect two nodes by giving them the same name (as you can in Neted).

C.11 Layout Verification

C.11.1 Design Rule Checking

As stated before, the initial design rule checking is done with the use of the REMEDI on line macros provided within Chipgraph. For more information on the operation of REMEDI and interpretation of its results, please read the REMEDI User's Manual by Jeff Correll.

Design rule checking is done by typing:

$: ~/pub/scn_drc1 cell_name
    to run Dracula DRC on //Naples, or
$: ~/pub/scn_drc2 cell_name
    to run Dracula DRC on //Webster.

When the run is completed, note that the final step created a cell "drc/outcell_name" and any error cells. The outcell_name cell will contain any error cells, so an easy way to view your errors is to instantiate that cell and peek two level down into it within Chipgraph:

```
Chipgraph> edit cell cell_name
Chipgraph> activate cell drc/outcell_name
Chipgraph> instantiate cell 0 0 w0
Chipgraph> toggle select      # selects everything
Chipgraph> peek 2             # peeks two levels down; you may want more
```

C.11.2 Electrical Rule Checking

Electrical rule checking should be done before layout versus schematic checking because it runs much faster and eliminates the simple connection errors, thus saving the user much time. To run ERC, type:

```
$ : ~/pub/scn_erc1 cell_name
        to run Dracula ERC on //Naples, or
$ : ~/pub/scn_erc2 cell_name
        to run Dracula ERC on //Webster.
```

The results file will be in the file erc/ercprt.sum. ERC will flag shorts between labeled nodes (e.g. VCC and VSS), or open circuits between nodes with the same name property. This will avoid a lot of time spent running LVS to find these same errors, as LVS gets very confused by them. ERC also checks for nodes with no paths to VCC nor VSS, transistors with only one source/drain, or other malformed transistors.

C.11.3 Layout Versus Schematic

The following instructions are to be used to run a Layout Vs Schematic check on your cells. It is ONLY for the MOSIS 2.0 μm SCN technology. Make sure you have added all the gdsii_text properties as described above, then perform the following steps:

1) Netlist your schematic:

```
$ : ~/pub/scn_netlist schematic_name
```

This only needs to be done if the schematic has been edited since the last time it was netlisted.

2) Compile your netlist. This also links in a macro library so that LVS will recognize some simple macros like NANDSs, NORs, CXFRs, etc. This only needs to be done if the netlist is not up to date (i.e. if it has been edited or netlisted since the last compilation).

```
$ : ~/pub/scn_log1 schematic_name
        to run it on naples, or
$ : ~/pub/scn_log2 schematic_name
        to run it on webster.
```

3) Run LVS.

```
$ : ~/pub/scn_lvs1 cell_name schematic_name
                or
$ : ~/pub/scn_lvs2 cell_name schematic_name
```

4) LVS will generate error cells (if necessary) which are useful in finding where it failed to find correspondence between your schematic and layout. Personally, I have only found these to tell me the rough area LVS got lost. The important file to read is "lvs/lvsprt.lvs". This has the discrepancy list in it.

One helpful hint in reading the report is to look for the discrepancies with question marks. These are the things that LVS couldn't match. Again, netlisting and compiling (scn_netlist and scn_log) only need to be rerun if there is a change in the schematic.

C.11.4 An Example of an LVS Run

As an example of how to interpret the LVS discrepancy list, a test cell was made (figure C-4). The cell was run through the cellstation script, and had the gdsii_text properties added (figure C-5). The cell was then corrupted by deleting a poly input to the second from the right NAND gate and run through LVS (figure C-6).

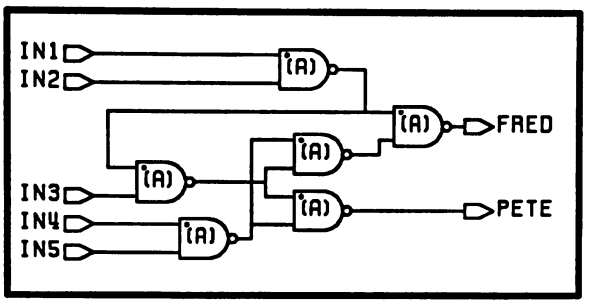

Figure C-4 Example Network

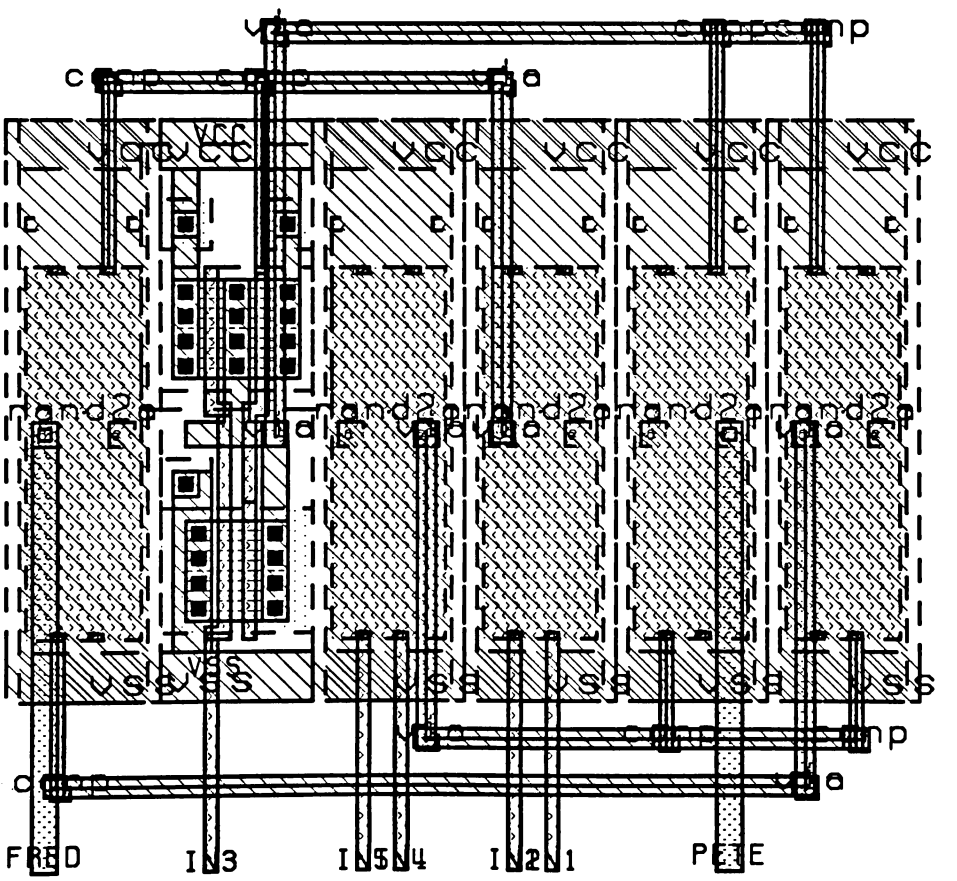

Figure C-5 Example Auto Placed and Routed

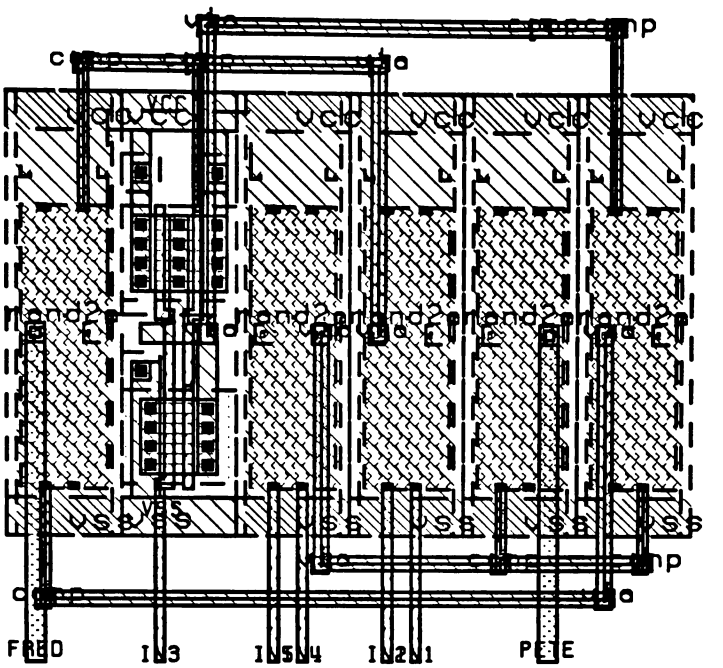

Figure C-6 Example With Missing Line

The following is a reduced copy of the LVS report resulting from comparing the schematic to the faulty layout:

```
  *********** LVSNET SUMMARY REPORT ***********
                        DATE : 23-JUL-91
                        TIME : 13:59:45
                        ***** LVS CHECK *****
                        PRINTLINE       =     1000
                        WPERCENT(MOS)   =    5.000 %
                        LPERCENT(MOS)   =    5.000 %
1         ************************************************
          *********   CORRESPONDENCE NODE  PAIRS   *********
          ************************************************
            SCHEMATICS                  LAYOUT          PAD TYPE
     VCC                  43     VCC                     8   P
     VSS                  44     VSS                     7   G
     FRED                 29     FRED                    1   I
     IN1                  31     IN1                     13  I
     IN2                  33     IN2                     12  I
     IN3                  35     IN3                     9   I
     IN4                  37     IN4                     11  I
     IN5                  39     IN5                     10  I
     PETE                 41     PETE                    2   I
```

```
  NUMBER OF LAY. PADS READ =          9   DISCARDED =     0
  NUMBER OF SCH. PADS READ =          9   DISCARDED =     0
  NUMBER OF VALID CORRESPONDENCE NODE PAIRS =           7

1         ************************************************
          ***********  LVS DEVICE MATCH SUMMARY ***********
          ************************************************

     NUMBER OF UN-MATCHED SCHEMATICS DEVICES       =    0
     NUMBER OF UN-MATCHED LAYOUT       DEVICES     =    0
     NUMBER OF    MATCHED SCHEMATICS DEVICES       =    6
     NUMBER OF    MATCHED LAYOUT DEVICES           =    6

1         ************************************************
          ********  DISCREPANCY POINTS  LISTING  **********
          ************************************************

 ************************ DISCREPANCY  1 *********************

    ----------MATCHED DEVICE UN-MATCHED NODE---------

 OCCURRENCE NAME  I_7

  DEV56     NAND             :  DEV26  NAND   X=103   Y=52
  PETE, N_15, N_63              PETE, ?14, N_63

          TOTAL          1 DISCREPANCY POINTS REPORTED

1         ************************************************
          **********  DISCREPANCY POINTS  SUMMARY  *********
          ************************************************

                    1 MATCHED DEVICE TO UN-MATCHED NODE
```

The discrepancy list is split into two columns. The left column is schematic information and the right column is layout information. Note the "?14" in the layout column of discrepancy entry. When Dracula LVS can not figure out what a node is, it assigns a number. The question mark further indicates this. A useful set of commands to view this area on your layout is:

```
Chipgraph> view center 103 52    #the x and y coordinates from above
Chipgraph> mark 103 52 w0
```

This will center the screen on the error spot and place the curser on it. It is typically the lower left corner of a MOS gate.

C.12 The Component Library

As of this writing, the component library is not complete, nor are all the layout cells completed for all the existing components. Many components have sizes that will not be supported as they are unnecessary or unrealistic. For example, a d-flip flop takes up the same amount of area for "A", "B", and "C" values of its sfx property, and has identical internal timing, so the "A" and "B" cells would be superfluous. Another example would be a "(E)" sized 8-input NAND gate. It's nMOS transistors would all be 320μm wide!

The current list of components and what they are follow:

Component	Description
and2	two input AND
and3	three input AND
and4	four input AND
and5	five input AND
buf	2 inverter buffer
cmux2	2 input cMOS gate MUX with internal decode
cmux2nd	2 input cMOS gate MUX without internal decode
cmux3nd	3 input cMOS gate MUX without internal decode
cmux4	4 input cMOS gate MUX with internal decode
cmux4nd	4 input cMOS gate MUX without internal decode
cmux5nd	5 input cMOS gate MUX without internal decode
crxfr	resistive cMOS transmission gate
cxfr	cMOS transmission gate
dff	d flip flop
dffar	d flip flop with asynchronous reset
dffarcf	d flip flop with async reset active falling clock edge
dffarcr	d flip flop with async reset active rising clock edge
dffarscr	d flip flop with async reset and set active rising clock edge
dffas	d flip flop with async set

dffascf	d flip flop with async set active falling clock edge
dffascr	d flip flop with async set active rising clock edge
dffcf	d flip flop active falling clock edge
dffcr	d flip flop active rising clock edge
dffqarcr	d flip flop with q* output async reset active rising clock edge
gnd_pad	Ground (Vss) power pad
ietri	tri-statable inverter
ilat	inverting latch
in_pad	input pad
inv	inverter
inv_inv	inverter with weak feedback inverter
inv_pch	inverter with weak feedback pull up
invr	resistive inverter
io_pad	io pad
keeper	inverter with weak feedback inverter (one terminal)
nand	2 to 8 input NAND gates
ndlat	latched 2 input NAND gates
netcon	used to attach two nets with different names
nmux2	2 input nMOS gate MUX with internal decode
nmux2nd	2 input nMOS gate MUX without internal decode
nmux3nd	3 input nMOS gate MUX without internal decode
nmux4	4 input nMOS gate MUX with internal decode
nmux4nd	4 input nMOS gate MUX without internal decode
nmux5nd	5 input nMOS gate MUX without internal decode
nor	2 to 6 input NOR gates
nrlat	latched 2 input NOR gates
nrxfr	resistive nMOS transmission gate
nxfr	nMOS transmission gate
or2	2 input OR
or3	3 input OR
or4	4 input OR
out_pad	tri-statable output pad
ph1	ph1 global label
ph2	ph2 global label
ports	input, output, and io ports
prxfr	resistive pMOS transmission gate
pxfr	pMOS transmission gate
rip	bus rippers
tribuf	tri-statable buffer (large)
tsg1	enabled inverter (enables on the inside)
tsg2	enabled inverter (enables on the outside)
vcc	Vcc global label
vcc_pad	Vcc power pad
vss	Vss global label
xnor	XNOR
xor	XOR

Rochester Institute Of Technology's

CMOS Standard Cell Library

Administrator's Manual

Jeffrey Correll
Christopher Insalaco
Shishir Ghate
Lawrence Rubin

July 30, 1991

D.1 Overview

This manual is intended for the administrators of the RIT CMOS Standard Cell Library. It contains instructions for the maintenance and further development of the library. The author assumes that the reader has a full understanding of the Mentor Graphics CAD platform and is familiar with the CMOS Standard Cell Library User's Manual.

The two central ideas to the library are the scalable design rules and the parameterizing of the components and cells. The scalable design rules have been supplied by MOSIS, which currently supports 1.2, 1.6, and 2.0 micron N-well CMOS processes. The components are parameterized alphabetically. Each letter corresponds to a numeric value in a lookup table located in single text file that is read by Expand, Mentor Graphics' netlister. Thus, by editing that single file, the schematics can automatically represent a different feature size technology without any schematic editing. This should allow easy upgrades to future fabrication processes supported by MOSIS with a minimum of re-work.

The overall objective is to create a complete, fully integrated, and flexible environment for the design, layout, and verification of CMOS VLSI microchips, using a scalable design rule set. Specific objectives are:

1) To create a component library. Each component will be a primitive that can be simulated both logically and electrically. The components will be designed in such a way as to ease future scaling or change in the design rules.

2) To create a cell library. Each cell will map to a specific size of a corresponding component. The design rules will be those of MOSIS' scalable N-well cmos (initially drawn for the 2.0μm or λ=1μm scale).

3) Intimately integrate the component and cell libraries into the Mentor Graphics platform, including their schematic capture, automatic place-and-route, layout editing, simulation, and layout verification tools. The cell layout will also support Cadence's Dracula II layout verification tools.

4) To allow the fabrication of chips via the MOSIS program.

D.1.1 The Component Library

For logical or circuit simulation, the designer enters the schematic using elements from the component library. These components have an advantage over those from the generic libraries that come with the Mentor Graphics package, as they support both logical and circuit simulation. Using only the Mentor Graphics package, the designer would have to enter a logical and a physical schematic, thus introducing a certain element of unnecessary human error into the design flow.

The designer can change a components size by simply editing a text property attached to the component on the schematic. There are two text properties used, size and sfx. The

size property is used by Expand as a lookup variable, whereas the sfx property is used by case frames within the schematic of a component. These two properties are exclusive; a component may have an sfx property, or a size property, but not both. The component library builds on itself, in that except for the basic transistors, each component has a representative symbol (that appears on the schematic) and a schematic of its own.

The component library exists in the directory /user/larry/thesis/lib.

D.1.2 The Cell Library

The cell library consists of all the implemented sizes of all the components. The convention for naming a cell is the component name followed by the size or sfx parameter (which is alphabetic). Thus the "b" sized two input NAND gate is simply called "nand2b". Each cell is designed to work with the automatic place-and-route tools from Mentor Graphics. For a new component to be added, a schematic and symbol for it must be created in the library directory and the specially written component menu for Neted should be edited to allow the users easy access to it. For a new cell to be added, it must be drawn with Chipgraph and the library must be re-compiled for the place-and-route tools. For information as to how the library is compiled, please see section D.4.5, "Compiling the Cell Library".

The cell library exists in the directory /user/larry/thesis/cells, the process definition files reside in /user/larry/thesis/pdfs, and the compiled library files used by Cellgraph (Mentor's place-and-route tool), exist in /user/larry/thesis/tecp2m2. For more information on the above commands and the files they operate on, please see the Semi-custom Technology and Design Formats manual, the Cell Station Users Manual, and the Cell Station Reference Manual, supplied by Mentor Graphics.

D.1.3 Support Software

The component library is designed to support Mentor's Quicksim and Accusim implicitly. The cell library is designed to support Cadence's Dracula II layout verification tools, Mentor's REMEDI design rule checker, and Mentor's (formerly Texas Instruments') Checkmate layout verification tools.

The support software for the Dracula II software exists in /user/pub and consists of:

scn_drc1, scn_drc2:	Design Rule Checking script
scn_erc1, scn_erc2:	Electrical Rule Checking script
scn_netlist:	Calls the ECAD netlister in /user/larry/thesis/ecad
scn_log1, scn_log2:	Compiles the netlist
scn_lvs1, scn_lvs2:	Layout Versus Schematic checking script
scn_lpe1, scn_lpe2:	Layout Parameter Extraction script

A "1" after a script name indicates that it will run on //naples and a "2" indicates that it

will run on //webster. Only one user may use either of the licenses at a time, so two semaphores have been implemented. The actual rule decks that are called by these scripts are located in /user/pub/design_checking.

REMEDI, Mentor's on-line design rule checker, and the Chipgraph menus and macros are located in /user/larry/thesis/chipgraph. The design rules themselves are contained within the process definition file (PDF), mosisproc_scn.pdf. This file and other PDFs are found in /user/larry/thesis/pdfs. The PDFs are compiled by "do"ing them within Chipgraph, and then writing the process to a file with the ".bin" extension (i.e. mosisproc_scn.bin).

Menus and macros for Neted are located in /user/larry/thesis/neted. The files lexpand and pexpand are also located there, with pointers in all the /com directories to them so that the users will have them in their search rules.

Another link in all the /com directories points to the gensym script in the neted directory. As stated in the CMOS Standard Cell Library User's Manual, gensym automatically generates symbols for given schematics. The source for gensym exists in /user/larry/src/gensym.

D.2 The Expand Files: lexpand and pexpand

The files lexpand and pexpand are scripts to run Expand in batch mode for logical and physical simulation, respectively. They contain the lookup tables for the size property, as well as several other parameters. The lookup sizes are not used for logical expansion, but are added anyway to prevent unnecessary warnings. Other parameters include:

```
par vplus  vcc      name of positive power node
par vminus vss      name of ground power node
par physical 1      switch for logical or physical case frames
par logical  0        "     "     "     "     "     "     "
par prise 1         default rise time (used by logical simulation only)
par pfall 1         default fall time (used by logical simulation only)

par a      8        The physical lookup table for the size property.
par b      16         These are the effective widths of the nMOS devices
par c      24         within a component. (See the section on circuit sizing
par d      32         in the CMOS Standard Cell Library User's Manual).
par e      40
par f      48
par g      64
par h      80
par i      160
par x      0
par nsub   vss      the base of the nMOS transistors
par psub   vcc      the base of the pMOS transistors
par nw     8        the default nMOS width in microns
```

```
par pw      16    the default pMOS width in microns
par nl       2    the default nMOS length in microns
par pl       2    the default pMOS length in microns
par al       4    the default length of the source/drain region
par pscale   2    the default nMOS width : pMOS width ratio
par pdrive sss    the default logical drive strength
```

Other primitive and property declarations are made. For more information on what these are, consult the Expand User's Manual and the Expand Reference Manual provided by Mentor Graphics.

D.3 Adding a Component

Components are added in /user/larry/thesis/lib. Special care must be taken when creating a library schematic and especially the symbol. **Remember, once released, every time you edit the symbol, EVERYONE who uses that symbol in ANY of their schematics must update those schematics, so be VERY careful about editing the symbols.**

D.3.1 Creating the Schematic

The schematic is created normally, with the pathname field of the title block given the value of "LIB". If you create a component that is sized via the sfx property, then you must make "if" frames with all the supported values of sfx, and you must also include an "X" value (unsized) frame.

If you make a component that is sized via the "size" parameter, then you must draw the schematic with nxfr4s and pxfr4s ONLY (four terminal nMOS and pMOS transistors), giving them the properties nl, nw, pl, and pw the values "(NL)", "(NW)", "(PL)", and "(PW)", respectively. The substrates should have the global properties of "(PSUB)" and "(NSUB)". The symbol that is used is the global label. As an example, please see figure 3-1, a panel from the inverter schematic. The instance properties of P and N have been added to avoid the automatic instance values such as "I$31", which tells the designer nothing.

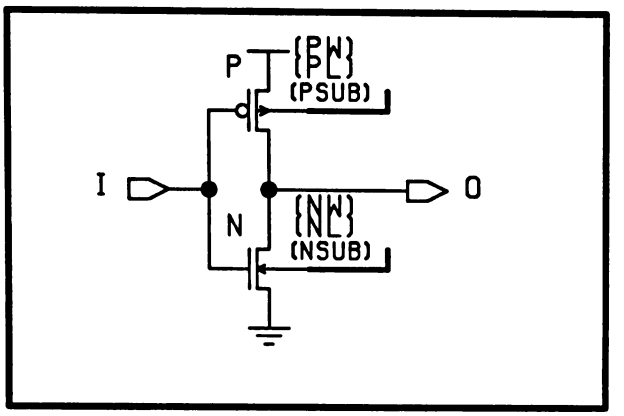

Figure D-1 Inverter Schematic

D.3.2 Creating the Symbol

For a symbol of an size component, the following properties (except for primg) and their flags must be used (the inverter symbol was used for this example):

Property	Value		Flags	
Model	INV	fixed	nohidden	noinstance
primg	INV	fixed	nohidden	noinstance
comp	('INV_'&SIZE)	nofixed	nohidden	noinstance
nsub	(NSUB)	nofixed	nohidden	noinstance
psub	(PSUB)	nofixed	nohidden	noinstance
nw	(SIZE)	nofixed	nohidden	noinstance
nl	(NL)	nofixed	nohidden	noinstance
pw	(PW)	nofixed	nohidden	noinstance
pl	(SIZE*PSCALE)	nofixed	nohidden	noinstance
vminus	(VMINUS)	nofixed	nohidden	noinstance
vplus	(VPLUS)	nofixed	nohidden	noinstance
size	(X)	nofixed	nohidden	instance
pdrive	SSS	nofixed	nohidden	noinstance

The primg property is used to tell Expand to use one of Quicksim's embedded primitives (if expanding for logical simulation). This is far more efficient than simulating even a switch level transistor design. For more information on what primitives are available, see Mentor's Quicksim Reference Manual. Obviously, if your component is not one of the Quicksim supported primitives, then you should not use the primg property.

Most of the parameters are looked up within the lexpand or pexpand scripts. Note how the pMOS width is calculated. The "size" parameter is multiplied by a scaling constant (also defined in the expand scripts). Many mathematical equations may be used to implement maximum flexibility of the design.

The comp property is used by Cellgraph (the auto-place-and-route tool). It will be used to match the schematic with the properly sized cell. This will take the string "INV_" and add the size value (for example an "A" size) and concatenate them to "INV_8", which corresponds to the cell name "inva" in the cell technology file (to be discussed later).

Some symbols offer the user more than one representation. This is useful for mixed logical representation, and is done by using a parameter "bub" and a "case" frame within the symbol. For example, see figure 3-2 for the symbol of the inverter.

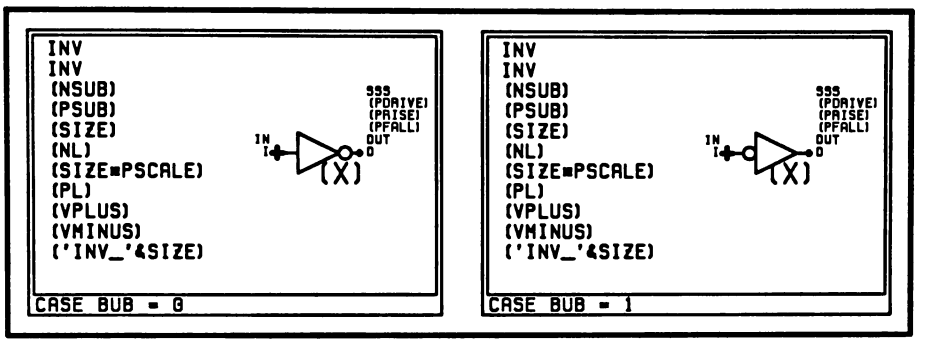

Figure D-2 Inverter Symbol

Symbols for components using the sfx property need only have the following properties attached to their bodies (the dff symbol was used for this example):

Property	Value		Flags	
inst	DFF	nofixed	nohidden	instance
comp	('dff'& sfx)	fixed	nohidden	noinstance
name	DFF	fixed	nohidden	instance
primg	MACRO	fixed	hidden	instance
sfx	X	nofixed	nohidden	instance

All pins of all symbols should have a "pintype" property of "IN", "OUT", or "IO".

D.3.3 Updating the Neted Menu

To add a component to the Neted menu (or a macro for that matter), you need to edit the file /user/larry/thesis/neted/menu. It should be pretty obvious what you need to add. The "Define menu" command creates a new sub-menu, and the "Define Item" command creates an item within a sub-menu. For more information on how to write macros or menu items, consult Mentor's Human Interface Manual. It should be noted that this is not going to be compatible with Mentor's 8.0 software and must be converted if and when RIT updates to that software.

After you have edited the menu file, type (within the /user/larry/thesis/neted directory):

$: update

This will automatically compile the new Neted menu, so that anyone using the startup afterwards will use the new menu.

D.3.4 Netlisting the Library

Every size or sfx value of every component (including X) must be netlisted. An efficient way to do this is to create a number of "all_" schematics and then netlist them. For example, all the inverters and buffers are in a schematic called "all_invs_and_bufs". Each component is instantiated a number of times so that every supported value of size and sfx is represented. The meta-schematic is then netlisted with the "-lib" option:

$: ~/pub/scn_netlist all_invs_and_bufs -lib

There will be a number of warnings concerning zero sizes (the X values). These are to be ignored. By including the unknown sizes, the Dracula II software can run unsized checks (which run faster, but our scripts are not currently written to do this).

D.4 Adding a Cell

Cells should be added in /user/larry/thesis/cells. Obviously, they must all use the mosisproc_scn.bin process definition file (process "mosis_scn"). The pin names must match those of the schematic exactly. It should go without saying (but I'll say it anyway) that all cells must be DRC, ERC and LVS clean before they are added to the library.

D.4.1 Cell Specifications

The following are a list of rules that must be followed in order to make a cell compatible with the rest of the library. Most of these rules are needed to make the edges of one cell compatible with the edges of any other cell that could be placed next to it.

1. All cells will have an 8μm wide VCC line on top and an 8μm wide VSS line on the bottom. The top of the VCC line to the bottom of the VSS line is 94μm.

2. Any metal2 must be at least 2μm from the left and right edges of the cell.

3. Any metal1 (except for the VCC and VSS lines, which must go to the left and right edges) must be at least 2μm from the left and right edges of the cell.

4. Any poly must be at least 1μm from the left and right edges of the cell.

5. Any active_area must be at least 2μm from the left and right edges of the cell.

6. The n_implant and p_implant layers should go up to the left and right edges of the cell.

7. The N-well will go from 47μm from the top of the VCC line up to at least the top of the VCC line (it may go over), and will go to at least the left and right edges of the cell (it may go over).

8. The cell is to be surrounded by a perimeter of layer cell_outline.e (excluding the N-well; it may overlap into other cells), and have a point of cell_txt.e in the center with the property cell_name attached to it. The easiest way to do this is to copy it from another cell.

9. The origin of the cell should be the lower left corner of the VSS line.

10. All polygons are drawn on a 1μm grid. The router may, however, route to a 0.5μm grid.

11. All input and outputs have non-overlapping, strait line paths out the top and/or bottom of the cell.

12. All inputs and outputs are drawn as poly.e or metal2.e ports. Some input or outputs may have multiple ports to allow for more efficient routing. These ports have the property "selgroup" added to them with incrementing integers as values.

13. All cells have blockage layers. These are metal1.e, metal2.e and poly.e perimeters that are drawn around all objects of their corresponding internal layers that the router may not route over.

D.4.2 Of Ports and Pins

The input and output signals should all have ports to go to/from. These objects should be of type port, and have a "pin" property with the value of the signal name. If you need the signal to come in to the cell at more than one place, simply copy the port, and the router will route to all instances of that port.

However, if you want to allow the router freedom as to which direction the route may come from, you may make multiple ports with the same pin property, but add a "selgroup" property with a numerical value (from 1). In the following example, note that I1 and I2 both have two choices of ports, with the top ports having the property selgroup "1", and the bottom ports having the property selgroup "2", thus allowing the router to route from above or below. The output port "O" also gives the router the option of routing from the left or right, whichever it finds is more efficient.

Power routing is done by adding a port over the VCC and VSS lines with the property POWER_PIN, with the value of "VCC" and "VSS", respectively. See figure 4-1 for an example of selgroup and power_pin properties.

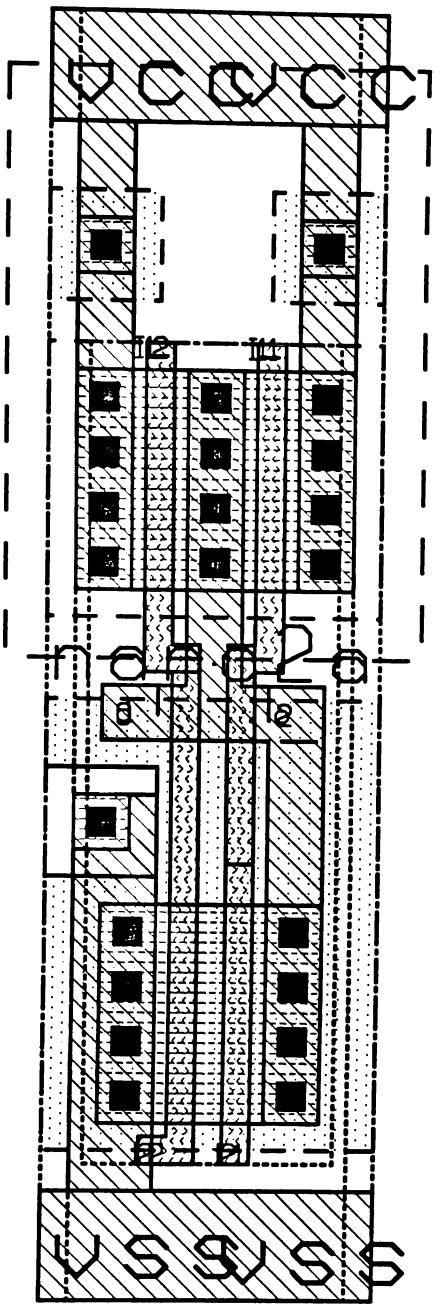

Figure D-3 2 Input NAND Cell

Legal port layers are poly.e, metal1.e, and metal2.e. If you use metal1.e, the router will have to draw a metal2 line over your cell and will be smart enough to add a via automatically. If you use metal2.e (i.e. your port is metal2), you must supply the internal via yourself.

D.4.3 Boundary Layers

Boundary layers (external object type perimeter) are used as blockage layers for the router. Other than aesthetics, they serve no other purpose. When routing, Cellgraph only considers one layer at a time. Thus, when it is routing poly, it only looks at the poly layer spacing and width rules. This makes it much faster, but quite a bit dumber. To compensate, you must add a blockage layer to indicate where poly can not be routed. Failure to do so will allow the router to route directly over your cell creating extra devices and shorting nodes. The router will treat the blockage layer as any other piece of poly (i.e. it will not route within 2μm of it).

For example, if there is a well tie with a 4μm square active_area.i with a 2μm square contact_a.i in its center, the closest a poly line could be routed would be 1μm away from the active_area.i. Because the poly to poly spacing rule is 2μm, you must have a poly.e perimeter 1μm inside the active_area.i (shadowing the contact_a.i). Thus the router will route 1μm (or more) away from the active area. Figure 4-2 is an example of a poly.e blockage layer, that might occur at the top of an inverter cell.

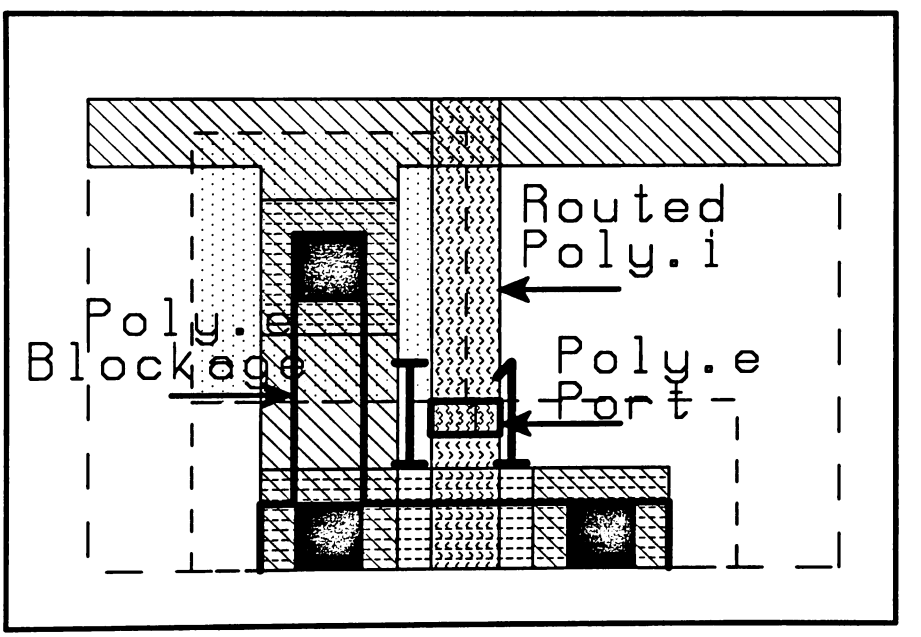

Figure D-4 Poly Blockage Example

Ports may be within boundary layers only if they are touching the edge. If you do not give the router enough room to route, then it won't (this is called an "overflow" by Cellgraph).

D.4.4 Updating the Chipgraph Menu

To add a cell to the Chipgraph menu (or a macro for that matter), you need to edit the file /user/larry/thesis/chipgraph/menu. It should be pretty obvious what you need to add. The "Define menu" command creates a new sub-menu, and the "Define Item" command creates an item within a sub-menu. For more information on how to write macros or menu items, consult Mentor's Human Interface Manual. It should be noted that this is not going to be compatible with Mentor's 8.0 software version and must be converted if and when RIT changes over to that software.

After you have edited the menu file, type (within the /user/larry/thesis/chipgraph directory):

$: update

This will automatically compile the new chipgraph menu, so that anyone using the startup afterwards will use the new menu.

D.4.5 Compiling the Cell Library

To compile the cell library, the following steps must be performed.

D.4.5.1 Adding A Newly Created Cell To The File "cell_list"

The file "cell_list" is located in the directory /user/larry/thesis/tech. There are three lines of information that must be added, they are as follows:

DEFINE NAME=<physical_name> LOGNAME=<logical_name> &
CLASS=<class_type> CELLNAME=((<physical_name>, 0.0, 0.0, N)) &
PLACETYP=<place_type> GEOMCELL=NO MODEL=<model> GROUP=0

Below are the description of the fields enclosed in the brackets. Words that are not enclosed in the brackets should be entered as is.

physical_name: This should be the name of the layout that was created.

logical_name: This is the logical name of the file. In most cases it is the same as the physical name. The difference occurs when the cell is going to have a size property associated to it. In this case the logical name will be the physical name with an '_' and number corresponding to the 'a', 'b', 'c', ... which is located at the end of the physical name. Please refer to the example below.

class_type: This tells Cellstation whether this is an internal or external cell. The two words that should be used are 'INT' if the cell is an

internal cell or 'EXT' if the cell is an external cell. An external cell is a cell that leads to the outside world. Basically these are input/output pads. Not entering one of these codes, or the wrong code, could cause Cellstation to incorrectly place the cells.

place_type: This piece of information is used by Cellstation when it comes time to auto-place the standard cells for a design. If the cell you are entering is a internal cell, then this value should be a '1'. If the cell you are entering is an external cell, then this value should be a '2'. If any other values are entered, Cellstation could place the cells incorrectly.

model: This field is the model type that Cellstation should use when extracting the information from the cell. There are only two possible values for this field. They are INTMODEL for an internal cell and PORTMODEL for an external cell. Please make sure that these values correspond to the place_type and class_type fields.

Below are examples of actual cells that have been defined in the file "cell_list".

This is an example of an A size two input AND. Notice how the logical_name has a '_8' following the word AND2 instead of an A. This is because this cell has a size property in the schematic and not a sfx property. The 8 comes from the lookup table for an A size component.

```
DEFINE NAME=and2a LOGNAME=and2_8 &
        CLASS=INT CELLNAME=((and2a, 0.0, 0.0 ,  N)) &
        PLACETYP=1 GEOMCELL=NO MODEL=INTMODEL GROUP=0
```

The next example is a two input A size mux. Here you can see that the logical_name is the same as the physical name. This occurs because this cell posses the SFX property in the schematic instead of the SIZE property. The rest of the information is very similar to the information above.

```
DEFINE NAME=nmux2a LOGNAME=nmux2a &
        CLASS=INT CELLNAME=((nmux2a, 0.0, 0.0 ,  N)) &
        PLACETYP=1 GEOMCELL=NO MODEL=INTMODEL GROUP=0
```

The last DEFINE statement we will look at is for an external cell. The cell shown below is the I/O pad used in all the designs. Notice how the CLASS, PLACETYP, and MODEL are different than the two examples above.

```
DEFINE NAME=io_pad LOGNAME=io_pad &
        CLASS=EXT CELLNAME=((io_pad, 0.0, 0.0 ,  N)) &
        PLACETYP=2 GEOMCELL=NO MODEL=PORTMODEL GROUP=0
```

D.4.5.2 Running BUILD_LIB

The next step is to run the program BUILD_LIB. This should be done in the directory /user/larry/thesis. To run this program, enter the following command at the prompt:

$: build_lib tech -tdf

As this program is running, it will list all the cells it is compiling and what model it is using for each of these cells. If, for some reason, you notice a mistake as the program is running, wait until it has finished executing and then modify the "cell_list" file. Once you have done that, you can run this program over again. Once this program is finished, you will have a file called "block.tdf" located in the TECH directory. You can, if you wish, page through this file, but **DO NOT** change any of the information found there.

D.4.5.3 Compiling The Technical Definition File

The last step is to compile the "block.tdf" file. Again you must be in the directory /user/larry/thesis. The compile command is as follows:

$: tdf_block_input tech

Once the compilation has worked successfully, you will be returned to the shell prompt. At this point you have completed all the necessary steps to adding a cell and can now use this cell in your schematic, and have Cellgraph automatically place-and-route it.

D.5 Dracula II Layout Verification Tools

The Dracula II tools are only licensed to run on two nodes, naples and webster. Each of these nodes may be used by only one user at a time, and thus a semaphore has been added to the scripts that call the Dracula II software. If a user quits out of a script prematurely (usually by hitting control-Q) the semaphore file (/user_temp/drc_semaphore) may not be deleted, and thus will lock out that node. The solution is to simply delete this file, after making sure that the person who was using it (read the file) is no longer running the Dracula II script.

D.5.1 Design Rule Checking

The actual design rule deck is in /user/pub/design_checking/scn_drc.com. It contains the design rules (in microns) for MOSIS' 2.0μm N-well CMOS process. Great care must be taken if you adjust ANYTHING in this file, as many things are inter-related and by attempting to make one rule more efficient, it is easy to break many other rule checks.

Concerning the contact rules, we have chosen to implement the dense rules, instead of the simple rule set. As the names imply, this gives us denser layout at the cost of having a more complicated rule set.

In the future, if support is planned for the 1.6μm or 1.2μm processes, or any other MOSIS scalable N-well CMOS process created after this writing, a new DRC deck must be made. It should be simply a copy with all the new measurements being a constant multiplication of the current numbers.

D.5.2 Electrical Rule Checking

Before you can run ERC, you must do three things. The first is to add a GDSII_TEXT property labeled VCC to a part of the design which is the actual VCC. The second thing that must be done is to add another GDSII_TEXT property labeled VSS to a part of the design which is the ground. The third is to add the necessary GDSII_TEXT properties labeling the proper inputs and outputs. Please refer to the LVS section on how to add these properties. These are necessary in order for ERC to work properly. Once this is done, you are ready to run ERC.

To run the Electrical Rule Checker, enter one of the following commands at the prompt:

```
$ : ~/pub/scn_erc1 <design_name>
                or
$ : ~/pub/scn_erc2 <design_name>
```

The design_name corresponds to your particular chipgraph layout.

After ERC has finished its run, you might end up with various error cells. All these error cells can be brought into chipgraph in the same way as DRC cells are brought in. As a note, some of the errors that are produced by ERC might not have to be taken into consideration, but most of them are correct. A description of what each of the error cells mean are given below.

MULLB50 :	A short exists between two different labels.
SAMLB50 :	An open exists between two identical labels.
NOVCC50 :	A node has no path to VCC.
NOVSS50 :	A nodes has no path to VSS.
NOPRGR50 :	A nodes has no path to VCC nor VSS.
NOALL50 :	A node has no path to VCC, VSS nor other texted pads.
NSDGT250, PSDGT250 :	A list of transistors with source/drain count > 2.
NSDEQ050, PSDEQ050 :	A list of transistors with source/drain count = 0.
FLOAT50 :	A nodes exists with no devices connected.
NPRGR50, PPRGR50 :	Transistors that are connected directly across VCC and VSS.
GATVCC50 :	pMOS transistors with gates connected to VDD (OFF!)
GATVSS50 :	nMOS transistors with gates connected to VSS (OFF!)
PDFVSS50 :	pMOS transistors with a source/drain connected to VSS.
NDFVCC50 :	nMOS transistors with a source/drain connected to VCC.
ONDEV50 :	Nodes with only once device connected.
NDEPL50:	nMOS transistors in a depletion mode configuration.
PDEPL50 :	pMOS transistors in a depletion mode configuration.
MIXUP50 :	Transistors with their source, drain, gate shorted.
WLNOVC50:	Any N-well not connected to VCC.

If you want to change anything in the original script, it is located in ~/pub/design_checking, and the file name is scn_erc.com. Inside this file are all the checks that produce the error files above. Please refer to page 11-38 in volume two of the Cadence Dracula II Manuals for a further description as to what and where these checks came from. If further information is necessary, one will have to refer to the ERC commands section on pages 5-94 through 5-112 in volume one of the Cadence Dracula II manuals.

D.5.3 Layout Versus Schematic

The setup stages for LVS consist of netlisting the schematic and compiling the netlist. The netlister takes the user's schematic and creates a SILOS format netlist file, which is then compiled by the LOGLVS program to be used by LVS directly. The Dracula II deck for LVS is the file /user/pub/design_checking/scn_lvs.com.

D.5.3.1 The ECAD Netlister

The ECAD netlister exists in /user/larry/thesis/ecad, along with all its source files. It is a rather efficient, hierarchical netlister. It creates macro files within each directory of each schematic of the design hierarchy called "ecad_mac". The final file it generates in the directory of the top level schematic is called "ecad.net". This is the final netlist file that is compiled for use by Dracula II.

For the library components, the netlister creates a different macro file for each size or sfx property (i.e. "ecad_mac.a", "ecad_mac.b", etc.). Each library component should have macro files for all the supported sizes, as well as an unsized macro (which is simply called "ecad_mac"). Netlisting the library is discussed in section 3.4.

The netlister has a library list file which contains a list of directories that are considered libraries. If a schematic lies within one of these directories, it will be treated as a library component. This file is /user/larry/thesis/ecad/libraries and is read at runtime, so no re-compilation of the netlister is necessary.

Another file, called "circuit.lib", in the /user/larry/thesis/ecad directory is for the Dracula II netlist compiler, LOGLVS. It contains a SPICE syntax file of the library primitives. Note that not all primitives are necessary. This file should probably not have to be altered in any way, as there are no sizes hardcoded in it, and all the library components are built from a small primitive set which is fully represented within this file.

In the future, if support is planned for the 1.6μm or 1.2μm processes, or any other MOSIS scalable N-well CMOS process created after this writing, an expand file will have to be created in /user/larry/thesis/ecad, and the netlister will have to be called appropriately. Below is a copy of the help manual page for ecadlist (~/pub/scn_netlist is just a script that calls ecadlist).

FORMAT

ECADLIST top_component [options]

ECADLIST netlists a macro or a whole design tree for ECAD. Previous macro netlists are reused if they are still current.

ARGUMENTS

top_component The file system pathname of the top component's directory.

OPTIONS

-USAGE Print command usage and abort.

-HELP Print command help and abort.

-M Only netlist top_component, don't travel down the hierarchy netlisting the macros this macro calls.

Default if omitted: Netlist the whole design tree.

-F | -FORCE Force all macros to renetlist even if they have an old netlist that could have been reused. Note that cmos lib components will not be renetlisted unless the -LIB switch is also specified. (Most users do not have the rights to renetlist cmos lib components.)

Default if omitted: Only netlist macros that have been edited, netlisted previously with errors, or have never been netlisted.

-E expand_file Use the specified file as an extra command file for expand. This file will be used immediately after the default expand file and can introduce new parameters, redefine parameters, forget parameters, or substitute properties.

Default if omitted: Use only the default expand file:
/user/larry/thesis/ecad/expand

-LIB Allows components in libraries to be renetlisted. Since most users don't have write rights in the libraries, they should not use this switch.

Default if omitted: Include current definitions of library components. If a component is out of date, print a warning message, include the out of date definition, and continue.

-LIBFILE lib_file Use the specified file to define what directories are libraries.

Default if omitted: Use the default library definition file:
/user/larry/thesis/ecad/libraries

-H header_file Include header_file at the start of the netlist output. This can be used to include externally defined components, set up output tables, etc.

-AF d Force all components that were last netlisted after date/time "d" to renetlist, even if they wouldn't otherwise have needed renetlisted.

-BE d Force all components that were last netlisted before date/time "d" to renetlist, even if they wouldn't otherwise have needed renetlisted.

-NOPAREN Remove parenthesis from all node names in the top level macro and any equiv nodes, while checking for any "node name collisions" this causes.

-PREFIX prefix_string
Use the specified name as the prefix for all filenames instead of "ecad". This way you can keep netlists that were produced with different expand parameters in totally separate sets of files.

Default if omitted: Use the string "ecad" as the prefix for all temporary and permanent filenames

EXAMPLES

1. $ ecadlist topmacro
2. $ ecadlist /user/xyz1234/logic/project4 -force -e /user/micro/expfile

D.5.3.2 The Netlist Compiler: LOGLVS

The following is a copy of the command script that calls LOGLVS. It first sets the maximum allocation parameter. This is necessary for large (>50000 actual transistors) designs, but may otherwise be left out. The script then loads the netlist and the circuit library files, and compiles (links and connects) them.

```
//webster/ecad/loglvs <<!
    transistor 500000
    inp ecad.net
    cir ~/larry/thesis/ecad/circuit.lib
    link
    con
    exit
!
```

D.5.4 Layout Parameter Extraction

The file containing the design rules for LPE is located in the directory /user/pub/design_checking and is called "scn_lpe.com". The calling scripts are located in /user/pub. Before LPE can be run the same steps have to be followed as for an LVS run, as LPE runs an LVS in its first stages. The layout must have the gdsii_text properties for VCC, VSS, and all input and output nodes. Also, the schematic must have been run through the netlister and LOGLVS. To run the Layout Parameter Extractor, enter one of the following commands at the prompt:

$: ~/pub/scn_lpe1 <layout_name> <schematic_name>
or
$: ~/pub/scn_lpe2 <layout_name> <schematic_name>

After the LPE run is completed, a file called "SPICE.DAT" is created in the schematic's directory. This is a normal spice deck. Parasitic capacitors lettered A - H larger than 0.0001 Pf may be formed for the transistors nodes.

CAP[A] corresponds to metal1 to poly parasitic capacitance

CAP[B] corresponds to metal2 to metal1 parasitic capacitance

CAP[C] corresponds to n active to substrate parasitic capacitance

CAP[D] corresponds to p active to substrate parasitic capacitance

CAP[E] corresponds to metal1 to substrate parasitic capacitance

CAP[F] corresponds to metal2 to substrate parasitic capacitance

CAP[G] corresponds to poly to substrate parasitic capacitance

CAP[H] corresponds to metal2 to poly parasitic capacitance

The parasitic capacitance values used were supplied by MOSIS for the first test 2.0μm N-well CMOS chip run at VLSI Technology and will need to be updated as these values change. The LPECHK command tells LPE to use the supplied schematic netlist from LOGLVS, the BLS switches form basic gates and flatten nodes. Be very careful in running LPE on large designs as it creates very LARGE data files while running. Individual cells should be run separately to avoid this problem. For more information on LPE see the Cadence Dracula II manuals.

D.6 REMEDI On-line Design Rule Checking

D.6.1 REMEDI Overview

The REMEDI tools that are included in the Chipgraph software package are used to provide a quick first-pass design rule check. It should not be used in place of complete design rule checkers such as Checkmate or Dracula II.

In order to provide an easier interface to the REMEDI tools, several macros were written to run design rule checks on the layout (See RIT REMEDI User's Manual). These macros can be executed from the Chipgraph static menus along the right side of the edit window.

The file /user/larry/thesis/chipgraph/menu contains the declaration statements for the additional static menus and menu items for Chipgraph. Many of these are for REMEDI macros that have been written for both the CMOS library and other processes used by several RIT courses (i.e. nMOS and pMOS checks).

The Chipgraph menus contain other features besides those relating to REMEDI. For information on modifying the CMOS Standard Cell library menus see Section 4.4.

D.6.2 The Custom REMEDI Menus

By looking at the "menu" file it should be fairly obvious how the menus are done. If more information is required see IDEA Series Common User Interface Manual. An example line follows:

def it 'REMEDI/CMOS remedi/poly' do ".../chipgraph/macros/remedi/cmos/poly.mac" -e

(where the "..." is /user/larry/thesis)

This line will create a box in the static menus called 'poly' which resides under the 'CMOS' sub menu which is below the 'REMEDI' submenu. When the user places the curser on the box and hits the left mouse button, the command 'do "/user/larry/thesis/chipgraph/macros- /remedi/cmos/poly.mac" -e' will be executed. The '-e' argument causes the line to be executed immediately. This could also be executed from the command line by typing the same line sans the '-e'.

By creating a line similar to this for each macro to be run, an easy interface to the REMEDI tools was developed. There are also several other boxes in each of the REMEDI menus. One is 'view err lay'. This box simply turns off all layers and then turns on the error layer. There is also 'view all lay' which turns on all the layers for full viewing. The 'clear drc' box is used to remove the errors layers from the display.

If one wanted to add new features to the Chipgraph menus, simply look through the "menu" file and examine the techniques used there. It is **HIGHLY** recommended that any

new additions be written into an ASCII file using the screen editor and perfected **BEFORE** incorporating them into the menus that everyone else uses.

To do this, simply create the ASCII file and put in the menus definitions. After this has been done call up Chipgraph using 'chip' or 'chipgraph -st' (you must have the proper links created in your user_data account to do this. See RIT REMEDI User's Manual). After Chipgraph has come up type:

Chipgraph> do filename

where "filename" is the name of the file containing the new menus. This technique will allow you to perfect your design without affecting the menus that others are using. When the new additions are perfected simply place them in the "menu" file wherever seems appropriate.

To recompile the menus simply type (while in /user/larry/thesis/chipgraph):

$: update

This will re-compile the menus and place them in a file call "rmenu". The makefile will also place the data of the last compilation in the "last_compiled" file.

D.6.3 Example REMEDI Macro File

Using the same example as before:

def it ' REMEDI/CMOS remedi/poly' do ".../chipgraph/macros/remedi/cmos/poly.mac" -e

the operation of the REMEDI macros will be explained. The following is a listing of the poly.mac file:

```
# This macro file is used to check the
# poly.i
# layer of the
# scn_cmos
# process

   Forget drc scan
   DRC1 drc1_poly poly.i -Width -Space -ACUte -SELF 0 0 ^$act_win[1]
   DRC2 drc2_poly_active poly.i active_area.i -Space -Extension 0 0 ^$act_win[1]
   show drc scan
   fit drc
```

Basically this file contains nothing more than the REMEDI commands that you would type in at the command line to execute several layer checks. The '^$act_win[1]' is used to

place the name of the active window (W0, W1, etc) at the end of the line to meet the requirements of the REMEDI DRCx checks.

D.7 Submitting Projects to MOSIS

All projects submitted to MOSIS for fabrication must be done through a class account. Each class account must have submitted the registration list to MOSIS previously. There is also an A-class account (one per university), which is currently administered by Dr. Unnikrishnan, for graduate studies. Consult your professor or department liaison for more information.

Projects must be submitted electronically via the Internet (as per the conditions of our National Science Foundation grant). This is effectively done by having the MOSIS mail server log on to our system remotely and ftp the appropriate CIF files out of a specially created account (/user/mosis).

MOSIS does not perform any design rule checks on your design. You are responsible for any and all design rule checking.

It is best to use one of MOSIS' standard sizes and/or standard pad frames, but not necessary. For more information on these, it is STRONGLY recommended that you read the MOSIS Information Guide, available from your local liaison. **MAKE SURE THAT IT IS UP TO DATE!!!** MOSIS operates in the real world, and is therefore constantly undergoing changes in its policies.

Once you are sure that your design conforms to all of MOSIS' specifications, type:

```
$ : ~/pub/tape_out_scn cell_name
```

This will create a CIF file cell_name.cif. If there are no error messages, then you are ready to submit your file to MOSIS. To do this, have a system administrator copy your CIF file to /user/mosis, and consult your liaison to get the account name and password, and to insure that there is enough money left in the MOSIS class account.

When mailing to the MOSIS mail server, address mail (from the vaxen) to in%"MOSIS@MOSIS.EDU". The subject field should indicate (briefly) who you are and what you want. It is not used by the server, but may be referenced by a technician if you run into problems, and this helps them help you.

There are several steps to actual submission. Each consists of EMAILing a message to the MOSIS mail server and getting its response. Things to know before you begin consist of:

1. The die size of your chip

2. The number of pins of your chip

3. The CIF checksum of your chip. To get this, type:

   $: ~/pub/cif_checksum cell_name

   The source for this was obtained from MOSIS and is in /user/larry/thesis.

4. Your MOSIS account, and password.

5. Your project name and password (you make up).

First, you must make a "New Project Request". This includes information on who you are , what the project is, and some information about the chip. A sample request follows. Please note that the various fields must come in the proper order. Also note that the technology is TINY-SCN. This indicates that this is a Tiny Chip, which gets an educational price of $450 (as opposed to $2300). To be sure that your chip conforms to Tiny Chip specifications, consult your MOSIS liaison. One of the Tiny Chip specifications you must conform to is that you must use a standard frame.

```
REQUEST:     NEW-PROJECT
D-NAME:      Your d-name (get from your liaison)
ACCOUNT:     Your account name (get from you liaison)
D-PASSWORD: Your account password (get from your liaison)
NET-ADDRESS: LHR0403@RITVAX.ISC.RIT.EDU
MAILING-ADDRESS:  Mr. Larry Rubin
                  Dept. of Computer Engineering
                  Rochester Institute of Technology
                  1 Lomb Memorial Drive
                  Rochester, New York 14623
PHONE:       (716) 475-2987
P-NAME:      TEST1
P-PASSWORD:  MORPH
DESCRIPTION: This is a test chip to compare the characteristics
               of the NSA (CMOSN) cell library with one being
               developed here. The results will determine which
               library will be used for subsequent masters projects.
TECHNOLOGY:  TINY-SCN
LAMBDA:      1.0
PADS:        40
REQUEST:     END
```

If this is processed properly, and your account information is valid, you will receive a message back giving you a project ID number. You may then make a "Submit Request", after putting a copy of your CIF file in /user/mosis. A sample follows:

```
REQUEST: SUBMIT
ID:          32896
P-PASSWORD:  MORPH
```

STD-FRAME: 40PC22x22
NET-ADDRESS: LHR0403@RITVAX.ISC.RIT.EDU
CIF-CHECKSUM: 91285030 1854226
CIF-FTP-PATH: /129.21.30.9/mosis/mosis//test1.cif/
REQUEST: END

The MOSIS mail server will then extract your CIF file from the mosis account on our system and compare its checksum with yours. If your design passes the checksum and a pad bonding rule test, then the MOSIS mail server will send you back a "send fabricate request to queue for fabrication" message.

All that is left to do then is the actual "Fabrication Request". A sample template follows:

REQUEST: FABRICATE
ID: 32896
P-PASSWORD: MORPH
REQUEST: END

If all goes well, then in a little over a month (depending on the schedule), MOSIS will mail the packaged and unpackaged dies to the address given in the new project request.

Rochester Institute of Technology's

REMEDI

User's Manual

Jeff Correll

7/23/91

Introduction:

REMEDI is a design rule checking tool that is invoked from the menus within CHIPGRAPH. Because it does reside within CHIPGRAPH, REMEDI circumvents the need for exiting CHIPGRAPH to run an external design rule checker such as Dracula. Although REMEDI is a more convenient DRC (Design Rule Checker), it does not understand the concepts of nodes or pseudo-layers. Because of these two shortcomings REMEDI can address only the simplest of DRC errors. However, used properly, REMEDI can greatly reduce the frequency of Dracula invocations and thus the total time necessary to produce an error-free layout.

Although REMEDI can be used with any set of process design rules, the local version of REMEDI is only configured to operate with the RITNMOS and MOSIS scn_cmos processes.

Using REMEDI:

Before using REMEDI, a link must be created in your /user_data directory that points to a startup file. The following steps will create the link for you:

1. Log into your Apollo account.
2. Use the **wd** command to change directory into /user_data from your HOME directory.
3. Type: crl chipgraph.startup3 /user/larry/thesis/chipgraph/chipgraph.startup

This link need only be created once. Now you can use the REMEDI menus by invoking chipgraph as such:

$: chipgraph -st

This command will start CHIPGRAPH and tell it to look in your /user_data directory for a startup file to use.

Once CHIPGRAPH has come up, load your process (RITNMOS or MOSIS.scn_cmos) as normal. (Note: If one of the two aforementioned processes are not used, then REMEDI will produce unknown layer errors when invoked) When REMEDI is needed simply click on the TRAVEL button at the top of the MAIN window. When the menu tree appears, you will notice an entry at the bottom for REMEDI with two leaf menus nMOS and CMOS. Next click on the appropriate menu and a list of the internal layers for that process will appear in the MAIN window.

The layer names that appear will activate macros that check errors associated with the layer selected. REMEDI will check all the polygons on the edit sheet, even though they may not be visible (ie. you have zoomed in on a particular section). In the case of hierarchical layouts, REMEDI will only check the sections of the layout that have been peeked (this may or may not generate false errors).

In addition to the layer names, there are also several other menu entries: **all rules, view only err, view all layers, clear drc** and in CMOS, **top level routing**. These macros perform specific checks or manipulations of the visible layers:

all rules performs a comprehensive rules check.

view only err turns off all the layers except for the error layer.

view all layers turns all the layers on.

clear drc removes the highlighted error layer from the layout.

top level rout checks all of the rules associated with routing busses. These include metal1, metal2, poly, poly contacts and vias.

Example: To check the errors associated with the POLY layer, click on **poly**. The transcript window will show a list of checks performed, a rule name for each check, the micron measurement value of the rule checked, the number of errors found and the layers associated with the check. If there are no errors the *# errors* column will read 0. If there were errors, then the *# errors* column will display the number of errors detected for each check performed. In addition the portion of the layout in error will be highlighted in white and the screen will be centered around those errors.

During a REMEDI run, a transcript of the checks and the errors found for that check will be displayed in the transcript window. The following is a list of the different types of checks that exist and a brief description of each:

WIDTH - This check examines the width of a polygon

SPACE - This check examines the distance between two polygons on the same layer or different layers

ACUTE - This check looks for a vertex on a polygon that is less than 90 degrees.

AREA - This checks the area of a polygon.

A sample scan transcript is attached. Note that the first row of the scan table contains an entry for drc1_poly check of type WIDTH. There are 5 errors of this type and the first check measures polygons of poly.int for a 4.0 micron width. What this means is that there were 5 places in the checked layout where poly polygons were less then 4.0 microns wide. The other entries follow similarly.

After the error(s) have been corrected, the same check may be run again by simply clicking on the same layer name. To remove the error layer without running another check, simply click on the CLEAR DRC button.

Note: When using the single layer checks only the errors associated with that layer may be viewed at any one time.

You may also switch between the other menus of CHIPGRAPH and those of REMEDI at any time. The error layer will remain until the CLEAR DRC button is clicked or "forget drc all" is typed on the command line.

Note: In order to check, save, and close the design you **must** CLEAR DRC first. If you don't, Chipgraph will not allow you to write and quit.

DISCLAIMERS:

The following is a list of known problems with the implementation of the REMEDI macros. If any other problems are encountered, please note the RULENAME of the check in question and contact Jeff Correll or Professor George Brown. A print out of the test transcript and cell layout would be appreciated.

- o REMEDI does not understand the concept of pseudo-layers and therefore cannot correctly check for errors with well/sub ties. It will always flag these as errors.

- o The area checking feature of REMEDI does not work. So care should be taken with the size of contact cuts and vias.

```
 [ DO /user/larry/thesis/chipgraph/macros/remedi/nmos/poly.mac
#  Note: No DRC database loaded (from Idea/CDS/DRC 0A)
#  Note: DRC database successfully created (from Idea/CDS/DRC 25)
#  Note: Check WIDTH has been activated. (from Idea/CDS/DRC 10)
#  Note: Check SPACE has been activated. (from Idea/CDS/DRC 10)
#  Note: Check AREA has been activated. (from Idea/CDS/DRC 10)
#  Note: Check ACUTE has been activated. (from Idea/CDS/DRC 10)
# RULENAME   CHECK  AMOUNT  # ERR  # ORIG  LAYER 1  LAYER 2  OPTIONS  TIME /DATE
# ------------------------------------------------------------------------------
#  drc1_poly    WIDTH    4.00      5       5       Poly.int    Poly.int  <none>   7/02/1991 13:21
#  drc1_poly    SPACE    4.00      2       2       Poly.int    Poly.int  <none>   7/02/1991 13:21
#  drc1_poly    AREA    16.00      0       0       Poly.int    Poly.int  <none>   7/02/1991 13:21
#  drc1_poly    ACUTE    0.00      0       0       Poly.int    Poly.int  <none>   7/02/1991 13:21
#  Note: Check SPACE has been activated. (from Idea/CDS/DRC 10)
# RULENAME   CHECK  AMOUNT  # ERR  # ORIG  LAYER 1  LAYER 2  OPTIONS  TIME /DATE
# ------------------------------------------------------------------------------
# drc2_poly_diff  SPACE  2.00      3       3     Poly.int   Diffusion.int  <none>   7/02/1991 13:21
```

F. Layout Plot of the MAP

The following MAP pad list starts in the upper left hand corner and counts clockwise.

1	VCC (UPPER LEFT)	45	X6(8)	89	VSS
2	W(8)	46	X6(7)	90	COLBLNK(0)
3	W(7)	47	X6(6)	91	COLBLNK(1)
4	W(6)	48	X6(5)	92	COLBLNK(2)
5	W(5)	49	X6(4)	93	COLBLNK(3)
6	W(4)	50	X6(3)	94	COLBLNK(4)
7	W(3)	51	X6(2)	95	COLBLNK(5)
8	W(2)	52	X6(1)	96	ROWBLNK(1)
9	W(1)	53	X6(0)	97	---
10	W(0)	54	VSS	98	---
11	ROWBLNK(2)	55	X7(8)	99	---
12	VCC	56	X7(7)	100	VSS (LOWER LEFT)
13	X3(0)	57	X7(6)	101	---
14	X3(1)	58	X7(5)	102	---
15	X3(2)	59	X7(4)	103	---
16	X3(3)	60	X7(3)	104	---
17	X3(4)	61	X7(2)	105	---
18	X3(5)	62	X7(1)	106	---
19	X3(6)	63	X7(0)	107	---
20	X3(7)	64	W_LATCH	108	---
21	X3(8)	65	WCLK	109	---
22	VSS	66	MAX	110	---
23	ROWBLNK(3)	67	VCC (LOWER RIGHT)	111	---
24	ROWBLNK(4)	68	CLK	112	VCC
25	X4(8)	69	XO(0)	113	X2(0)
26	X4(7)	70	XO(1)	114	X2(1)
27	X4(6)	71	XO(2)	115	X2(2)
28	X4(5)	72	XO(3)	116	X2(3)
29	X4(4)	73	XO(4)	117	X2(4)
30	X4(3)	74	XO(5)	118	X2(5)
31	X4(2)	75	XO(6)	119	X2(6)
32	X4(1)	76	XO(7)	120	X2(7)
33	X4(0)	77	XO(8)	121	X2(8)
34	VSS (UPPER RIGHT)	78	VCC	122	VSS
35	X5(8)	79	ROWBLNK(5)	123	X1(0)
36	X5(7)	80	Y(0)	124	X1(1)
37	X5(6)	81	Y(1)	125	X1(2)
38	X5(5)	82	Y(2)	126	X1(3)
39	X5(4)	83	Y(3)	127	X1(4)
40	X5(3)	84	Y(4)	128	X1(5)
41	X5(2)	85	Y(5)	129	X1(6)
42	X5(1)	86	Y(6)	130	X1(7)
43	X5(0)	87	Y(7)	131	X1(8)
44	VCC	88	Y(8)	132	ROWBLNK(0)