

Novel method for the analysis of printed circuit images

Jon R. Mandeville

Manufacturing Research Center
IBM Thomas J. Watson Research Lab
Yorktown Heights, New York 10598

Abstract

To keep pace with the trend towards increased circuit integration, printed circuit patterns are becoming denser and more complex. A variety of automated visual inspection methods to detect circuit defects during manufacturing have been proposed. This paper describes a method that is a synthesis of the *reference-comparison* and the *generic property* approaches that exploits their respective strengths and overcomes their respective weaknesses. It is based on the observation that the local geometric and global topological correctness of a printed circuit can be inferred from the correctness of simplified, skeletal versions of the circuit in a test image. These operations can be realized using simple *processing elements* which are well suited for implementation in hardware.

1. Introduction

A variety of approaches for automated optical inspection of printed circuits have been reported over the last decade; see the tutorial surveys of Chin and Kruger.^{1,2} These approaches typically use an analog subsystem for part handling and image acquisition and a digital subsystem for image analysis and overall system control. Feature verification and defect detection is usually based on the analysis of *discrete, binary images* generated by sampling analog images on a rectangular grid and thresholding the result to a zero or one. As illustrated in Fig. 1, these images are represented as n by m matrices whose elements (*pixels*) are zero or one.

Most proposed methods for the analysis of printed circuit images are variations of either the *reference-comparison* or the *generic property* approaches. In general, reference-comparison uses complete knowledge of the circuit under test, whereas the generic property approach uses knowledge of properties common to a circuit family but not knowledge of the specific circuit under test. There are two types of reference-comparison: The simpler approaches involve some kind of direct image comparison. (e.g., boolean exclusive or) between pixels in a test image and pixels in an idealized reference image. Somewhat more sophisticated approaches involve recognition of circuit features in the test image (e.g., pads, corners, etc.) followed by comparison against a reference. The generic property approach also takes two forms. One is based on the notion that idealized circuit features are simple, regular geometric shapes, whereas defects typically are not. With this approach one looks for unexpected irregular features. The second approach directly verifies *design rules*, e.g., trace width, feature spacing, pad location and size, etc. In both forms, defects are usually detected using strictly local neighborhood processing throughout the test image.

In Section 2 we describe a method that is a synthesis of the reference-comparison and the generic property approaches. It is a powerful and flexible analysis technique to verify typical circuit features and detect typical circuit defects. It replaces both image comparison and design rule checkers, exploiting their strengths and overcoming their weaknesses. In Section 3 we describe algorithms based on the generic method for verifying trace width, feature spacing, and pads. Implementation of the basic image processing operations in simple, low cost digital hardware is briefly described in Section 4.

2. Overview of Analysis Method

Mathematical background

The concepts underlying our method are derived from recent work in *discrete geometry* on the geometric description and analysis of discrete, binary images; for formal treatments of discrete geometry, see the work of Pavlidis, Rosenfeld, and Serra.^{3,4,5,6} The concept of *neighboring pixels*, *connectivity*, and *regions* formalize the intuitive notion of what distinct objects are contained in an image. Two pixels are said to be neighbors if they are adjacent to one another either to the right or left or above or below or on the diagonal (formally, two pixels with indices (i,j) and (k,l) in image I are said to be neighbors if and only if $\max(|i-k|, |j-l|) \leq 1$). Two nonzero pixels are said to be connected if and only if there exists an unbroken sequence of nonzero neighbors between the two pixels. A region is a set of nonzero pixels each of which is connected to all other pixels in the set.

In addition, the image to image transformations *contraction*, *expansion*, and *thinning* provide a formalism to infer the shape, size, and topology of regions.^{7,8,9} Expansion expands regions by setting zero elements to one if certain of its neighbors are equal to one. With an appropriate sequence of expansion steps it is possible to achieve the discrete octagonal expansion as illustrated in Fig. 2 and Fig. 3(a) and (b). Contraction shrinks regions by setting ones to zero if certain of its neighbors are zero (Fig. 3(c)). As illustrated in Fig. 3(d), thinning reduces regions to their *skeleton*, a simplified, skeletal form. The thinning we use preserves the *homotopy* of an image (preserving the homotopy means, among other things, that the connectivity of regions and the holes contained in regions are preserved).

Finally, we use the notion of *joins* to classify important pixel patterns (Fig. 4). An *n-join* is a nonzero element with *n* nonzero neighbors. A *T-join* is a 3-join all of whose neighbors belong to a skeletonized region. A *blob-join* is a skeletal point that has a neighbor that is not a skeletal point. Join order and type play a key role in the inspection algorithms.

Generic method for analysis of printed circuit images

Application of the above concepts to the analysis of printed circuit images led to the following observation: the local geometric and global topological correctness of typical circuit features can be inferred from the correctness of skeletal versions of the circuit features in a test image. This insight in turn led to the following generic method for the analysis of printed circuit images:

Generic method for the analysis of printed circuit images

- Step 1: Transform and thin the test image in such a way that defects and good circuit features induce skeletal features that can be easily and reliably detected and classified.
- Step 2: Compile a detected feature list that records the position and type of all detected features.
- Step 3: Compare the detected feature list with a design feature list generated from circuit design data.
- Results: Features in the two lists that cannot be brought into correspondence imply faults.

The inspection algorithms described in this paper are instances of the generic method. Each algorithm uses a different thinning process designed so that a particular defect class induces a known corresponding class of skeletal features that can be easily and reliably detected. It turns out that the presence of 0-, 1-, T-, and blob-joints is sufficient to infer the existence of typical defects (as well as desired circuit features, such as pad-to-trace or trace-to-trace connections, trace ends, etc.).

The feature comparison method is quite flexible and powerful for two reasons. First, it is possible to define arbitrary correspondence criteria between arbitrary sets of detected and predicted features. In many cases, however, the following simple criterion probably will suffice: Detected and predicted features correspond if they are the same type and within a given distance of each other. Features that cannot be brought into correspondence imply the existence of defects; the type of defect can be inferred from the type of feature. Second, it is possible to derive measures of global circuit correctness by combining the results of all isolated feature correspondences, e.g., spatial distortion throughout the entire circuit layer.

For some circuit types and design rules it is not necessary to use reference data; the existence of certain skeletal features unambiguously implies defects. For example, if a circuit consists only of single width traces that must end in pads, 1-joints imply the presence of defects when using the algorithm for verifying trace width described in the Section 3. However, for other classes of circuits a comparison step is needed in order to distinguish between true defects and good circuit features that induce the same type of skeletal features.

Because our method does not compare a reference image and the test image pixel by pixel, it eliminates the need for the storage, generation, registration, and comparison of a reference image with the test image. Instead, a relatively small list of predicted feature types and locations is compared with a list of detected features in a straightforward way. Unlike direct image comparison, it is straightforward to incorporate context dependent tolerances and attributes for features, e.g., pad location, type, and size. It is also easier to compensate for global distortion in the test image (e.g., scale and skew) because the inverse of the distortion function is applied to a relatively small set of feature variables, e.g., location, instead of the entire reference image. Finally, this method is relatively insensitive to local distortion and vagaries that can cause false alarms with direct image comparison. For example, irrelevant differences on the edges of traces or displacement of traces by a few pixels will not be flagged as errors (unless the minimum spacing rule is violated).

This method is a major improvement over design rule approaches because it can detect missing features and extraneous circuitization that looks like good features. In addition, unlike most design rule approaches, this method is not limited to verifying just minimum trace width and spacing; it can also verify pads, maximum trace width, and various trace connections, as well as detect isolated blobs, voids, etc. The new method is also capable of handling complex circuit features and circuit vagaries that can cause false alarms with design rule checkers. Finally, it can readily accommodate changes in circuit features and design rules that often require modification of inspection algorithms in design rule checkers.

3. Inspection Algorithms

In general, the generic method can be applied to detecting and verifying the shape and size of a large class of feature types, including spacings, holes, lines, angles, corners, triangles, rectangles, octagons, and composites of these basic shapes. Using this approach we have developed algorithms for verifying trace width, minimum spacing, the position, size, and shape of pads (or holes), trace and feature interconnections, etc. We make this concrete by describing algorithms for verifying minimum trace width, minimum spacing, and pad integrity:

Algorithm for verifying minimum trace width

This algorithm can: detect all instances of local trace width less than a programmable minimum; detect missing traces and a variety of spurious connections between traces, pads, and isolated blobs; detect voids in traces. It is based on two observations: First, the skeleton of connected traces contains only 2-joints while broken traces, i.e., open circuits, typically generate 1-joints. Second, contraction can be used to break regions corresponding to traces into two or more disconnected regions if the width of the trace is less than a specified number of pixels (implied by the design rule for minimum trace width).

Algorithm for Minimum Trace Width Verification

- Step 1: Contract image just enough to break traces whose width is less than minimum allowed.
- Step 2: Thin contracted image to skeletonize traces whose width is less than or equal to maximum allowed.
- Step 3: Detect 1- and blob-joins in image: result is list of detected joins and their locations.
- Step 4: Compare detected join list with design list
- Result: 1-joins not in design list imply width violations; 1- and blob-joins in design list not in detected list imply missing features

The behavior of this algorithm is illustrated in Fig. 5 for the special case described there. In the contraction step, entities U, V, and W are contracted just enough to generate breaks in traces where the trace width is less than the minimum allowed. Note that region W in Fig. 5(b) has broken into the two regions W' and X', while U and V have contracted to the two regions U' and V', respectively. In the thinning step, the contracted traces are thinned just enough to produce their skeletons. This thinning does not cause the contracted regions to break up or disappear. It does, however, generate the two important feature types indicated in Fig. 5(d). (1) 1-joins wherever there exist minimum width violations; (2) blob-joins, the join points between trace skeletons and contracted pads. This does not imply, however, that there exists a one-to-one correspondence between 1-joins and width violations or between blob-joins and connections from actual traces to pads. As one might expect, all features that look something like traces, i.e., elongated features whose width is less than the minimum allowable trace width, can generate 1-joins. For example, spurious blobs or large cracks in pads can generate 1- or blob-joins. The key point is that all bad input features that "look" like traces with minimum width violations will generate 1-joins; therefore, all such features can be detected.

The feature recognition step generates the detected feature list of all the 1-joins and blob-joins in the image. If the circuit type is such that 1-joins would not be generated for a good circuit, the comparison step is not needed: 1-joins imply the existence of minimum width violations. Here, we assume that a trace is never completely missing. If a trace can be entirely missing, its absence can be inferred using a comparison step because blob-joins will be missing in the detected feature list. Circuit types for which this is true are those consisting of single width traces, all of which must end in pads or features that are roughly circular (square) and whose diameter (width) is somewhat larger than the maximum allowable trace width. If 1-joins can occur, a comparison step is necessary.

Algorithm for verifying minimum spacing

This algorithm can: detect all instances of local spacing between distinct entities less than a programmable minimum; detect hard shorts between features, e.g., traces, pads, etc. It is based on two observations: First, that typical shorts occur as bridges between traces, pads, or traces and pads. When skeletonized, these bridges generate extraneous T- and blob-joins. Second, by expanding regions, it is possible to connect two disconnected regions if the minimum distance between them is less than a specified number of pixels (implied by design rule for minimum spacing).*

Algorithm for Verifying Minimum Spacing

- Step 1: Expand image just enough to connect distinct regions whose distance apart is less than minimum allowed
- Step 2: Thin expanded image to skeletonized traces whose width is equal to the maximum allowed
- Step 3: Detect T- and blob-joins in skeletonized image: result is list of detected joins and their locations
- Step 4: Compare list of T- and blob-joins with design list
- Result: T- and blob-joins not in design list imply shorts; joins in design list not in detected list imply missing features

The behavior of this algorithm is illustrated in Fig. 6 for the special case described there. The first step in detecting spacing violations is to expand all regions just enough so that regions touch (connect) at all locations where the spacing between them is less than the minimum allowed. The result of expanding the defect image twice is shown in Fig. 6(b). Note that the bridge at c enlarges and a new bridge is formed at b; however, a bridge is not formed at a. The thinning step then skeletonizes the traces as well as all bridges with width less than or equal to fifteen pixels. The key point to observe is that spacing violations like those in Fig. 6(a) generate extraneous T-joins (or blob-joins) in the expanded and thinned image. The defect features, along with good features, can be readily detected in the feature recognition step.

Algorithm for verifying pads

Typically there are a variety of design rules for pads. For example, the design rules may constrain the size, shape, and position, as well as limit the number and type of pad-to-trace connections. The thinning operation used in trace width verification can be used to partially verify pads. In particular, the existence and position of all blob-to-trace connections can be verified. Additional thinning can also be used to detect cracks, voids, and insufficient pad area. Fig. 7 illustrates how this can be accomplished; cracks, voids, and insufficient pad area induce extraneous 0-, 1-, T-, and blob-joins. Since extraneous joins do not match the design list, the defects that induced the joins can be detected.

* The definition of distance on the discrete grid depends on the type of expansion employed. We use the distance measure implied by octagonal expansion.

Algorithm for Pad Verification

- Step 1: (using result of algorithm for verifying minimum trace width) In addition to 1- and blob-joins, detect 0- and T-joins
Result: Joins not in design list imply possibility of cracks, voids, or extraneous features
- Step 2: Thin enough to skeletonize pad if it does not contain an octagonal area of specified diameter (implied by design rule for minimum pad size)
- Step 4: Detect 0-, 1-, T-, and blob-joins: result is list of joins and their locations
Result: 0- or 1-joins not in design list imply insufficient pad area, cracks, or extraneous features; T- and blob-joins not in design list imply cracks, voids, or possibly hard shorts

4. Hardware Implementation of Basic Image Processing Operations

Contraction, expansion, thinning, and join detection are the basic image processing operations used in the inspection algorithms. These operations can be realized using sequences of so called *3 by 3 neighborhood operations* that map each pixel and its eight neighbors in a binary matrix *I* to a zero or one. If execution time is not critical, e.g., algorithm development work or analysis of only a few images per printed circuit layer, these operations can easily be implemented on a general purpose computer or a dedicated image processing system like a Vicom or De Anza. Unfortunately, such an approach is far too slow to achieve full one hundred percent inspection of typical printed circuit layers in a few minutes or less (which would require a throughput of several megapixels/sec). For example, to implement the three algorithms described in this paper using the design rules given in the examples would require about one hundred neighborhood operations per image frame. Assuming a ten by fifteen inch layer and a one mil square pixel, it would take about two hundred minutes to inspect one layer.*

Sternberg has shown how neighborhood operations can be implemented in hardware by *streaming* the elements of an image row by row through a simple *processing element* (PE) like the one shown in Fig. 8.⁹ A sequence of *p* neighborhood operations, e.g., *p* contractions, can be implemented by *pipelining* (cascading) *p* PE's as also shown in Fig. 8.

Solid state imaging devices (especially linear arrays) are ideally suited as input image sources for the pipeline architecture. Image data acquired from a linear array can be input continuously, row by row, into the pipeline. Except for a small overhead in time to fill and empty the pipeline, the throughput of the pipeline is equal to the clock rate of the image source. It is well within the state of the art to run linear arrays and this kind of pipeline architecture in the range of five to ten megapixels per second.

By using multiple, independently configured pipelines, all the inspection algorithms required for a particular application can be run in parallel. The net throughput for such a system will be limited only by the clock rate of the image source and pipeline. For example, the three algorithms described in this paper could be implemented with three independent pipelines containing a total of approximately one hundred PE's. Assuming a conservative clock rate of five megapixels per second, it would take only thirty seconds to do the bulk of the image processing! (Of course there are additional sources of overhead and time needed for the final list comparison step. However, we estimate that this time is on the same order of magnitude as that required for the neighborhood operations.)

5. Concluding Remarks

This paper described a new method for the analysis of printed circuit images. As the algorithms presented here demonstrate, the new method can detect a variety of common defect types in discrete, binary images of printed circuit features. However, the method is by no means limited to these particular algorithms or defect types. The method is a flexible analysis technique that can be applied to detecting and verifying the shape and size of a wide class of feature types, including lines, angles, corners, triangles, rectangles, octagons, and composites of these basic shapes. However, there does not exist a "magic formula" for generating an inspection algorithm as a function of feature type; this still requires ingenuity. For example, we have developed algorithms for verifying the existence and diameter of holes, checking clearance between holes and other features, handling traces and pads of different sizes, verifying that a local feature is connected to other arbitrarily distant features, etc.

In addition to flexibility this method is well suited for high speed implementation using pipelines of simple processing elements. The speed of processing is roughly proportional to the number of independent pipelines used and the number of PE's per pipe. Independent pipelines can be configured to process different image areas in parallel, to implement different algorithms in parallel on the same image area, or any combination of the two.

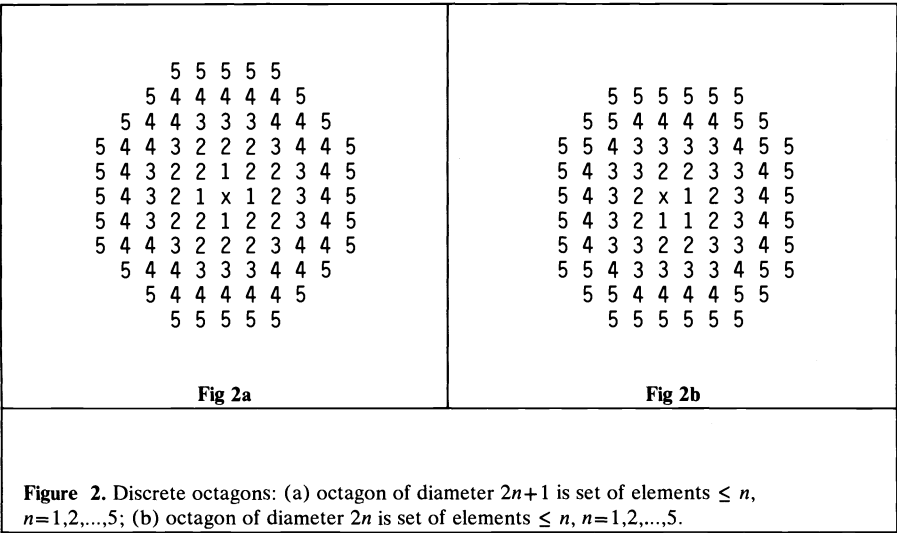
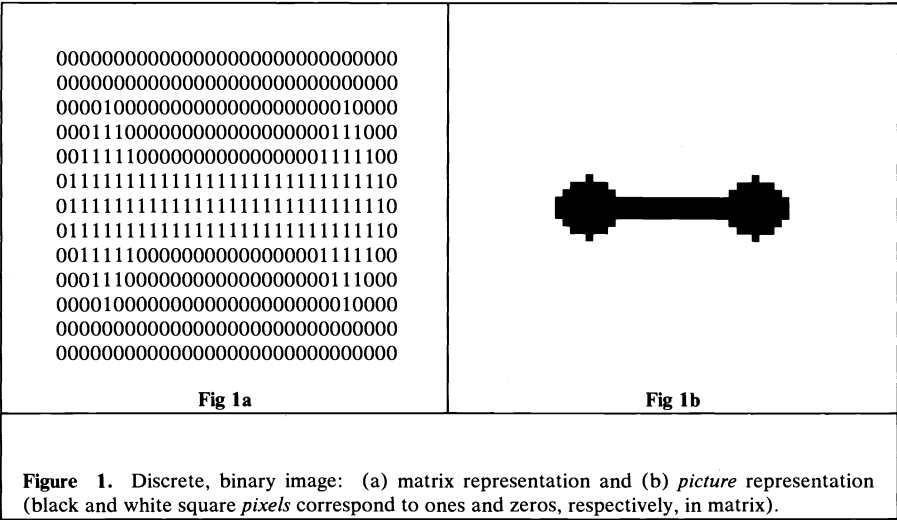
Finally, we note that a reliable, flexible, efficient, and cost effective system is more than just a set of inspection algorithms -- no matter how elegant the algorithms may be. On the research side, each inspection problem has to be thoroughly analyzed and experiments conducted to determine the limits and capabilities of this method. On the engineering side, reliability, throughput, and cost considerations will dictate the specifics of a given implementation.

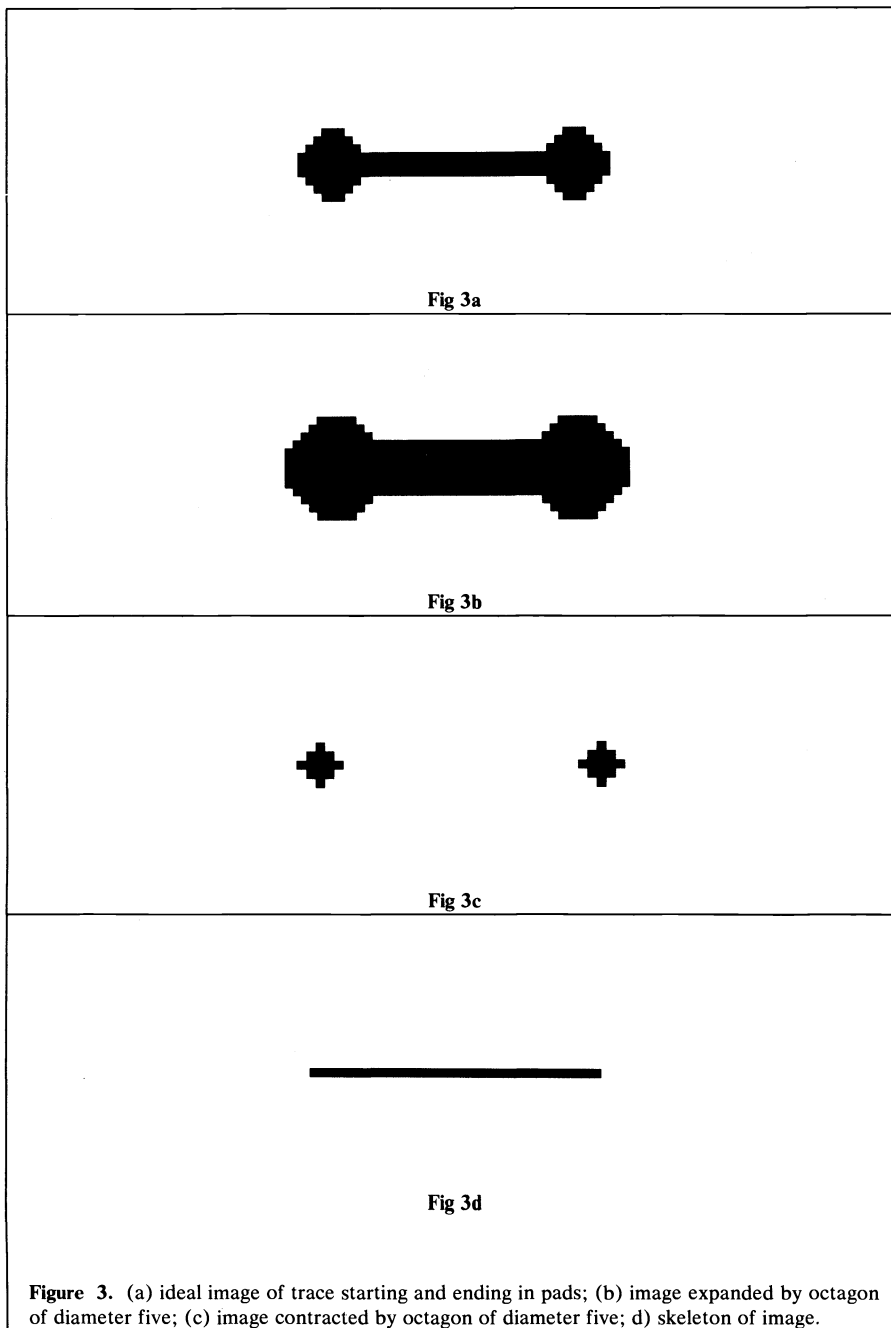
References

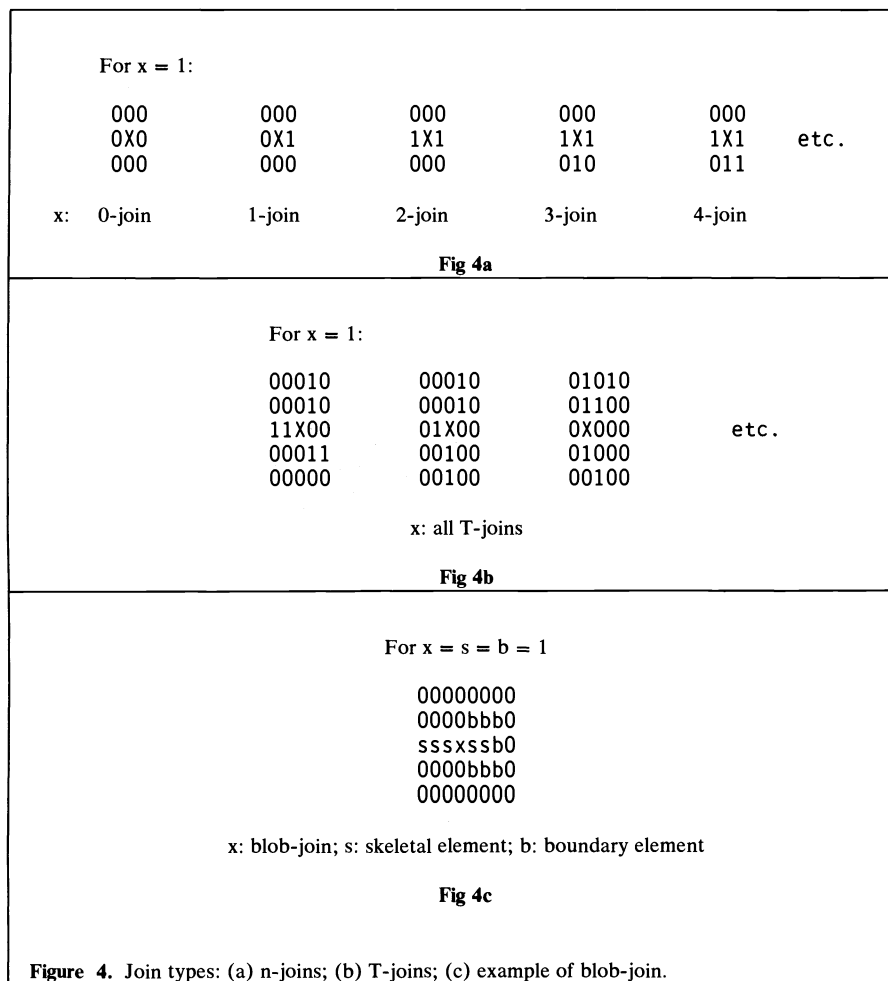
1. R. Chin, "Automated Visual Inspection: A Survey," *IEEE PAMI-4*, No. 6, November 1982, pp. 559-562.
2. R. Kruger and W. Thompson, "A Technical and Economic Assessment of Computer Vision for Industrial Inspection and Robotic Assembly," *IEEE Proc.*, Vol. 69, No. 12, December 1981, pp. 1526-1529.

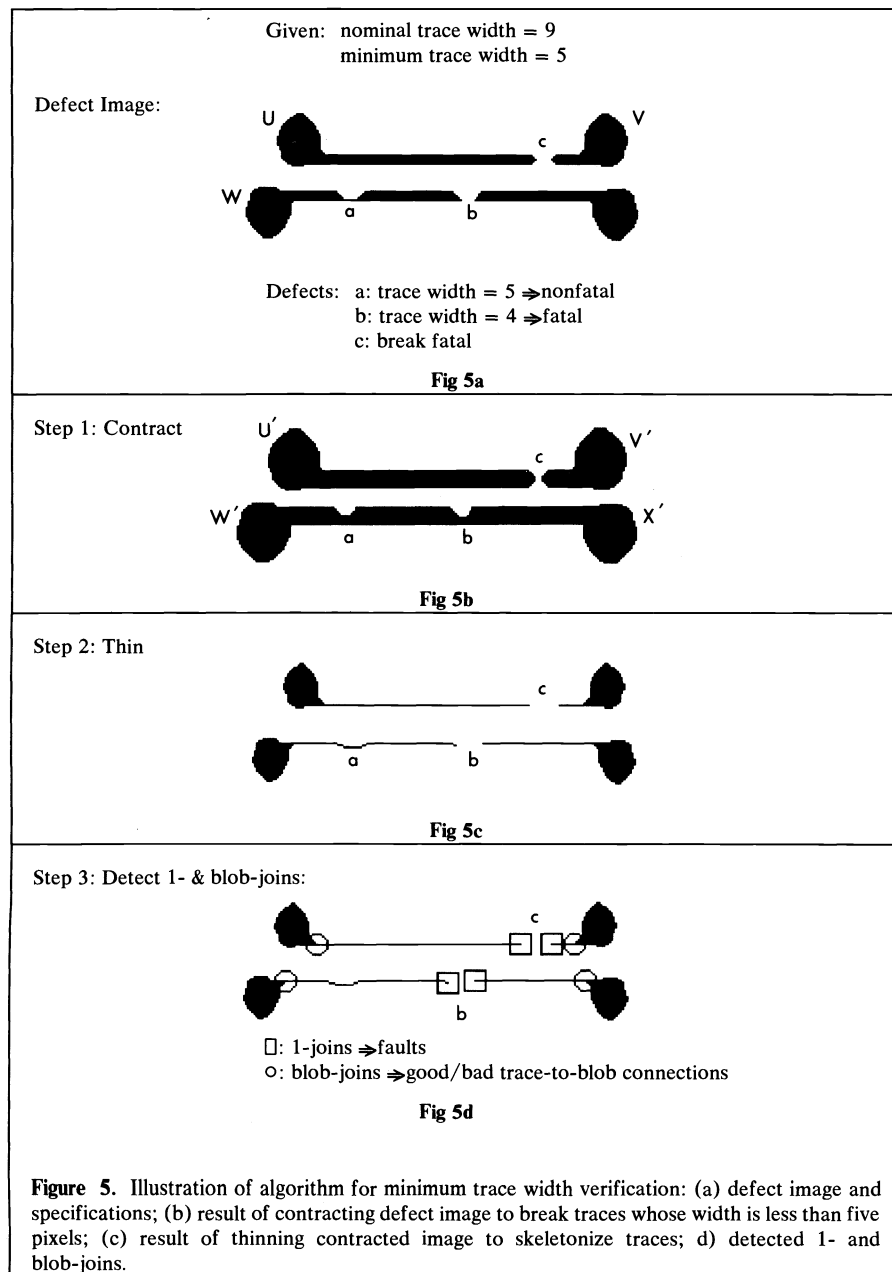
* This is based on using a Vicom Model VDC 1600 with an array processor that can do about five neighborhood operations per second on 512 by 512 images. A ten by fifteen inch layer at one square mil per pixel contains about six hundred 512 by 512 images.

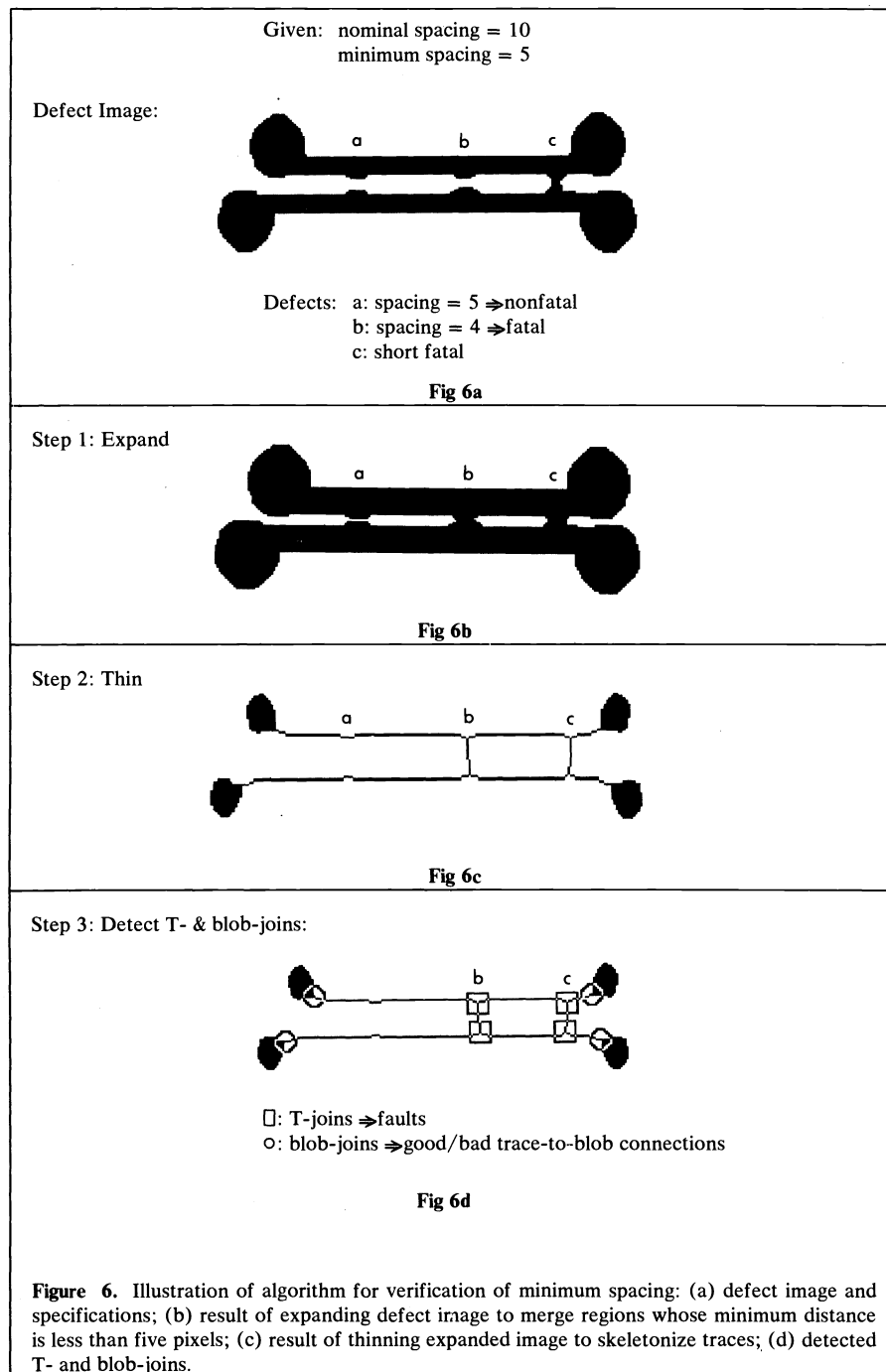
3. T. Pavlidis, *Structural Pattern Recognition*, Springer-Verlag, 1977, Chaps. 3 and 9.
4. A. Rosenfeld, and J. Pfaltz, "Distance Functions on Digital Pictures," *Pattern Recognition*, Vol. 1, 1968, pp. 33-61.
5. A. Rosenfeld, *Picture Languages*, Academic Press, 1979, Chap. 2.
6. J. Serra, *Image Analysis and Mathematical Morphology*, Academic Press, 1982, chaps. 1-8.
7. E. Abbott, et al, "Computer Algorithms for Visually Inspecting Thick Film Circuits," *Proceedings of RI/SME Conference on Applied Machine Vision*, Memphis, Tennessee, February, 1983.
8. M. Ejiri, et. al., "A Process for Detecting Defects in Complicated Patterns," *Computer Graphics and Image Processing*, Vol. 2, 1973, pp. 326-339.
9. S. Sternberg, "Biomedical Image Processing," *IEEE Computer*, January 1983, pp. 22-28.

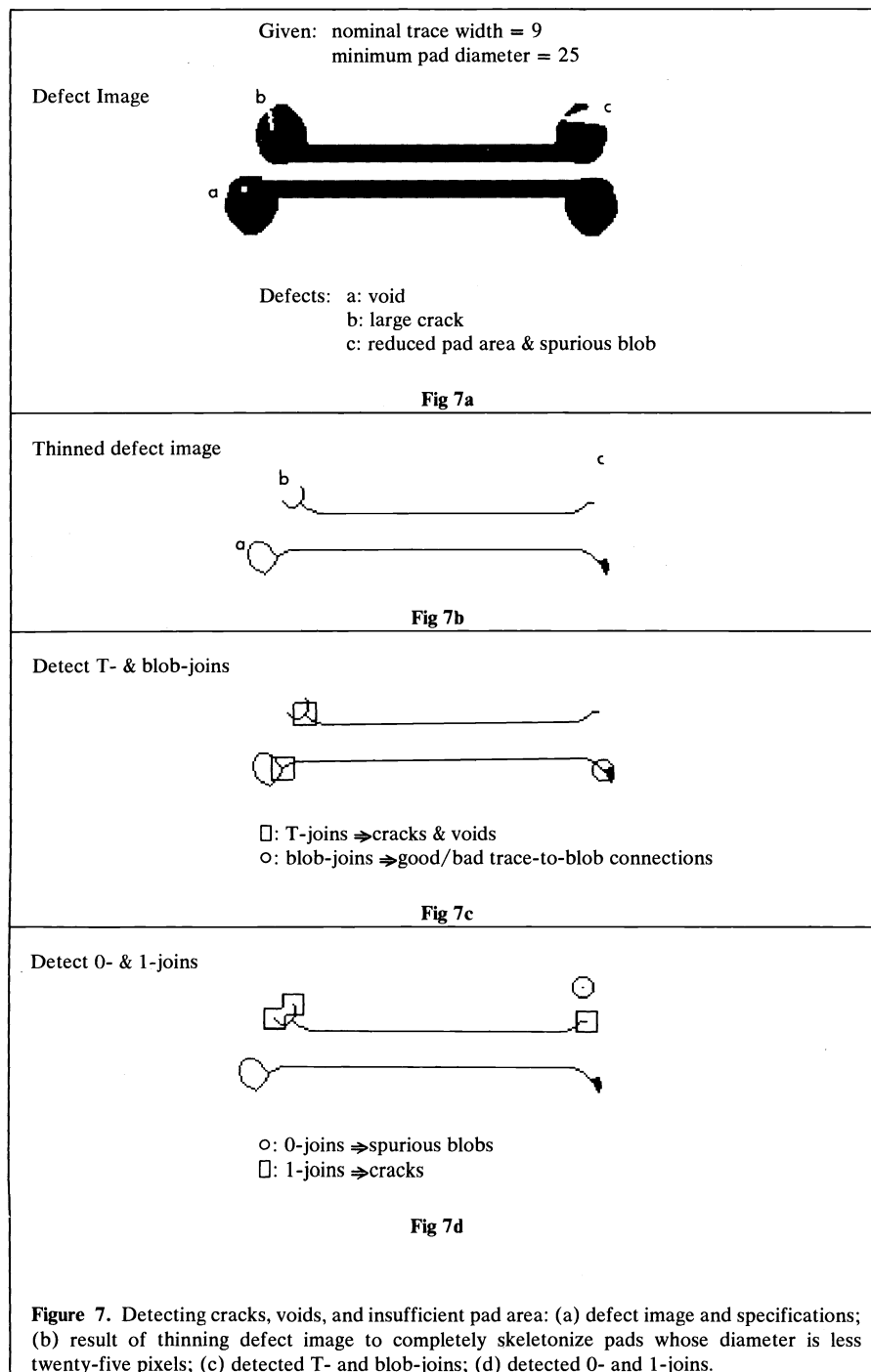












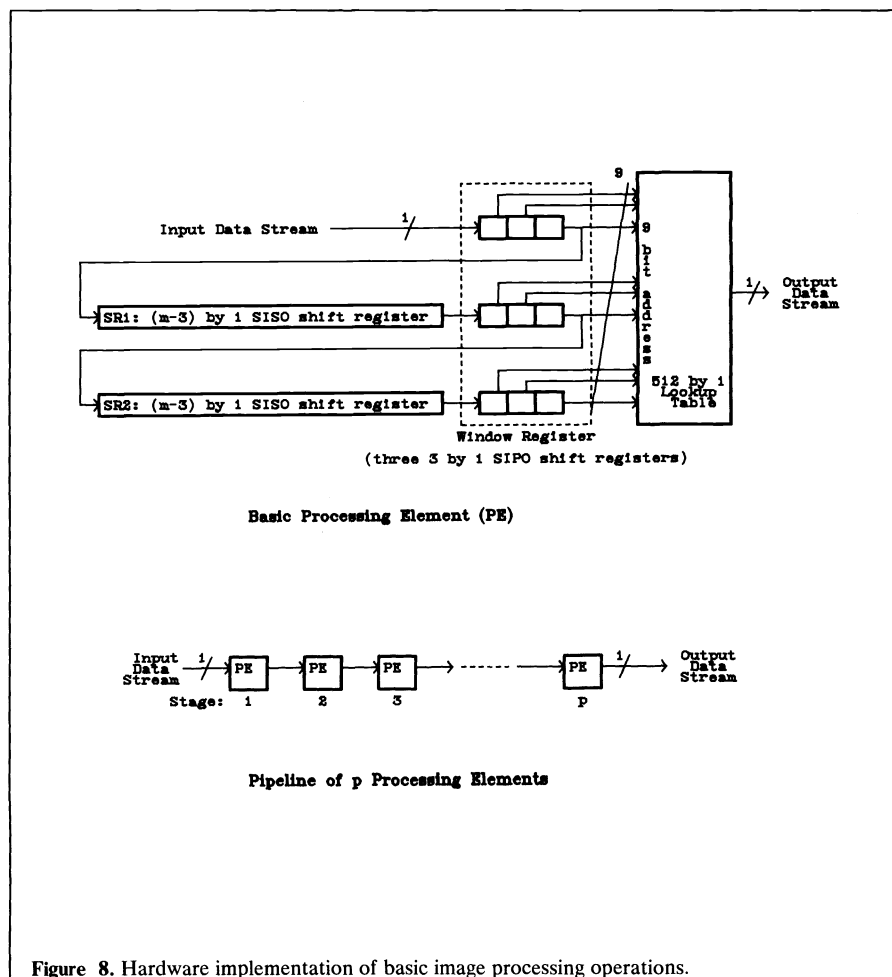


Figure 8. Hardware implementation of basic image processing operations.