CELLULAR IMAGE PROCESSING TECHNIQUES FOR VLSI CIRCUIT LAYOUT VALIDATION AND ROUTING*

T. N. Mudge, R. A. Rutenbar, R. M. Lougheed**, D. E. Atkins

Electrical and Computer Engineering Department
University of Michigan
Ann Arbor, MI 48109
and
**Environmental Research Institute of Michigan
Ann Arbor, MI 48107

Abstract

The architecture of the Cytocomputer‡, an existing special-purpose, pipelined cellular image processor, is described. A formalism used to express cellular operations on images is then given. Cellular image processing algorithms are then developed that perform (1) design rule checks (DRC's) on VLSI circuit layouts, and (2) Lee-type wire routing. Two sets of cellular image processing transformations for checking the Mead and Conway design rules and for Lee-routing have been defined and used to program the Cytocomputer. Some experimental results are shown for these cellular implementations.

1. Introduction

Integrated systems of VLSI complexity require sophisticated computer-assisted tools during the implementation phases (e.g., cell placement, routing) and validation phases (e.g., simulation, DRC's, connectivity verification) of the total design process. Consequently, much research is being conducted to provide smarter, faster, cheaper, and better CAD aids. This work describes the use of special-purpose hardware to improve the performance of CAD tools. We show how algorithms for design rule checking and routing are performed on bit-map representations of IC masks using the Cytocomputer [1,2,3], a special-purpose computer developed for high-speed image processing by the Environmental Research Institute of Michigan (ERIM). The design rules checked are those proposed by Mead and Conway in [4], and detailed more in [5]; they are appropriate for nMOS technology in which the features conform to a rectangular grid. The routing scheme implemented uses the well known Lee algorithm [6] as a foundation. This paper shows that efficient processing of these tasks can be done by a computer whose architecture is optimized for cellular image processing. In particular, it shows how these tasks are performed using neighborhood transformations that are no more than Boolean operations and table-lookups on the elements of the bit-map representation of the IC masks.

The Cytocomputer is a special-purpose high-speed image processor based upon cellular automata concepts. It consists of a pipeline of identical processing stages capable of performing neighborhood transforms. Experiments were performed using a prototype Model II Cytocomputer configured with one processing stage[1] and controlled by an interpreter that allows the machine to be used interactively for algorithm development. The paper outlines the machine's architecture and how it implements the above neighborhood transformations.

Our use of a mathematical formalism [9,10] to express cellular algorithms is motivated by a desire to produce algorithms whose properties can be analyzed. Where possible, we prefer to formulate algorithms generically, especially for DRC applications. Design at this level, rather than by exhaustive pattern inspection, will minimize the effort needed to update the checker when the rules change.

A course in VLSI circuit design and fabrication offered by the ECE Department at the University of Michigan motivated our investigation of DRC algorithms. Portions of the masks produced here have served as benchmarks for these experiments. Results presented in Section 4 suggest that a multi-stage Cytocomputer can perform the necessary DRC's fast enough for an interactive environment.

Routing experiments were performed on images representing rectangular grids with arbitrary obstacles; single layer conductors and two-point nets were considered. Results presented in section 5 indicate that routing time can be decreased significantly over similar serial/software implementations.

The paper is organized as follows: the next section describes the Cytocomputer architecture and its capabilities; Section 3 presents the formalism in which our cellular algorithms are expressed; Sections 4 and 5 describe, respectively, the Cytocomputer implementations of the design rule checker and router, including experimental results; and Section 6 is the conclusion.

2. Cytocomputer Architecture

An obvious architecture for implementing algorithms which are cellular in nature, e.g., bit-map manipulations, Lee-routing, etc, is a two dimensional parallel array of processors. Indeed, several such architectures have been proposed to support various design automation activities [11,12,13,14]. In such an array one processor exists for each cell in the cell-space, where a cell may be a square on a routing grid, a bit in a bit-map, or, in our terminology, a *pixel* in an *image*. These processors operate in lock-step and communicate with their nearest neighbors to perform *neighborhood* operations. Although such architectures can

*This work was supported in part by NSF grants MCS-8009315 and MCS-8007298, and the Environmental Research Institute of Michigan.

**R. M. Lougheed is also an instructor with the University of Michigan.

‡"Cytocomputer" is a trademark of the Environmental Research Institute of Michigan for image processing systems.

[1] Multiple processing stages can be used. Indeed, results of earlier work on design rule checking using a simpler 88-stage TTL prototype of the Cytocomputer appear in [7,8].

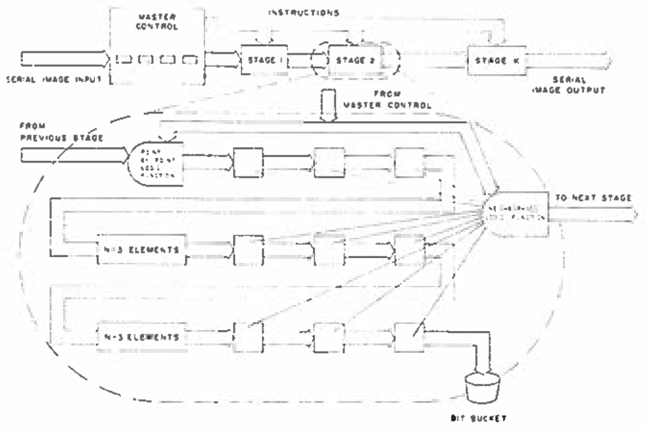


Figure 1. Cytocomputer Block Diagram

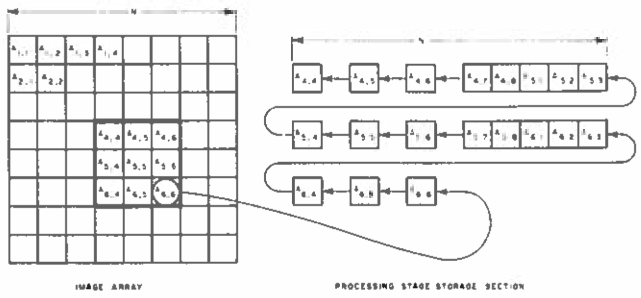


Figure 2. Moving Window Implemented With Shift Register

be very fast, they have drawbacks in light of the limitations of present fabrication and packaging technology.

The practical shortcomings of parallel array image processors led ERIM to develop an alternative parallel structure, the Cytocomputer. In [2] Lougheed and McCubbrey compare the Cytocomputer and array cellular processor architectures and show that the Cytocomputer architecture can have several advantages over the array architecture, such as low complexity, high bandwidth, and considerable architectural flexibility. A Cytocomputer (see Figure 1) consists of a serial pipeline of neighborhood processing stages, with a common clock, in which each stage in the pipeline performs a single neighborhood transformation of an entire image. Images are entered into the pipeline as a stream of eight-bit pixels in sequential line-scanned format and progress through the pipeline of processing stages at a constant rate. Alternatively, the data stream can be viewed as eight independent one-bit images. Following the initial delay to fill the pipeline, processed images are produced at the same rate they are entered. Shift registers within each stage store two contiguous scan lines while window registers hold the nine neighborhood pixels which constitute the 3x3 window input of a neighborhood logic module. This module performs a preprogrammed transformation of the center pixel based on the values of the center and its eight neighbors. Neighborhood logic transformations are computed within the data transfer clock period, allowing the output of a stage to appear at the same rate as its input. At each discrete time interval a new pixel is clocked into the stage. Simultaneously, the contents of all delay units are shifted one element. In addition, operations which do not involve the states of a pixel's neighbors, such as bit anding, bit setting, etc., are performed in a separate point-by-point logic section to simplify the neighborhood logic circuit.

To visualize the transformation process, imagine a 3x3 window moving across an image array as shown in Figure 2. The processing stage storage section shows the contents of the latches after pixel A[6,6] has been read. The contents of the neighborhood latches allow the stage to compute the transformed value of the pixel in the center latch, A[5,5]. This transformed pixel becomes an element of the serial input to the next stage. From this example, one can see that the latency of a stage is equal to N + 2 time steps, N being the line length. One can now visualize a series of 3x3 stage windows following each other across the image, each one processing the previous stage's output as shown in Figure 1.

A simple expression can be developed which approximates the time T required for a Cytocomputer with S pipeline stages to perform K neighborhood operations on an image with M × N pixels. Let t_c be the stage cycle time in sec/pixel, then

$$T \approx \lceil K/S \rceil \, (S(N+2)t_c + MNt_c).$$

The $\lceil K/S \rceil$ term is the number of passes through the Cytocomputer pipeline; the last term accounts for the time to fill and empty this pipe[2].

The concept of a serial processing stage has previously been described in [15]. The important features of a Cytocomputer involve the "chaining" of a large number of physically identical, software programmable modules to achieve high speed (real-time) image processing. This gives very low cost solutions to the majority of image processing problems in which the original source and final destination of the images are serial in nature, i.e., color-display terminals, disk storage, etc.

Cytocomputer architecture lends itself well to VLSI implementation for several reasons. First and foremost, the limited number of I/O connections required between stages fits well within the pin constraints of single packages. As IC technology advances, more stages may be combined in a single package with no increase in pin requirements. In addition, the large number of identical parts per system provides the volume necessary for economical manufacture. Finally, the flexible, programmable nature of the devices means diverse areas of application are possible, leading to a wide market for such a part.

Following the successful completion of the first generation system, a design effort was carried out to produce a processing stage suitable for implementation in a custom LSI circuit. This design has been fabricated by a major IC vendor with production scheduled for mid-1982; engineering samples are currently being evaluated. Each chip incorporates all functions of the previous generation hardware and has a cycle time $t_c \approx 1 - 2\ \mu s$. Because of the tremendous size reduction, a 100-stage Cytocomputer pipeline will fit on a few modest size circuit boards. All experiments reported herein were performed on a single-stage MSI TTL

[2] When K mod S ≠ 0, i.e., on the last pass through the pipeline, the unused processing stages are programmed to pass the image with negligible delay.

prototype of this chip with $t_c = 2\ \mu s$.

3. Formalism for Cellular Algorithms

We use a formalism called *Image Algebra* [9,10] which is related to the work of [16,17]. We treat a two-dimensional binary image as a set of points, e.g., the black points on a transparent mask, where a point is an ordered pair of coordinates in the plane. In our case the plane is a digital rectangular grid; this is a practical but not a mathematical necessity -- practical insights may be gained from the study of ideal problems in the continuous Euclidean plane.

Given that A and B are binary images, and hence sets, we have all the usual set-theoretic operators: intersection $A \cap B$, union $A \cup B$, difference A/B, complement $\bar{A}$. Also, the usual operations of addition and scaling can be applied to points. Next, we define the notion of *translation* of a set by a point: the set A translated by point p, denoted A_p is $\{a+p \mid a \in A\}$, which is A with its origin moved to p. With translation we can define the two essential primitives of *dilation* and *erosion*. Dilation $\oplus$, and erosion $\ominus$ are defined as follows:

$$A \oplus B = \bigcup_{b \in B} A_b \qquad A \ominus B = \{p \mid B_p \subseteq A\}$$

The dilation $A \oplus B$ is the union of translations of A by points from B. The erosion $A \ominus B$ is the set of points to which we can translate B and still have it contained in A. Loosely, we can think of erosion and dilation as formal generalizations of the intuitive ideas of expanding and shrinking. However, erosions and dilations are defined for arbitrarily complex sets A and B, whereas expands and shrinks are usually specified with simple patterns (sets).

The utility of this formalism rests on the fact that all these operations are *local* in nature. Each operation can be reduced to a sequence of neighborhood transformations in the elementary 3x3 window of the Cytocomputer stage. This property allows us to design algorithms at a high level, knowing that they can be turned into an executable sequence of Cytocomputer operations, minimizing the need for manual pattern-matching specification.

To illustrate, consider Figure 3 in which image A is eroded by the elementary image B. The points in the result are those points in A to which the origin of B (marked * in its center) can be translated and still have B contained in A. The Cytocomputer stage hardware implements this operation by a nearest neighbor transformation which tests each 3x3 window of pixels in A as follows: the center pixel of the window is set to 1 in the result just if the north, south, southwest, southeast and center pixels in A are currently set to 1 (ignoring the rest), i.e., the configuration *matches* image B.

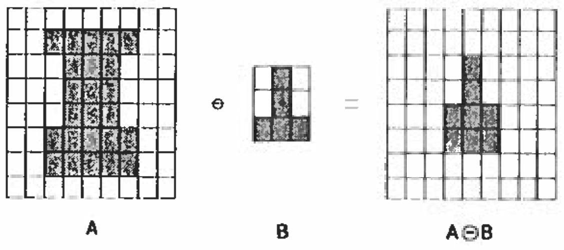


Figure 3. Elementary Erosion

This example uses an elementary image B. In general, operations with a larger, more complex B will be decomposed into a sequence of elementary operations suitable for execution in the stage hardware. The image algebra includes identities which permit simplifications similar to those done in Boolean algebra. For example, to compute $A \ominus B$ when B is a dilation of elementary images:

$$B = B_1 \oplus B_2 \oplus B_3 \oplus B_4$$

it is sufficient to compute

$$A \ominus B = ((((A \ominus B_1) \ominus B_2) \ominus B_3) \ominus B_4)$$

which can be done directly by the hardware.

4. Design Rule Checking

The design rules are a set of geometrical constraints that the masks of the wafer fabrication process must satisfy. There are two general approaches to the implementation of DRC's, which reflect the computer representation chosen for the IC mask. Geometric-shapes checkers perform checks on masks represented as sets of intersecting polygons or rectangles (see [18] for a survey). Raster-scan checkers (e.g. [19,20,21]) assume that the mask is represented as a grid whose squares are labeled according to the presence or absence of particular mask layers; the idea here is to pass a small window over the grid and identify local violation-patterns appearing in the window. Recently, a top-down hierarchical approach has been taken in the research on DRC's [22,23,24] wherein checking is done first on the cells which are repeated in the layout, and then on the interactions between the instances of these cells. This removes the need for a complete instantiation of the chip as polygons or grid elements.

The Cytocomputer checker is local and non-hierarchical in nature, i.e., global connectivity is not ascertained and a complete bit-map is required. However, the image-algebraic formalism frees our implementation of some of the tedious detail of pattern specification implied by the small-window metaphor.

4.1. Design Rules

The process of interest in this paper is that described in [4,5] for nMOS. The constraints are expressed in a dimensionless form as multiples of a basic length-unit λ.

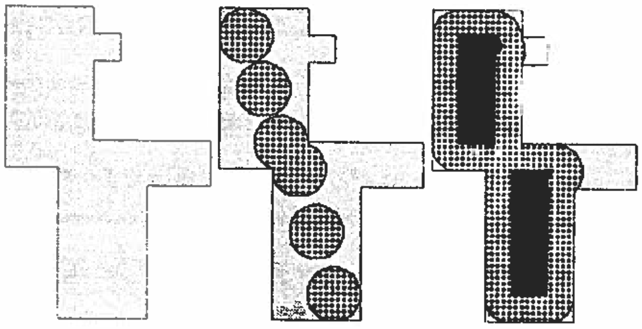
The masks involved are those for diffusion, contact windows, ion-implantation, metal, and polysilicon. The design rules partition into four classes:

1-mask-spatial:	width, spacing
n-mask-spatial:	mask-to-mask spacing
n-mask-logical:	coverage, containment
complex-entity:	transistor, contact, pad, etc

4.2. DRC Algorithms

To illustrate the cellular DRC, we describe here two different algorithms to perform a width-3λ 1-mask-spatial check on an orthogonal mask.

In the algorithms defined below it is convenient to think of the Cytocomputer as operating on eight separate binary images in parallel, rather than one image of eight-bit pixels. Recall from Section 2 that the Cytocomputer can operate in either of these modes. The images of the five masks of interest--contact cut, diffusion, implant, metal, polysilicon--can all be processed simultaneously on each pass through the machine. Binary pixels from each of these mask images occupy five of the eight bits in the Cytocomputer's data



(a) (b) (c)

Figure 4. (a) Original Mask, (b) Sliding Disk, (c) Regions Covered by Disk (dots) and Center (black)

path. The remaining three bits are the error-image and temporary variables for the intermediate results which occur when the neighborhood logic applies a transformation to an image, or when the point-to-point logic performs a bit-wise logical operation between images.

The first algorithm is based on the simple observation illustrated in Figure 4. At the left is a mask represented in the continuous plane, on which we wish to perform a width-*w* check. We imagine that a disk of diameter *w* slides around the inside of the object to all possible locations at which it may be completely contained. This appears in the middle of Figure 4. While it slides, we note those points covered by the disk and trace the path of its center. Intuitively, we know the disk should not pass through regions which are too narrow, i.e., regions which fail the width test. The result is shown at the right of the figure. Except for some square-corner effects, those regions left uncovered all violate the width test. Note also that the region traced out by the center of the disk is not connected across the diagonal neck of the mask.

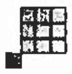
With these observations we can construct an executable width-3λ algorithm. Call the mask to check M. Because this mask will have features small with respect to the grid size, and because it will have square corners, we replace the disk with a 3x3 square to be denoted S. The operation of finding all points covered by S as it slides in M is called an *opening* [9,16]; it is the union of all translations of S that fit in M, which is precisely $((M\ominus S)\oplus S)$. The region traced out by the center of S is just $M\ominus S$; call this C. The algorithm can be outlined as follows:

Algorithm 1 : Width-3λ Test

Step 1: Open M with S. Areas of M not in the opening are errors.

Step 2: Erode M by S to get C. This is the path traced by the center of S.

Step 3: Tag the northeast corner of each component of C; call this set of points T. We are preparing to identify the regions of M which restrict the passage of S by finding the breaks in C.

Step 4: Dilate T over M with the following shape: 

This dilation intersects the southwest corner of another region of C if and only if a break has occurred along a northeast/southwest axis. Mark the points in this intersection as errors; they indicate a diagonal width violation.

Steps 5,6: Similar to steps 3,4 to find errors along the northwest/southeast axis.

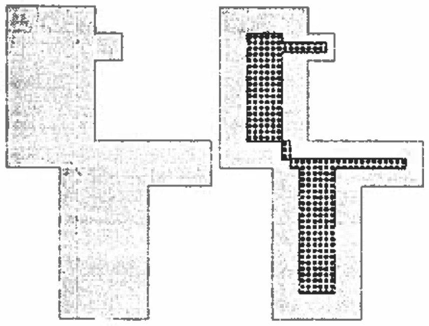
This algorithm tags the *north* side of each diagonal width violation, and any region smaller than $3\lambda \times 3\lambda$. It is also generic in the sense that it extends easily to larger width checks.

The second algorithm is again based on the observation that the center of the square S will not trace a single connected region as it slides through a figure with width violations. This algorithm uses the notion of a *homotopic thinning* [9,16], which conceptually shrinks a figure without changing its connectivity, i.e. without introducing holes or breaking the figure into several pieces. Figure 5 illustrates a simple thinning. Thinning can be implemented by nearest-neighbor transformations which sequentially erode the mask image from the north, east, south, and west directions, conditioned so that removal of a point does not alter the local connectivity of the figure. The algorithm can be summarized as follows:

Algorithm 2 : Width-3λ Test

Step 1: Open M by S. Any regions of M not in the opening are errors.

Step 2: Erode M by S to get C, then thin the opening from Step 1 and compare to C. Regions present in the thinning and not on the erosion mark diagonal width errors, i.e., they bridge the gaps between the disconnected blocks of C, marking the errors.



(a) (b)

Figure 5. (a) Original Mask, (b) Thinning Superimposed

4.3. DRC Experimental Results

Algorithm 1 required 15 Cytocomputer stages to run, and Algorithm 2 required 6 stages. Figure 6 shows the results of running each algorithm on a simple test pattern. Notice all errors are correctly detected. On the single-stage Cytocomputer prototype, each of these tests required about 0.1 seconds to run. Depending on the sophistication of the checking needed, and the amount of extra processing required to label errors in meaningful ways, a complete DRC will take between 150 and 250 stages. Table 1 estimates the time required to run a complete DRC on a $2000\lambda \times 2000\lambda$ chip for several different pipeline lengths. We assume a cycle time $t_c = 1\mu s$, and a sufficient mask-data transfer rate to keep pace with the pipeline.

Pipeline	DRC Time in Seconds	
Length	150-Step-DRC	250-Step-DRC
1	600.	1000.
10	60.	100.
100	8.4	10.5
250	4.5	4.5

Table 1. Estimated DRC Time, $2000\lambda \times 2000\lambda$ IC

5. Cytocomputer Routing

Although the Lee routing algorithm [6] was introduced over two decades ago, a recent survey [25] indicates that it is still a popular approach for PC board and LSI routing applications. The Cytocomputer router is implemented using a specially designed version of the Lee algorithm for two-point nets with a single conductor layer. Note that the Cytocomputer router is analogous to the *inner-loop* of a software implementation: its only job is to find a path between two points on a grid. It is the job of the host processor to deal with wire-list determination, ordering, unroutable connections, etc.

5.1. The Lee Algorithm

Because of the Cytocomputer's 8-bit pixel size, we chose not to label cells in the expanding Lee wavefront with numerical weights representing distance from the source, but rather, as is usually done, we encoded the states of the cells into a few values. The encoding used is that mentioned for the array architecture in [11], where cells are labeled with rectilinear arrows pointing back to the source of the wavefront. Our cell states are encoded into the following alphabet:

S, T	source and target cells
O, #	free and blocked cells
←, ↑, ↓, →	arrows on wavefront pointing to source
T←, T↑, T↓, T→	target labeled by wavefront (path found)
ST	source labeled during target traceback

The routing of a single wire takes three steps: first, *grow-out* from the source S a wavefront of arrows until target T is labeled; second, *trace-back* along the source-pointing arrows from the target to the source; last, *clean-up* extraneous labeled cells not on the new path and mark the new path as an obstacle.

5.2. Routing Algorithms

In the transformations that follow, it is now convenient to view the Cytocomputer as operating on 8-bit pixels each representing a number from 0 to 255. We have suitably encoded the cell alphabet given above into these values. Note that we are not using the image algebra here, but rather, we are using the Cytocomputer as an emulator for an array-processor whose processing-elements are finite-state machines.

The grow-out, trace-back, and clean-up procedures are now best thought of as neighborhood operations which change the value of the center pixel based on the result of set-membership tests on the pixels of the neighborhood. Let

$$\Lambda = \{ \leftarrow, \uparrow, \downarrow, \rightarrow, S \}$$

be called the *Label* set; membership in this set is tested in both the grow-out and trace-back procedures.

```
GROW-OUT:
  IF center ∈ { O, T }
  THEN BEGIN
     IF n ∈ Λ
     THEN attach ↑ to center
     ELSE IF w ∈ Λ
        THEN attach ← to center
        ELSE IF e ∈ Λ
           THEN attach → to center
           ELSE IF s ∈ Λ
           THEN attach ↓ to center
     END
```

GROW-OUT expands a wavefront of backward-pointing arrows from the source S. When more than one arrow labeling may be chosen, the arbitrary order tests in the transform chooses directions in the order ↑, ←, →, ↓. Note that the operation *attach* does the obvious function on the alphabet, e.g., T attach ↑ = T↑.

The grow transform is run until the target receives an arrow label, at which point the trace-back is run.

```
TRACE-BACK:
  IF center ∈ Λ AND
     ((n = T↓) OR (w = T→) OR (e = T←) OR (s = T↑))
  THEN attach T to center
```

TRACE-BACK simply follows the arrows from the target, attaching a T to each, until the source is reached and labeled ST[3]. At this point, clean-up is run, a simple point-by-point transform for the Cytocomputer, which removes extraneous arrows and labels the newly found path as an obstacle for future routes[4].

Figure 7 shows a simple example of these three processes.

5.3. Routing Experimental Results

As a typical problem, we routed a connection of length 700 on a 200 × 200 grid. Although this test required about 120 seconds (including trace-back) to run on the 1-stage Cytocomputer prototype, straightforward expansion to more pipeline stages drastically reduces this time. For example, with $t_c = 2.0\mu s$ a 100-stage system could route this example in 1.7 seconds; a 250-stage system would require 1.1 seconds. Table 2 shows the Cytocomputer time required to route wires of varying lengths on a 500 × 500 grid for different numbers of stages in the Cytocomputer pipeline. We assume $t_c = 1.0\mu s$ and that the data-transfer rate can keep up with the pipeline.

[3]Although trace-back is actually a sequential operation, for small images, long pipelines, or memory-limited host machines, it is reasonable to use the Cytocomputer here.

[4]To route multi-point nets, we could relabie each pixel on the new path as a Source, introduce a new Target, and route again.

Pipeline Length	Routing Time in Seconds		
	Wire-Len 100	Wire-Len 500	Wire-Len 1000
1	50.2	250.	500.
10	5.1	25.4	51.0
100	.60	3.00	6.00
250	.35	1.50	3.00

Table 2. Estimated Routing Time, 500x500 grid.

It can be seen that a modest sized system (≈ 100-stage system) is quite capable of routing even long wires rapidly enough for an interactive environment.

6. Conclusions and Future Research

Speedups in the time taken to check local design rules for VLSI layouts and to perform Lee-type routing can be achieved by applying some simple cellular image processing techniques in conjunction with the Cytocomputer, a special purpose computer oriented towards cellular image processing. The anticipated increase in complexity of IC designs together with the anticipated increase in demand for IC's would seem to justify such a machine as a component in a VLSI CAD system, particularly since a VLSI Cytocomputer implementation would represent only an incremental cost to the overall system.

In future research, we plan to extend our experiments to more complex design rules and routers, to study the application of the techniques presented here in an interactive CAD environment, and to consider alternate pipeline architectures for the Cytocomputer. For example, we are now studying the use of short *parallel* pipelines, rather than the single long pipeline assumed in the text, as a way to implement in parallel the individual checks that comprise a complete DRC.

Acknowledgements

For technical, administrative, and moral support at several points in this work, we acknowledge the assistance of Michelle Aminoff, Dr. William Becher, David N. Smith, William Teel, and Steven Westin.

References

[1] R. M. Lougheed, et al., "Cytocomputers: Architectures for Parallel Image Processing," *Proc. IEEE Workshop on Picture Data Description and Management*, Aug. 1980.

[2] R. M. Lougheed, D. L. McCubbrey, "The Cytocomputer: A Practical Pipelined Image Processor," *Proc. 7-th Annual International Symposium on Computer Architecture*, May 1980.

[3] S. R. Sternberg, "Cellular Computers and Biomedical Image Processing," *Proc. U.S. - France Seminar on Biomedical Image Processing*, May 1980.

[4] C. Mead, L. Conway, *Introduction to VLSI Systems* Addison-Wesley, Reading, 1980.

[5] R. F. Lyon, "Simplified Design Rules for VLSI Layouts," *Lambda*, vol. II, no. 1, 1st Quarter, 1981, pp. 54-59.

[6] C. Y. Lee, "An Algorithm for Path Connections and Its Applications," *IRE Trans. on Electronic Computers*, Vol. EC-10, September 1961, pp. 346-358.

[7] T. N. Mudge, R. M. Lougheed, and W. B. Teel, "Design Rule Checking for VLSI Circuits using a Cellular Computer," *Abstracts of the 1981 ACM Computer Science Conference*, St. Louis, February 1981, pp. 29.

[8] T. N. Mudge, R. M. Lougheed, and W. B. Teel, "Cellular Image Processing Techniques for Checking VLSI Circuit Layouts," *Proceedings of the 1981 Conference on Information Sciences and Systems*, The Johns Hopkins University, March 1981, pp. 315-320.

[9] S. R. Sternberg, *Image Algebra*, lecture notes, 1980.

[10] S. R. Sternberg, "Language and Architecture for Parallel Image Processing," in *Pattern Recognition in Practice*, E. S. Gelsema and L. N. Kanal, eds., North Holland Publishing Co., 1980.

[11] M. A. Breuer, K. Shamsa, "A Hardware Router," *Jour. of Digital Systems*, Vol. IV, Issue 4, 1981, pp. 393-408.

[12] A. Iosupovicz, "Design of an Iterative Array Maze Router," *Proc. ICCC*, 1980, pp. 908-911.

[13] C. R. Carroll, "A Smart Memory Array Processor for Two Layer Path Finding," *Proc. 2-nd Caltech Conference on Very Large Scale Integration*, January 1981.

[14] T. Blank, M. Stefik, W. vanCleemput, "A Parallel Bit Map Processor Architecture for DA Algorithms," *Proc. 18-th Design Automation Conference*, June/July, 1981, pp. 837-845.

[15] B. Kruse, "A Parallel Picture Processing Machine," *IEEE Trans. Computers*, Vol. C-22, 1973.

[16] J. Serra, *Image Analysis and Mathematical Morphology*, 1980, in press.

[17] G. Matheron, *Random Sets and Integral Geometry*, Wiley, New York, 1975.

[18] H. S. Baird, "Fast Algorithms for LSI Artwork Analysis," *Jour. of Design Automation and Fault Tolerant Computing*, Vol. 2, No. 2, May 1978, pp. 179-209.

[19] C. M. Baker, *Artwork Analysis Tools for VLSI Circuits*, (M.S. Thesis) MIT, 1980.

[20] C. M. Baker, C. Terman, "Tools for Verifying Integrated Circuit Designs," *Lambda*, 4-th Quarter, 1980.

[21] L. Seiler, "Special Purpose Hardware for Design Rule Checking," *Proc. 2-nd Caltech Conference on Very Large Scale Integration*, January 1981.

[22] T. Whitney, "A Hierarchical Design Rule Checking Algorithm," *Lambda*, vol. II, no. 1, 1981, pp. 40-43.

[23] M. E. Newell, D. T. Fitzpatrick, "Exploiting Structure in Integrated Circuit Design Analysis," *Proc. Conf. on Advanced Research in VLSI*, Jan. 1982, pp. 84-92.

[24] S. C. Johnson, "Hierarchical Design Validation Based on Rectangles," *Proc. Conf. on Advanced Research in VLSI*, Jan. 1982, pp. 97-100.

[25] M. A. Breuer, A. D. Friedman, A. Iosupovicz, "A Survey of the State of the Art of Design Automation," *Computer*, October 1981, pp. 58-75.

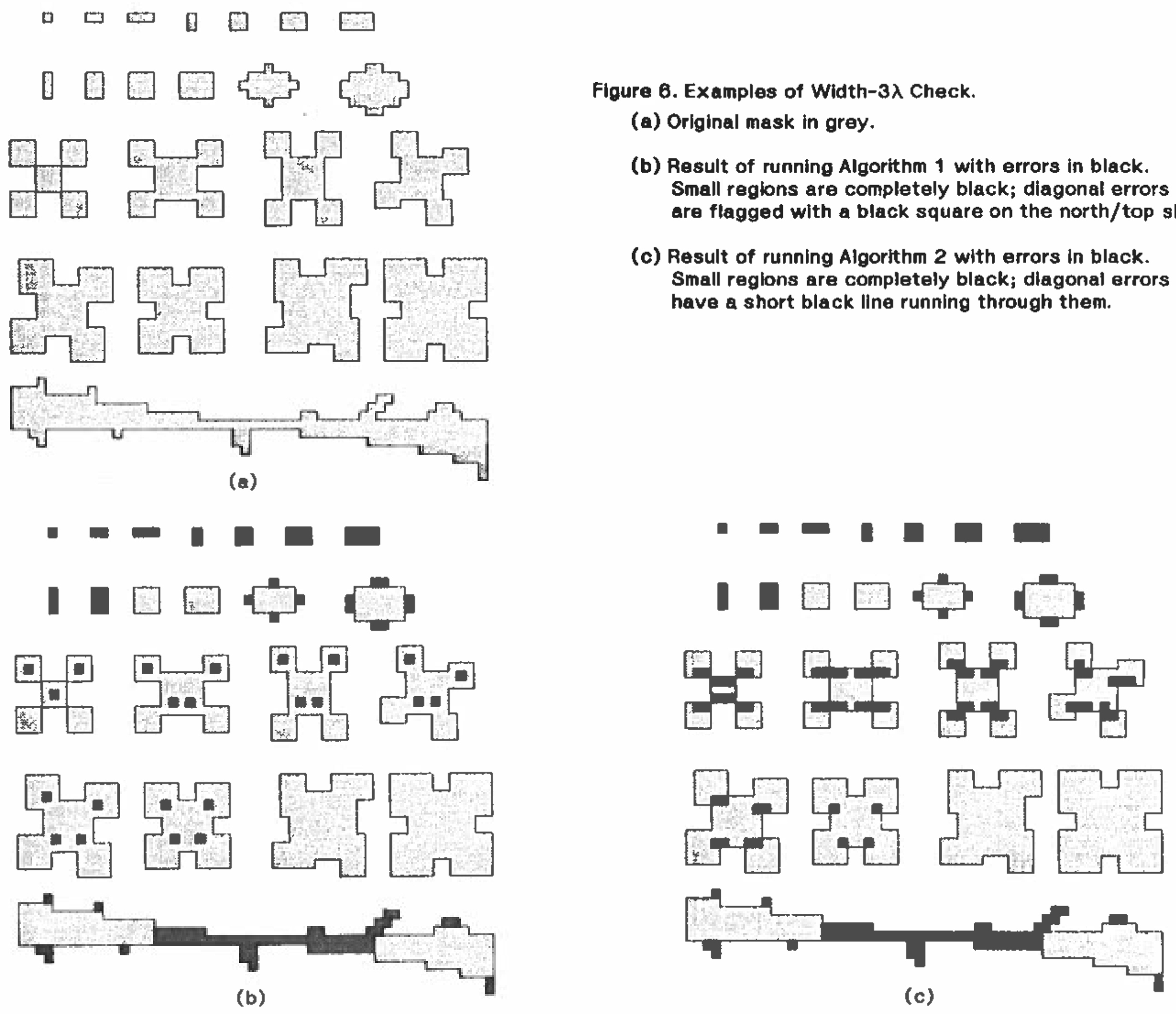


Figure 6. Examples of Width-3λ Check.
(a) Original mask in grey.
(b) Result of running Algorithm 1 with errors in black. Small regions are completely black; diagonal errors are flagged with a black square on the north/top side.
(c) Result of running Algorithm 2 with errors in black. Small regions are completely black; diagonal errors have a short black line running through them.

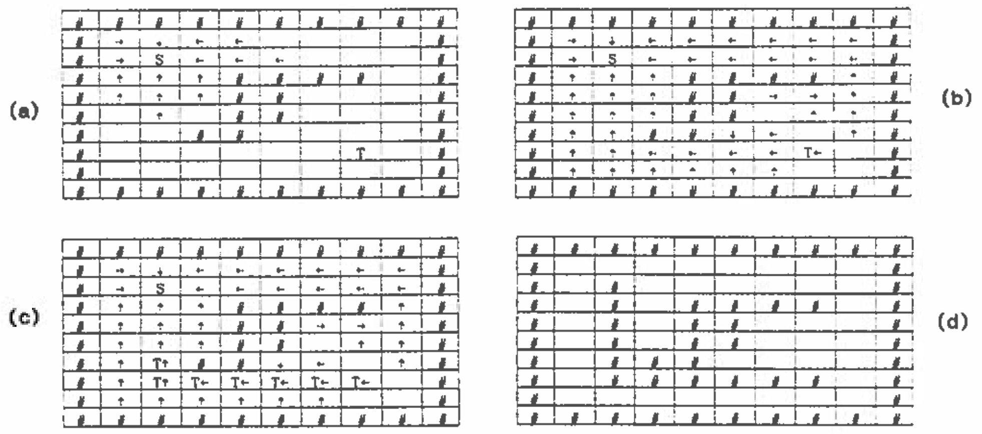


Figure 7. Routing Process.
(a) Grow-out in progress.
(b) Target found; end of grow-out.
(c) Trace-back in progress.
(d) After trace-back complete, clean-up.