Machine Vision and Applications (1989) 2:193–214

Machine Vision and Applications

An Architecture for Region Boundary Extraction in Raster Scan Images Suitable for VLSI Implementation

James M. Apffel,* K. Wayne Current, Jorge L. C. Sanz,† and Anil K. Jain
Department of Electrical Engineering and Computer Science and Computer Vision Research Laboratory, University of California, Davis, California, USA

Abstract. A novel hardware architecture for extracting region boundaries in two raster scan passes through a binary image is presented. The first pass gathers statistics regarding the size of each object contour. This information is used by the hardware to allocate dynamically off-chip memory for storage of boundary codes. In the second raster pass the same architecture constructs lists of grid-joint codes to represent the perimeter pixels of each object. These codes, referred to variously as "crack" codes or "raster-chain" codes in the literature, are later decoded by the hardware to reproduce the ordered sequence of coordinates surrounding each object. This list of coordinates is useful for a variety of shape recognition and manipulation algorithms that utilize boundary information. We present results of software simulations of the VLSI architecture, along with measurements on the coding efficiency of the basic algorithm, and estimates of the overall complexity of a proposed VLSI chip.

Key Words: segmentation, boundary-coding, raster scan, VLSI, architectures

1 Introduction

A variety of approaches to performing image segmentation has been considered, including spatial clustering, region growing, and split-and-merge algorithms (Haralick et al. 1985; Mitiche et al. 1985; Samet et al. 1982). Boundary-based segmentation methods for bitonal images are a subclass in which regions of connected 1's are separated by their contours. These algorithms typically detect edge transitions in the image and then extract from the contour parameters that are useful for subsequent processing. Since the locations of 1 and 0 transitions in a binary image contain all the information relevant to represent the objects, the extraction of these points is usually the first step in boundary-based segmentation. The transition points can be provided in a line-by-line order, as in run-length encoding techniques, or in a sequential list, as in contour following methods. A sequentially ordered list of the boundary coordinates is especially attractive because it facilitates many kinds of useful processing, such as segmentation, analysis, synthesis, and recognition. For example, individual objects can be manipulated graphically, features such as perimeters, corners, and moments can be extracted, and pattern-matching techniques can be applied (Ellis et al. 1979; Paglieroni and Jain 1985; Paglieroni and Jain 1986; Persoon and Fu 1977; Price 1984).

Historically, there have been two basic approaches to the extraction of boundary points from objects in binary images. The first uses a contour following method analogous to a person traveling the perimeter of a wall. The second uses a technique borrowed from algorithms that label objects in raster-scanned images (Milgram 1979; Sobel 1978). Before describing our chosen algorithm and architecture, we review in the following sections issues that are relevant to the hardware implementation of each method.

1.1 Boundary Following Approaches

A clear advantage of boundary following algorithms is that they are intuitively easy to understand. They also do not require complex data structures or even the storage of boundary codes, as lists of ordered coordinates can be produced directly while a contour is being tracked.

When addressing hardware issues, however, a

Address offprint requests to: James M. Apffel, SiSCAN Systems, 911 Dell Avenue, Campbell, CA 95008, USA.
* Present address: SiSCAN Systems, Campbell, CA, USA.
† Present address: IBM Almaden Research Center, San Jose, CA, USA.

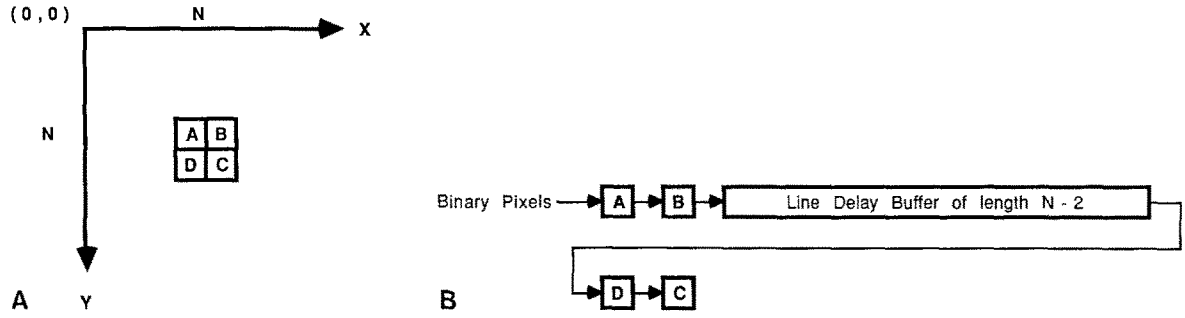


Figure 1. (A) 2 × 2 pixel window definition; (B) one line-delay buffer.

number of potential bottlenecks must be resolved. The most serious of these for VLSI implementations is the need for random access to neighboring pixels. Such access occurs during contour tracking, when the follower may need to visit pixels to the right, left, top, or bottom of the current pixel location. Since the process is "random" or irregular, it is more difficult to implement in fast, pipelined hardware than is the case for raster-scanning methods (Ruetz and Brodersen 1988). A possible solution to this problem is a memory storage structure that permits parallel transfers of two-dimensional blocks of adjacent pixels.

A second difficulty with VLSI contour followers is their requirement to mark boundaries that have already been tracked. A marking is necessary because after it closes a contour, a follower must not trace that contour again. To achieve this marking, one can increase the bit depth of the original image and insert a special tag into locations that have already been visited. Unfortunately, this then creates the need for *write-access* to a deeper image, which becomes impractical for processing large images at real-time rates. Another approach, suggested in Morrin (1976), would be to *peel* continuously the boundary of an object until it is erased from the image. Of course, this would be prohibitively expensive in processing time for large objects. Still another approach, used in D'Amato (1982), and Ruetz (Tokyo 1986), would be to save polygonal information about tracked objects. This could be in the form of a convex hull or enclosing rectangle, but if too simple, it could lead to missed contours, especially for cases with nested objects and holes.

1.2 Raster-Scan Methods

An alternative approach to following the boundaries of objects is to collect edge information in a raster-scan fashion (Cederberg 1979; Chakravarty 1981; Danielsson 1982; Lindgren 1983; Luhscher and Beddoes 1989; Milgram 1979; Sobel 1978). Algorithms of this type use pixel information from previous scan lines to determine connectivity and thereby label all contours. Raster-scan methods generally require more sophisticated "bookkeeping" than direct contour tracking, but they are attractive for hardware implementation because of inherent regularities in data manipulation. These algorithms also couple well with the current generation of solid-state sensing devices, such as charge-coupled devices, as well as with raster display devices, since these provide and utilize pixels in a raster-scan order.

Raster-scan boundary coding shares most of the hardware advantages of general raster-scan algorithms. Pixel access is regular and sequential, which facilitates straightforward data paths and faster clock speeds. By including the now common line-delay unit, illustrated in Figure 1, a fixed size, small neighborhood of pixels is examined each clock cycle and an update decision is made. This processing lends itself readily to table look-up methods and pipelined implementations (Milgram 1979; Sobel 1978). The source image is often passed through without a need to add markers to any image, thus overcoming a common drawback of boundary followers.

The most serious disadvantage of raster-scan algorithms is their use of fairly complicated data structures. Since object information is accumulated line by line and for many objects in parallel, a good deal of bookkeeping is necessary to track contours. The need to manipulate and organize these data structures presents formidable challenges in designing a VLSI implementation. Most raster-scan algorithms either do not address these difficulties or presume a hardware intelligence and organization that is rarely achievable at the chip level.

In this paper we describe a new architecture suitable for VLSI integrated circuit implementation that will produce the edge coordinates of each object in a binary image. The proposed chip set is called the "RASBOC," an acronym for the raster

scan boundary encoder. It is designed to scan an image from top to bottom, left to right, and to encode the boundary points of each object in such a way that sequentially ordered lists of edge coordinates can be produced (Apffel et al., Nov. 1987). In the next section we discuss the manipulation and storage of the dynamic data structures generated during raster scanning. Our algorithm takes advantage of a statistic-gathering initial pass through the image prior to the scanning for boundary segmentation. By the use of this dual-scan approach the hardware is able to simplify the storage of contour information and to make most efficient use of available memory.

2 Raster-Scan Grid-Joint Coding

When choosing to store ordered lists of boundary coordinates, one quickly realizes that every individual coordinate need not be stored (Freeman 1974). Rather, since a continuous contour can be defined, an initial reference location can first be found and then a path of simple directional codes can be saved. These codes can be independent of the contour following mechanism and should be more compact than storing the individual coordinates.

In order to store the coordinate list of each object efficiently, we have chosen a coding method that is well understood and widely applicable. A variation of the generalized chain-code, called the crack code or grid-joint code, is used to specify the path of the contour from a starting point (Cederberg 1979; Danielsson 1982; Rosenfeld 1970). Instead of the conventional 8-connected grid-intersect chain-code of Freeman (1961), a 4-connected grid-joint code (GJC) is used to represent the *joints* or *cracks* between pixel grid points. Crack following was considered in the late 1960s (Rosenfeld 1970), and raster-scan coding schemes were described in the 1970s (Cederberg 1979; Danielsson 1982; Milgram 1979). From the large set of boundary representations grid-joint coding was chosen for our internal data structure. This lossless code is useful for a number of reasons.

First, the GJC guarantees closure of the boundary code for all possible contours of binary objects, including those that are ambiguous or disallowed in other coding schemes. The GJC accurately encodes isolated pixels, single-pixel wide curves, multiply-connected junctions, and nested holes or objects. Algorithms described in Caponetti et al. (1984), Chakravarty (1981), Juven et al. (1982), and Kundu et al. (1985) can break down under these nonideal conditions. Coding with GJCs does not require a preprocessing thinning operation or a subsequent marking of the source image, as suggested in Atkinson et al. (1985), Landy et al. (1985), Morrin (1976), Pavlidas et al. (1982), and Suzuki et al. (1985).

Second, the GJC preserves enough information of the boundary to be useful. The original binary object can be readily reconstructed from it, and object features can be extracted from the code list.

A third advantage of the GJCs are their compactness. When oriented in the raster-scan direction, they require one bit per grid-joint (Danielsson 1982), as explained subsequently. While enabling reproduction of a contour without distortion, the code requires on the order of one bit per contour pixel. In Section 4.1 we present measurements of the coding efficiency of the GJC.

A final advantage of the GJC is that it is straightforward to generate in hardware using raster-scan component-labeling methods. Detection of joint connectivity and orientation requires only local window information provided in a pipelined manner. Thus, coding/decoding rules can be stored in short ROM tables.

Sample binary images and the GJCs for their objects are shown in Figure 2. Each small square represents a pixel that contains either a 0 or a 1. The intent is to encode the boundaries of connected 1's, and the arrows along the image *cracks* represent grid-joint directional vectors. The two types of arrows surrounding objects correspond to Cederberg's *right-handed* and *left-handed* contours (Cederberg 1979; Danielsson 1982). Each term derives from the crack-side for which the first link of a GJC list encodes.

A first link is begun when scanning finds a 1 pixel that is not 8-connected to the line above (Cederberg's MAXPOINT). The grid-joint arrow, which proceeds down the crack immediately to the left of the MAXPOINT pixel, starts a *left-handed* contour or queue. The arrow that proceeds across the top of the MAXPOINT pixel starts a *right-handed* contour or queue list. Since these "subcontours" are now directed, there are only two possible orientations for an arrow to point: down or to the right. If the hardware keeps track of which type of directed contour the arrow belongs to, then a single bit can be used to encode its orientation. Our architecture thus uses a GJC of "0" to indicate a horizontal crack and a GJC of "1" to indicate a vertical crack. The direction of the crack arrow is determined by the raster-scan directions.

Perhaps the greatest drawback of the GJC is that it is primitive. A more complex code could have been chosen to generate fewer bits per pixel or, say, per line segment, but at an increase in surrounding hardware complexity. We felt that a simple internal

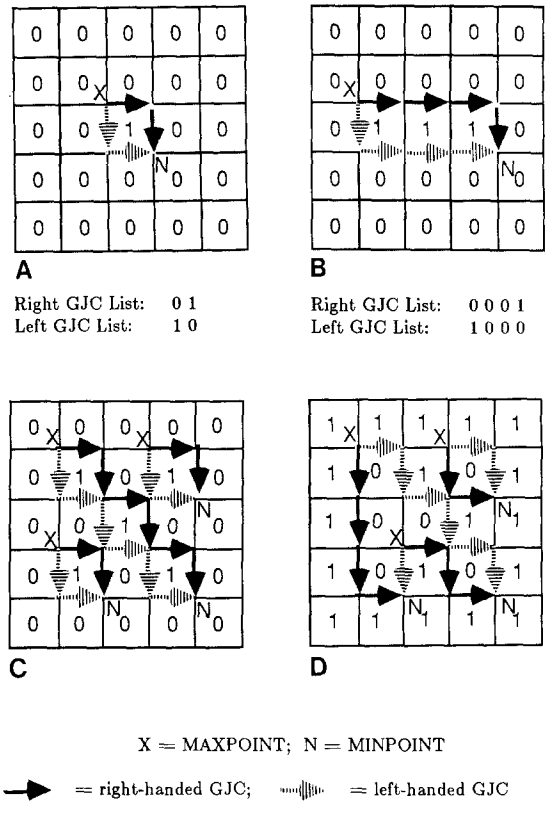


Figure 2. Grid-joint coding examples.

code representation was more practical when the alternatives meant significant increases in design time and hardware complexity. Once the GJCs for a contour are extracted, subsequent processing can compact or recode the boundary further (Danielsson 1982).

2.1. Generation of Grid-Joint Codes

The RASBOC chip set generates GJCs using the coding algorithm described in Cederberg (1979) and Danielsson (1982). In this method a GJC list is built for each boundary through operations similar to raster-scan component-labeling (Milgram 1979; Sobel 1978). During the encode operation each image pixel in a 2 by 2 window is examined, and a code bit is chosen from the rules given in Table 1 (Apffel 1987; Danielsson 1982). For example, when a new object pixel is encountered during the raster scan, the right- and left-hand sides of the outer subcontour are assigned a unique queue number or *label*. This label is used in subsequent references to the evolving contour code list. At the time of the label assignment the X and Y coordinates of the first pixel are saved. Later in the image, when a pair of subcontours previously labeled are found to join, the two are marked as connected by saving an equivalent pair (Cederberg's MINPOINT). In this manner each GJC is linked by either its position or a label to form a string that codes the entire contour.

At each pixel clock cycle up to five operations can take place as indicated in Table 1. First, if the window is empty, no code update is made. If the 1 pixels specify such, a left and/or a right GJC will be appended to the proper GJC list. If a MAXPOINT is detected, the location will be saved in the MAXPOINT memory, and a new set of queue labels will be generated. And finally, if a junction of grid-joints is detected, an equivalent pair of queue labels must be saved.

To give an example of how a pixel is coded from the rules of Table 1, consider the third 2 by 2 pixel window from the top of the table ($ABCD$ = 0010). This is the condition that occurs when a new object pixel is encountered during a raster scan, at a MAXPOINT. There are no "prior link" arrows for pixels A, B, or D because there were no cracks that were labeled in the previous line. The x's in columns 2 and 4 indicate that a GJC is to be added to both the right- and left-hand queues for pixel C. Column 3 shows that the right-hand GJC is horizontal (0), and column 5 shows that the left-hand GJC is vertical (1). These arrows are drawn in the pixel window of the first column. The last column of the table indicates that the label for these queues is a new one. That is, two new queues are to be started, and the X, Y location of pixel C is to be saved.

2.2 Storage of Grid-Joint Codes

The accumulation of contour codes presents formidable challenges for hardware implementation, yet is often ignored in the literature. This can be a serious omission because the data bandwidth from the coder to the memory can be the main bottleneck limiting performance. That is, images can be scanned and contour codes can be generated much faster than the codes can be arranged and stored in a usable structure. Also, the practical limits of code memory can place undesirable restrictions on the complexity of the input image. For example, one might restrict the input image to contain a specific type of graphic data, such as typewritten text. Thus, with an a priori knowledge of the size of the text font and assumptions about the amount of noise in the image, one would have an upper limit on the length of any *single* contour code list. This fact could be exploited when choosing a size of memory to assign to a particular GJC list.

In order to support a general set of input images, RASBOC is designed with virtually no limit on the length of any *single* contour code list. This is accomplished by repartitioning code storage, depend-

Table 1. Rules for generating grid-joint codes

2 × 2 Pixel Window	Append Right Q	Direction (GJC)	Append Left Q	Direction (GJC)	Equivalent Queues	New Queues
0 0 0 0						
0 0 → 1 ⬇ 0	x	1				
0 0 ➡ 0 ⇓ 1	x	0	x	1		x
0 0 → ➡ 1 1	x	0				
0 ↓ 1 ⇒ 0 0			x	0		
0 ↓ 1 → ⇒ 1 ⬇ 0	x	1	x	0	x	x
0 ↓ 1 0 ⇓ 1			x	1		
0 ↓ 1 → 1 1					x	
1 ↓ 0 → 0 0					x	
1 ↓ 0 1 ⬇ 0	x	1				
1 ↓ 0 → ➡ 0 ⇓ 1	x	0	x	1		
1 ↓ 0 ➡ 1 1	x	0				
1 1 → ⇒ 0 0			x	0		
1 1 ⇒ 1 ⬇ 0	x	1	x	0		x
1 1 → 0 ⇓ 1			x	1		
1 1 1 1						

↓: prior links; ⇒ left-handed links; ➡ right-handed links; x: action taken; empty box: no action taken; 0: horizontal crack code; 1: vertical crack code.

ing on contour characteristics, so that each GJC list can fit within a linear space of memory. An alternative approach would have been to implement the linked-list mechanism usually suggested for storage of variable length strings, as in LISP machines. But this would have been expensive for short strings of single-bit depth, especially with the overhead incurred in assigning new storage and reclaiming old storage (Cohen 1981; Tanaka 1985). Our dynamic reconfiguration comes at the expense of an additional pass through the source image, thus requiring either a rescan or a complete buffering of the image. This was seen as an acceptable trade-off for VLSI viability and efficient code storage because the source image is often externally buffered, and in any case the second pass can be a stage in the pipeline. To increase flexibility further, code storage takes place external to the coding chip, in the form of common RAM, so that designs can be tailored to the limits of a given application.

To facilitate repartitioning of external memory, a level of indirection is introduced, as illustrated in Figure 3. An indirect addressing unit, indexed by the label of the code list, holds a pointer to the end of the queue storing the GJC bits. After the first pass through the image, the external queue memory is repartitioned by means of adjusting each pointer. Once the memory has been repartitioned, a second raster-scan pass through the image takes place. During this pass GJCs are detected exactly as before, but now they are physically stored in the external queue memory.

2.3 Decoding of Grid-Joint Codes

After two raster scans through the binary image to build contour codes, the GJC lists can be translated into the coordinate lists. This can be done immediately or later at the request of some external processor. Decoding involves starting at the uppermost boundary point saved for each closed contour and proceeds with successive increments or decrements to the coordinates depending on the values in the GJC list. The list traversal is first down the right-hand queue and then continues up the corresponding left-hand queue. This is illustrated in the sequence of decoding cycles shown in Figure 4.

The reconstruction phase typically is more complicated than is previously described, since most objects in a binary image will be composed of multiple queues. It will often be the case that a right-hand queue will be logically linked via an equivalent label pair to a left-hand queue other than the one joined at its MAXPOINT (Cederberg 1979). Consider the sample binary image shown in Figure 5A. The

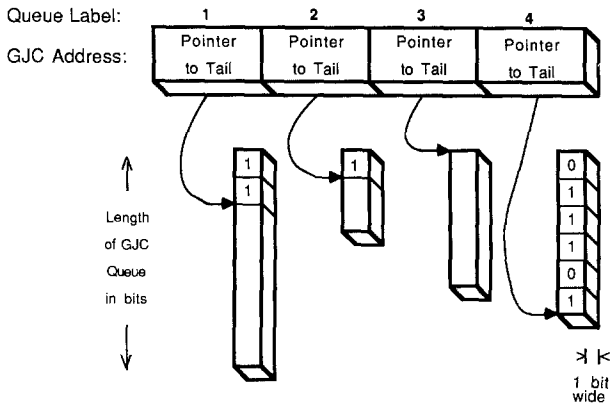


Figure 3. Indirect addressing mechanism for appending grid-joint code queues.

equivalent pair structure that would be generated for the outermost contour is shown in Figure 6. The outermost contour is composed of four pairs of left/right GJC queues.

Similar to conditions in raster scan component labeling, subsequent to the labeling of all subcontours in an image, label equivalences must be resolved to reproduce an entire contour (Milgram 1979; Sobel 1978). Unlike equivalent pair merge in component labeling, however, the equivalent pairs in our grid-joint coding scheme are unique. That is, the end of one right-hand queue will be connected

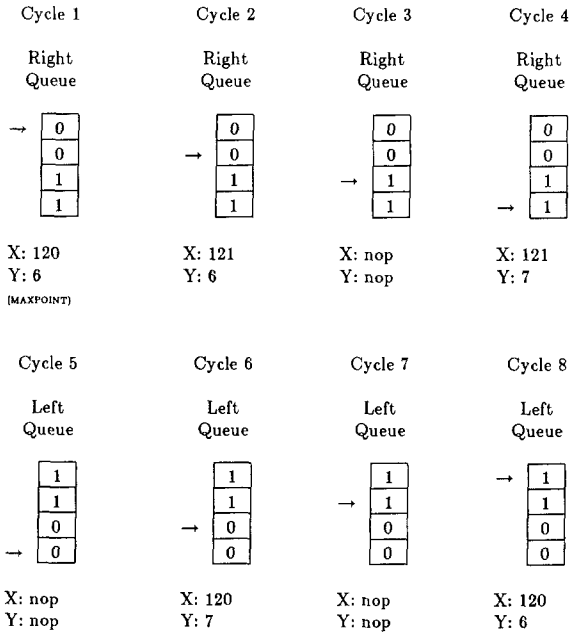


Figure 4. Sequence of GJC decoding cycles used to generate coordinate pairs: first down the right-hand queue, then up the left-hand queue.

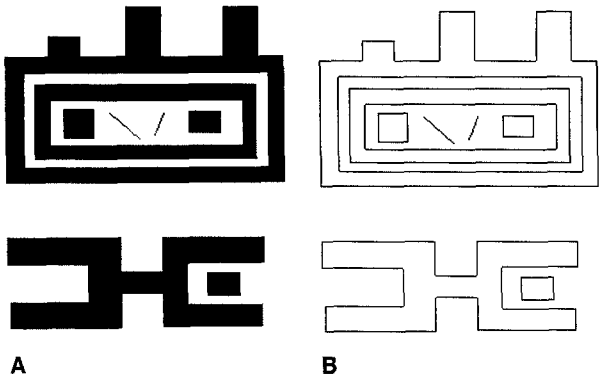


Figure 5. (A) Binary image of size 256 by 256 pixels, and (B) a plot of its coordinate lists.

to one and only one left-hand queue. This property derives from the rules for GJC generation in Table 1 and from the fact that all contours will close. It greatly simplifies the merging phase of equivalent pairs over that done for raster scan component labeling. In the latter, multiple and possibly redundant equivalences can occur for each labeled subentity, which usually means that recursive passes through the pair list are needed for merging. For grid-joint coding, on the other hand, connectivity of subcontours can be resolved in a single pass through an equivalent pair table. The table cannot contain labels with multiple equivalences, nor can it contain more than one indirection per subcontour label.

3 Architecture of the RASBOC

The block diagram of the RASBOC architecture is given in Figure 7. It includes the proposed custom VLSI boundary coder/decoder chip and the RAM blocks associated with the code storage unit. During coding the boundary coder measures the lengths of subcontours and generates their GJC bits. These bits are stored in the code storage unit, along with the starting locations of each subcontour. Also stored off-chip are the large arrays for performing the intermediate data-manipulation functions.

The input to the chip set is a binary image of up to 4K by 4K size, with pixels being fed in a raster-scan order. A single pixel enters at the top left of the coder each clock cycle. Once the pixel buffer has stored the first line of image pixels, grid-joint coding begins. As queue labels and codes are generated, they are passed to the label buffers and queue memories, respectively. As subcontours join at a junction, the associated queue labels are stored by the equivalent pair unit. The output of the chip set exits the code storage unit at its top right. The ports of the **coordinate bus** provide a sequential list of boundary points for each object in the binary image.

With the knowledge that off-chip communication is costly in terms of pin outs and time delays, we feel that this partitioning of hardware is most efficient. It delineates well-defined interfaces between the units and permits scaling for different sizes of applications. For example, while retaining the same boundary coder/decoder chip, the memory sizes in the code storage unit can be tailored to the expected complexity of the source images.

3.1 Boundary Coder/Decoder

The internal structure of the proposed boundary coder/decoder chip is shown in Figure 8. Important goals of its design include the following:

- It must produce an ordered GJC list for each object.
- It must accept pixels at real-time rates to be a useful coder.
- It will have off-chip image storage to provide for scaling.
- It also will have off-chip contour code storage.
- It will utilize pipelined operation wherever possible.
- It will provide for manipulation of dynamic data structures off-chip.

The boundary coder/decoder contains components found in most implementations of raster-scan algorithms, as well as specialized structures to handle the special features of grid-joint coding. The former include row and column position counters, an image line pixel buffer, a next label counter, and look-up tables for control. The latter include a pair of queue access controllers, and the equivalent pair unit. The major components of the coder/decoder are now described.

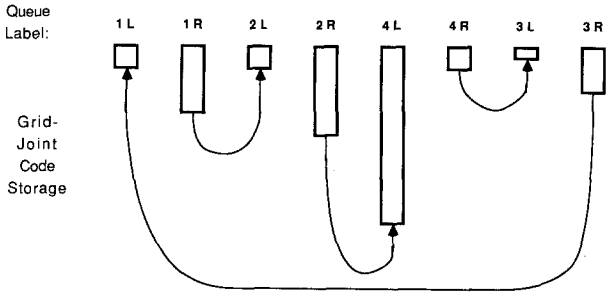


Figure 6. Storage of right- and left-hand queues with corresponding equivalences for the outer contour in Figure 5A.

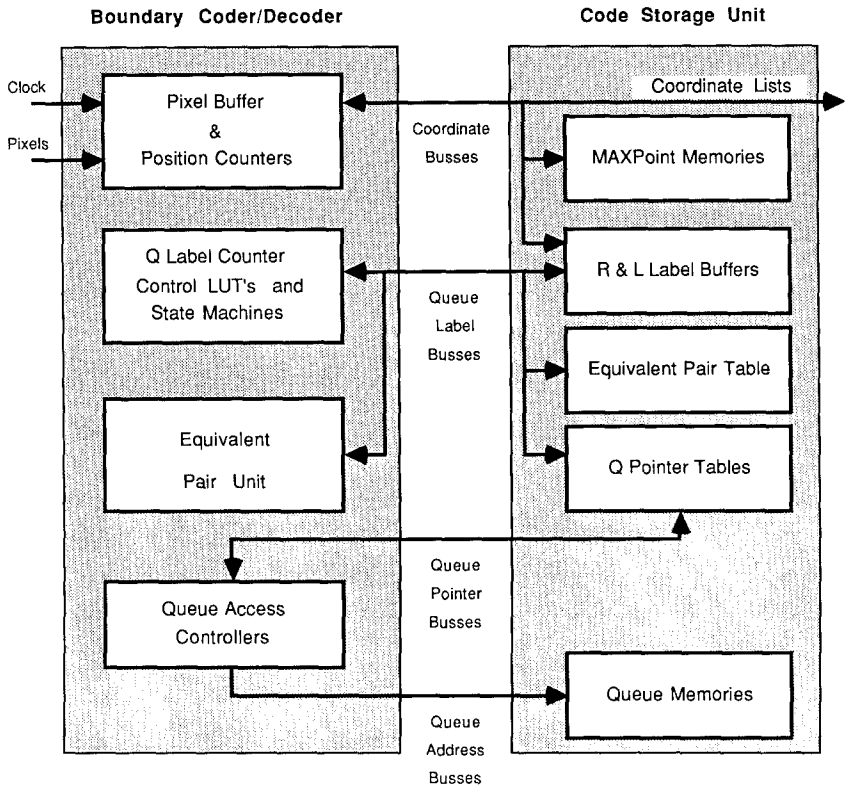


Figure 7. Block diagram of the RASBOC architecture.

Position and label counters. The **row** and **column counters** are used to maintain the current location in the image during scanning. Their contents are needed when a MAXPOINT (Cederberg 1979) is detected and saved off-chip in the MAXPOINT memory. The column counter is also used as an index into the label buffers to direct updating. Both counters are dual-ported and have inputs that are loaded in the decode phase for generating the contour coordinates.

The **next queue counter** is an up counter that generates the labels for new queues. It contains enough bits to represent a number equal to the total possible queues. This single number is used to update either or both the right- or left-hand label buses, depending on pixel connectivity in the 2 by 2 window of Table 1.

Line buffers. A common component of most hardware implementations of raster-scan image algorithms is the line-delay buffer. Its purpose is to provide parallel access to data from one or more previous image lines so that local window information can be examined. The boundary encoder requires three delay units, termed the **P, R,** and **L buffers.** The on-chip **P buffer** stores a line of image pixels and provides access to a 2 by 2 local neighborhood. The **R** and **L buffers,** which are located in the code storage unit, each store one line of queue labels and provide access to the assignment of GJC labels immediately above the current pixel.

Queue access controllers. There are two queue access controllers in the coder/decoder: one for left-handed queues and one for right-handed queues. A queue access controller (QAC) is an address generation unit for accessing GJCs in the code storage unit. We provide two separate controllers to allow for simultaneous updating of right- and left-handed GJC queues.

A block diagram of a single QAC and its associated RAM is given in Figure 9A. It contains a pointer table made up of standard RAM. The address is provided by the current label counter attached to the label bus, either the right or left, depending on which orientation the QAC controls. The data input/output of the pointer table is provided by an internal path loop. This loop contains the pointer and feedback registers, the last pointer latch, and the adder. These are used to keep track of starting and ending queue locations, to increment and decrement pointers, and to partition the external address block in a single pass through the table.

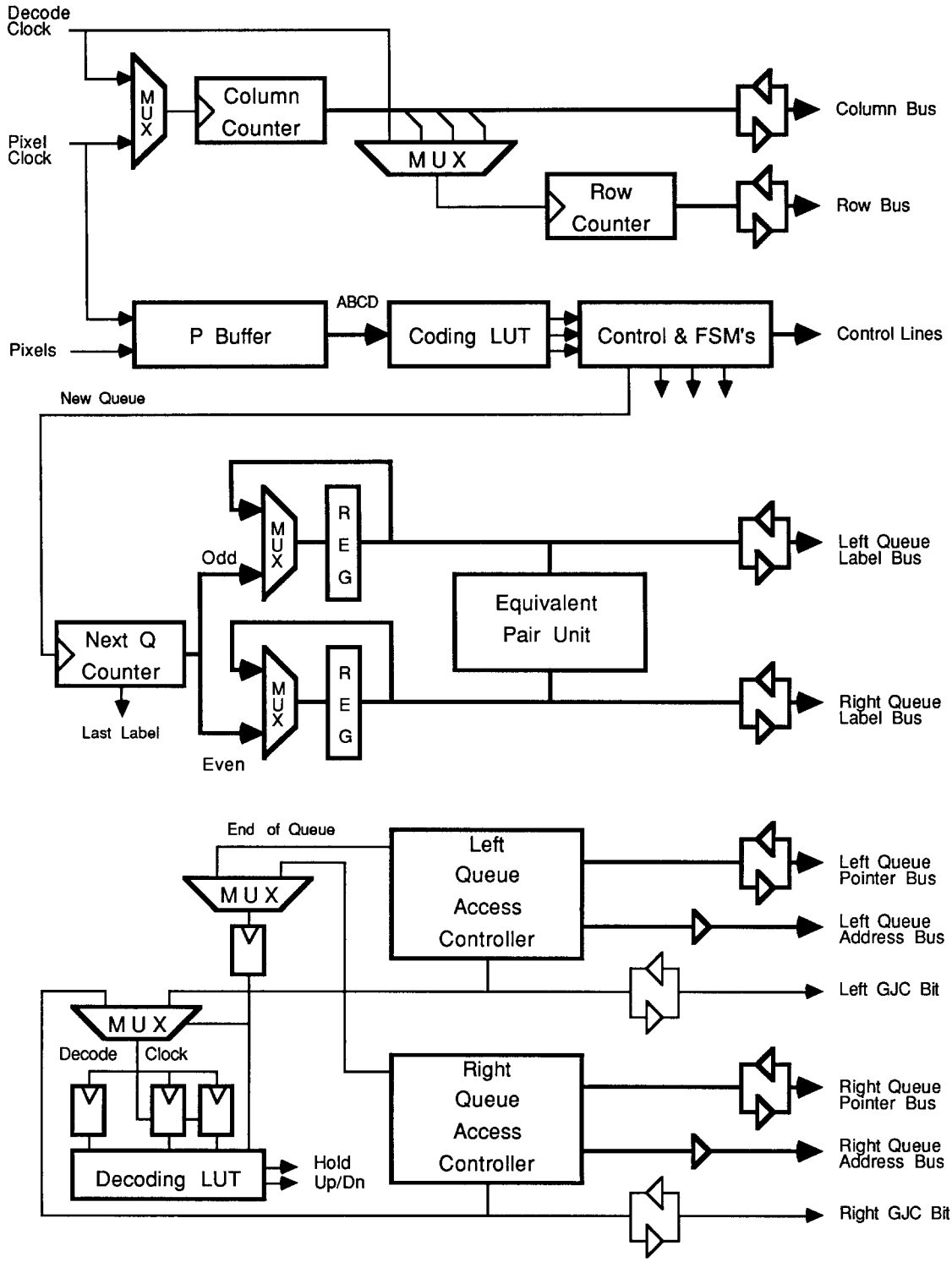


Figure 8. Block diagram of the boundary encoder/decoder.

Equivalent pair unit. The equivalent pair unit and its associated RAM in Figure 10 store the labels of all logically linked queues. The unit contains an equivalent pair table made up of standard RAM and a few other registers. As equivalent pairs are detected, the left-hand queue label from the **L buffer** is written into the table using the right-hand queue label as an address. Thus, a write operation is per-

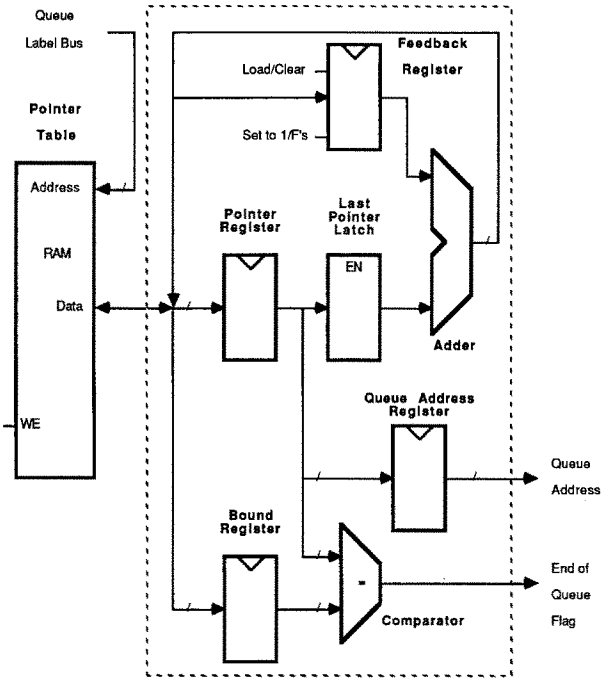


Figure 9. A queue access controller.

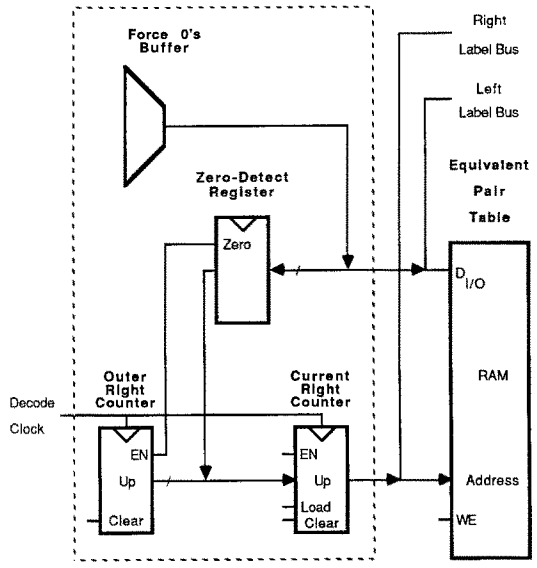


Figure 10. The equivalent pair unit.

formed whenever equivalent queues are detected as per the rules given in Table 1. Since each label pair is unique, every valid cell in the equivalent pair table is written once. During decoding the linked list of labels is traversed and used to determine the selection of particular GJC queues to be decoded from the queue memories.

Control unit. A complete description of the control section for the RASBOC would include a discussion of bus timing, off-chip communication protocol, update control, and error-flagging (Apffel 1987). We limit the discussion here to that portion of control specific to the raster-scan coding/decoding algorithm.

The pixel-window look-up table (LUT), shown in Figure 11a, generates the flags for updating of GJCs in the queue memories. It accepts four pixels as input and produces six control lines that initiate the functions specified in Table 1.

The portion of the control unit that is specific to the decoding algorithm is shown in Figure 11b. The decoder LUT generates the flags for proper updating of output coordinates in the row and column registers. It accepts four inputs: the current GJC bit, the previous GJC, a status bit indicating whether the queue being decoded is right- or left-handed, and a status bit indicating whether this GJC is the first of a queue. Its output lines determine the direction the coordinate registers are updated, as per the rules shown in Table 2. The LUT examines each pair of GJC bits in order to handle cases where the list follows cracks around a corner. For these, rather than output the same coordinate twice, the no-output flag is driven to indicate a skip in the generation of coordinates.

3.2 Phases of Operation

To implement boundary coding and decoding, the RASBOC architecture cycles through five separate phases, which are termed the **setup, queue count, queue allocate, queue build,** and **decode** phases. A flowchart of the operations performed in each of these phases is shown in Figure 12 and explained here.

The **setup** phase is used to initialize the internal registers and status flags of the chip in preparation for a new binary image. The address pointers of each QAC are cleared to zero for use in the next phase, and image-specific parameters are set.

The **queue count** phase refers to the initial pass through the binary image in which the size of each contour is determined. No GJCs are built in this phase because queue storage for them has yet to be allocated. Instead, the label of each boundary is determined using the rules in Table 1, and a running count of the size of each queue is kept by each QAC. Illustrated in Figure 13A is a portion of the QAC after the first pass through the image of Figure 5A. Each cell value represents the total size in bits of the addressed queue. Equivalent queues are also detected in this phase and corresponding label pairs are saved in an equivalent pair table. Figure 14

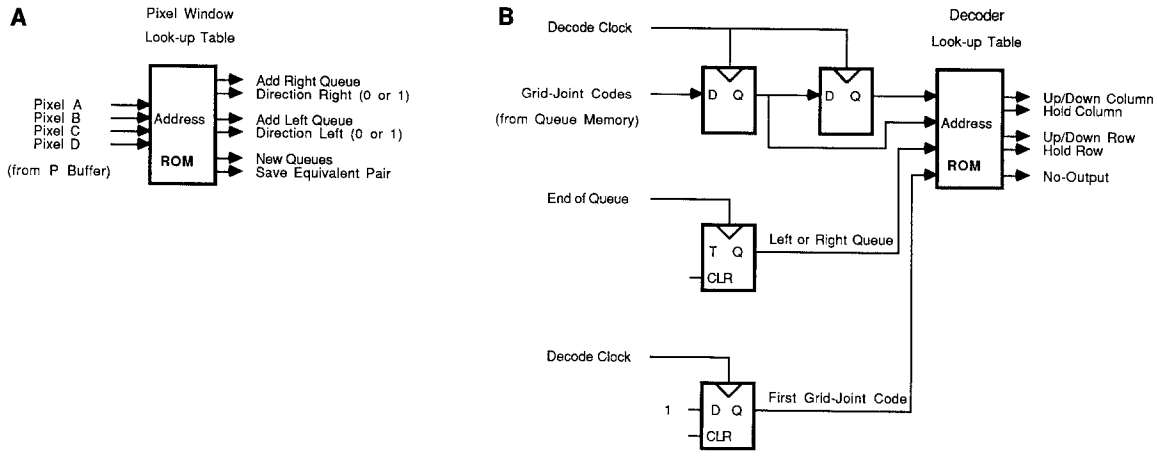


Figure 11. (A) The pixel-window look-up table, and (B) the decoder look-up table.

shows a section of this equivalent pair table for the image in Figure 5A.

The **queue allocate** phase begins following the first pass through the image during which all possible queues were detected and measured. It remains to partition the code memory for receipt of incoming GJCs. Pointer adjustment operations take place in the QACs to set up external memory for use in the next phase. Illustrated in Figure 13B is a portion of the QAC following the repartitioning pass for the image in Figure 5A. Starting at address 0, each successive pointer was adjusted to point to the first available memory location in linear space, given the queue sizes counted in Figure 13A.

Following repartitioning of the available queue memory, the **queue build** phase begins. This consists of a second raster scan through the image with processing identical to the count phase. In this pass, however, the counts in the pointer tables directly address the queue memories, and these force the storage of GJC bits at the tails of the corresponding queues. Since equivalent pairs were collected in the count phase, there is no need to store them a second time.

Finally, the **decode** phase is entered when a list of boundary coordinates is to be translated from the GJC queues stored in the code storage unit. The list is initiated by fetching the starting location of the

Table 2. Decoding rules for updating the coordinate registers during the decoding phase. The four input signals determine which, if, any, of the registers to modify in the current decode cycle

First GJC in Sequence?	Queue Direction	Current GJC	Last GJC	Row Register	Column Register
No	Right	0	0	Hold	Count up
No	Right	0	1	Count up	Count up
No	Right	1	0	No output	No output
No	Right	1	1	Count up	Hold
No	Left	0	0	Hold	Count down
No	Left	0	1	Count down	Count down
No	Left	1	0	No output	No output
No	Left	1	1	Count down	Hold
Yes	Right	0	0	Invalid	Invalid
Yes	Right	0	1	No output	No output
Yes	Right	1	0	Count up	Count down
Yes	Right	1	1	Invalid	Invalid
Yes	Left	0	0	Invalid	Invalid
Yes	Left	0	1	No output	No output
Yes	Left	1	0	Count down	Count up
Yes	Left	1	1	Invalid	Invalid

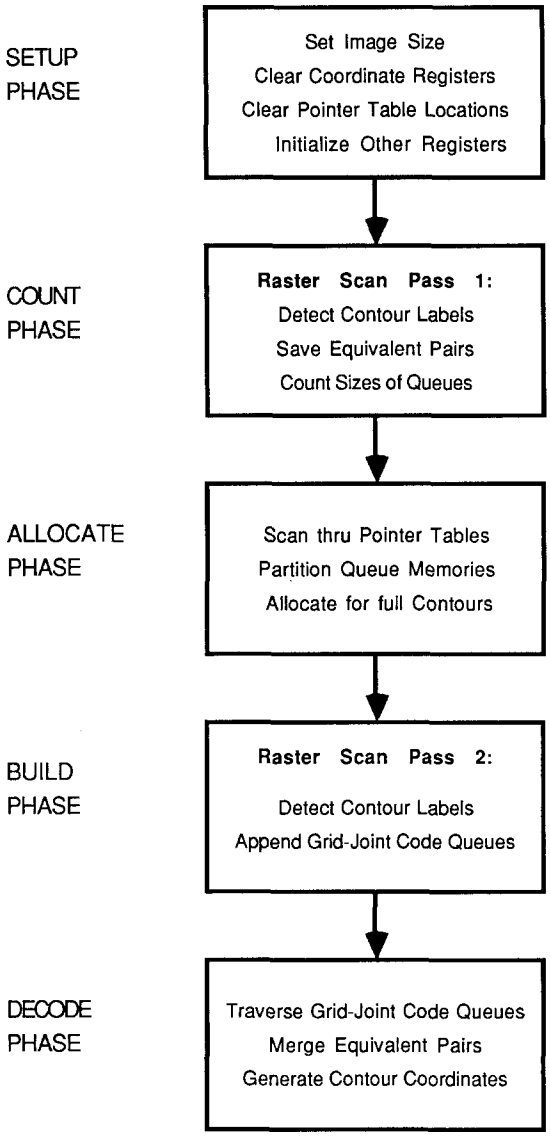


Figure 12. Flowchart of the RASBOC hardware phases.

contour from the MAXPOINT memories, and then setting this as the current contour coordinate. With each subsequent decode cycle, GJCs are read from the queue memories by the QACs. In this phase, note, the traversals of GJC queues must be directional: right-handed queues are followed from MAXPOINT to end, and left-handed queues are followed from end to MAXPOINT. This requires that each QAC be able to detect the crossing of a partition boundary and switch direction.

As code bits are read from each queue memory, they are fed to the decoding portion of the control unit. This circuitry in turn directs the coordinate registers to generate the next position in the coordinate list. Depending on the decoding rules given in Table 2, the current X and Y locations are modified to reflect the next coordinate in the contour list.

The complete contour for a given object can consist of multiple logically linked queues. When a contour is being decoded, therefore, the end of one queue must be linked to the end of the next equivalent queue. This linkage is not done physically but, rather, temporally by generating the *complete* list of coordinate pairs in sequential order. The orientation of this order is dependent on whether the contour is an interior or exterior boundary; the former is oriented counterclockwise, and the latter is oriented clockwise. The traversing of the linked list of equivalent queues is performed by the equivalent pair unit.

4 Simulation Results

The RASBOC architecture and coding/decoding algorithms were functionally simulated in software on a workstation. The programs were written in C and

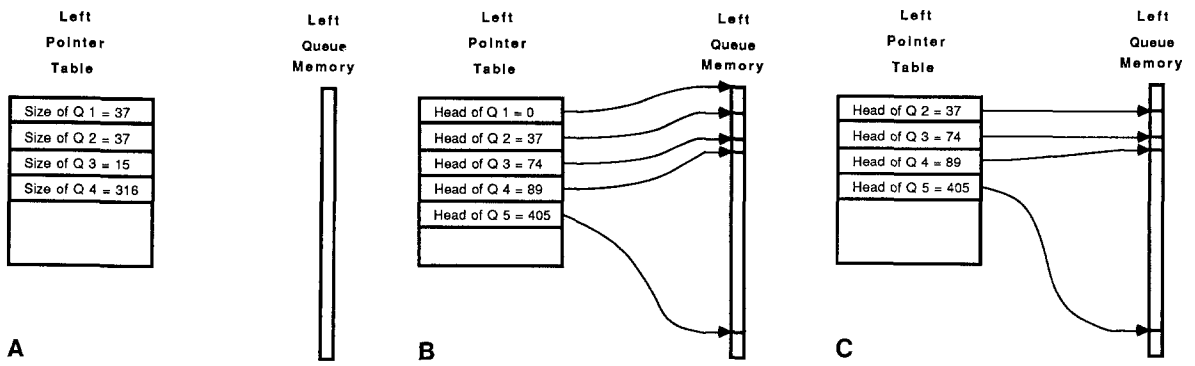


Figure 13. The left-hand QAC for the outer contour in Figure 5a: (A) after the queue count phase, (B) after the queue allocate phase, and (C) after the queue build phase.

Right Queue Label:	1	2	3	4
Equivalent Left Queue:	2	4	1	3

Figure 14. The equivalent pair table for the outer contour in Figure 5A.

were designed to mimic the hardware functions described in the previous section. No operation was implemented without concern for its possible realization in hardware. Thus, procedures were written to perform such functions as line-buffers, shift-registers, LUTs, counters, queue traversers and mergers, and equivalence tables. In addition, testing of status bits and Boolean operations were performed directly rather than using higher levels of abstraction, in order to emulate more closely the hardware architecture. Beyond simulating the hardware algorithms, the program also collects useful statistics, generates various intermediate data forms, permits interactive verification of results on the screen, and produces color images useful for understanding the operations.

A wide variety of test images was run through the RASBOC simulator. The Committee International Communique on Telegraph and Telephone (CCITT) images analyzed in the next section were the largest (approximately 1700 by 2000 pixels). But to begin with a simpler example, refer back to the 256 by 256 binary image in Figure 5A. This image contains inner and outer contours, single-pixel wide lines, and a number of equivalent queue pairs. The GJCs produced by the simulator are shown in Figure 15 for the first five queue pairs. Left-hand queue 1 encodes the left-hand side of the center "smokestack" at the top of the image; the edge is a downward leading line of 37 pixels ("1"s) starting at (117, 0). Right-hand queue 1 codes the top and right edges of the center smokestack, along with the line between the center and right stacks; it has 113 links starting at (117, 0). Left-hand queue 2 is of similar length and codes the left-hand side of the rightmost smokestack.

The GJCs are translated later into coordinate lists, as illustrated in Figure 16, depending on the connectivity indicated in the equivalent pair table (Figure 14). The partial list of the coordinate pairs for contour 1 in Figure 16 corresponds to the 801 pixel outer boundary of the smokestack object in Figure 5A. Finally, the lists of coordinates are plotted by the simulator in Figure 5b to verify that the coding/decoding sequence is accurate. Shown in Figure 5B is a plot of the coordinate lists generated for the binary image of Figure 5A. Similarly, shown in Figures 17B through 19B are plots of contours alongside their corresponding source images. Information is also given on the number of contours and codes determined. Each plot provides the same visual information that an edge extraction operation would produce, but in this case the RASBOC has extracted not merely the edge, but has also generated the list of coordinates tracing that edge.

4.1 Coding Efficiency

In order to measure the compression efficiency of the grid-joint coding method, we ran the RASBOC simulator on a set of standard bitonal images from the CCITT. Recall that each contour in an image is represented by a starting (X, Y) location followed by a string of GJC bits. The number of bits required for the X and Y coordinates is, of course, $\log_2 W$ and $\log_2 L$, where W equals the image width in pixels and L equals the image length in lines. If we compress the list of GJCs by run-length encoding each separate run of 0's and 1's, we can measure the information content in the strings.

The entropy of the runs of GJCs for an L by W image is given by:

$$\mathbf{H}_k = -\sum_{\text{all } i} P_{i,k} \log_2(P_{i,k}) \text{ bits/run,}$$

for i = length of run

where

$$P_{i,k} = \frac{\text{number of runs of } k \text{ with length } i}{N_k}$$

and

N_k = total number of k runs,

for k = 0's and 1's runs

Once the entropy is found, we can calculate **B**, the average number of code bits per pixel, as follows:

$$\mathbf{B} = \frac{|H_0|N_0 + |H_1|N_1 + (\text{no. of contours})[\log_2 L + \log_2 W]}{L * W} \text{ bits/pixel}$$

which includes the bits to encode the starting coordinates of each contour. The reciprocal of **B** gives the maximum compression rate.

Histograms of the lengths of GJCs for CCITT images 1, 2, 3, and 5 are given in Figures 20 through

```
Left Queue  1 has 37 links beginning at (column, row) = ( 117,   0):

  1111111111111111111111111111111111111

Right Queue 1 has 113 links beginning at (column, row) = ( 117,   0):

  00000000000000000000000000011111111111111111111111111111111111100000
  00000000000000000000000000000000000000000

Left Queue  2 has 37 links beginning at (column, row) = ( 193,   0):

  1111111111111111111111111111111111111

Right Queue 2 has 182 links beginning at (column, row) = ( 193,   0):

  00000000000000000000000000011111111111111111111111111111111111100000
  00000000000000011111111111111111111111111111111111111111111111111111
  11111111111111111111111111111111111111111

Left Queue  3 has 15 links beginning at (column, row) = (  56,  22):

  111111111111111

Right Queue 3 has 76 links beginning at (column, row) = (  56,  22):

  00000000000000000000000001111111111111100000000000000000000000000000
  000000

Left Queue  4 has 316 links beginning at (column, row) = (  22,  37):

  11111111111111111111111111111111111111111111111111111111111111111111
  11111111111111111111111110000000000000000000000000000000000000000000
  00000000000000000000000000000000000000000000000000000000000000000000
  00000000000000000000000000000000000000000000000000000000000000000000
  00000000000000000000000000000000

Right Queue 4 has 34 links beginning at (column, row) = (  22,  37):

  0000000000000000000000000000000000

Left Queue  5 has 263 links beginning at (column, row) = (  37,  50):

  00000000000000000000000000000000000000000000000000000000000000000000
  00000000000000000000000000000000000000000000000000000000000000000000
  00000000000000000000000000000000000000000000000001111111111111111111
  11111111111111111111111111111111111111111111111111111

Right Queue 5 has 263 links beginning at (column, row) = (  37,  50):

  11111111111111111111111111111111111111111111111111111111111111111111
  11000000000000000000000000000000000000000000000000000000000000000000
  00000000000000000000000000000000000000000000000000000000000000000000
  00000000000000000000000000000000000000000000000000000000000000000000
```

Figure 15. GJCs for image in Figure 5A.

23. Note that most of the runs of GJCs are very short, indicating that at this high of image resolution, there are few long straight lines. Of the tens of thousands of coding runs, the majority are under 40 bits in length. Thus, at the high resolution of the CCITT images, there are few long horizontal or vertical lines. Most of the runs can be encoded from a small set of symbols.

Table 3 provides the compression statistics for running the CCITT images through the RASBOC simulator. If we compare the maximum compression rates to published values for standard run-length encoding, we see that grid-joint coding is more efficient for these images. As reviewed in Jain (1988), run-length encoding of CCITT image 1 provides a maximum compression rate of 18.0, compared to 29.6 for grid-joint coding. For CCITT image 2, run-length encoding provides a rate of 21.4, compared to 70.3 for grid-joint coding. The relative values illustrate the importance of simple contour-based segmentation. The results in the table also show that grid-joint coding is better suited to line drawings with relative few contours. As the number of contours increases, the contribution of encoding

```
Contour 1 has 801 Coordinate pairs:
( 117,  0) ( 118,  0) ( 119,  0) ( 120,  0) ( 121,  0) ( 122,  0)
( 123,  0) ( 124,  0) ( 125,  0) ( 126,  0) ( 127,  0) ( 128,  0)
( 129,  0) ( 130,  0) ( 131,  0) ( 132,  0) ( 133,  0) ( 134,  0)
( 135,  0) ( 136,  0) ( 137,  0) ( 138,  0) ( 139,  0) ( 140,  0)
( 141,  0) ( 142,  0) ( 143,  0) ( 144,  0) ( 144,  1) ( 144,  2)
( 144,  3) ( 144,  4) ( 144,  5) ( 144,  6) ( 144,  7) ( 144,  8)
( 144,  9) ( 144, 10) ( 144, 11) ( 144, 12) ( 144, 13) ( 144, 14)
( 144, 15) ( 144, 16) ( 144, 17) ( 144, 18) ( 144, 19) ( 144, 20)
( 144, 21) ( 144, 22) ( 144, 23) ( 144, 24) ( 144, 25) ( 144, 26)
( 144, 27) ( 144, 28) ( 144, 29) ( 144, 30) ( 144, 31) ( 144, 32)
( 144, 33) ( 144, 34) ( 144, 35) ( 144, 36) ( 145, 37) ( 146, 37)
( 147, 37) ( 148, 37) ( 149, 37) ( 150, 37) ( 151, 37) ( 152, 37)
( 153, 37) ( 154, 37) ( 155, 37) ( 156, 37) ( 157, 37) ( 158, 37)
( 159, 37) ( 160, 37) ( 161, 37) ( 162, 37) ( 163, 37) ( 164, 37)
( 165, 37) ( 166, 37) ( 167, 37) ( 168, 37) ( 169, 37) ( 170, 37)
( 171, 37) ( 172, 37) ( 173, 37) ( 174, 37) ( 175, 37) ( 176, 37)
( 177, 37) ( 178, 37) ( 179, 37) ( 180, 37) ( 181, 37) ( 182, 37)
( 183, 37) ( 184, 37) ( 185, 37) ( 186, 37) ( 187, 37) ( 188, 37)
( 189, 37) ( 190, 37) ( 191, 37) ( 192, 37) ( 193, 36) ( 193, 35)
( 193, 34) ( 193, 33) ( 193, 32) ( 193, 31) ( 193, 30) ( 193, 29)
( 193, 28) ( 193, 27) ( 193, 26) ( 193, 25) ( 193, 24) ( 193, 23)
( 193, 22) ( 193, 21) ( 193, 20) ( 193, 19) ( 193, 18) ( 193, 17)
( 193, 16) ( 193, 15) ( 193, 14) ( 193, 13) ( 193, 12) ( 193, 11)
( 193, 10) ( 193,  9) ( 193,  8) ( 193,  7) ( 193,  6) ( 193,  5)
( 193,  4) ( 193,  3) ( 193,  2) ( 193,  1) ( 193,  0) ( 194,  0)
( 195,  0) ( 196,  0) ( 197,  0) ( 198,  0) ( 199,  0) ( 200,  0)
( 201,  0) ( 202,  0) ( 203,  0) ( 204,  0) ( 205,  0) ( 206,  0)
( 207,  0) ( 208,  0) ( 209,  0) ( 210,  0) ( 211,  0) ( 212,  0)
( 213,  0) ( 214,  0) ( 215,  0) ( 216,  0) ( 217,  0) ( 218,  0)
( 219,  0) ( 220,  0) ( 220,  1) ( 220,  2) ( 220,  3) ( 220,  4)
( 220,  5) ( 220,  6) ( 220,  7) ( 220,  8) ( 220,  9) ( 220, 10)
( 220, 11) ( 220, 12) ( 220, 13) ( 220, 14) ( 220, 15) ( 220, 16)
( 220, 17) ( 220, 18) ( 220, 19) ( 220, 20) ( 220, 21) ( 220, 22)
( 220, 23) ( 220, 24) ( 220, 25) ( 220, 26) ( 220, 27) ( 220, 28)
( 220, 29) ( 220, 30) ( 220, 31) ( 220, 32) ( 220, 33) ( 220, 34)
( 220, 35) ( 220, 36) ( 221, 37) ( 222, 37) ( 223, 37) ( 224, 37)
( 225, 37) ( 226, 37) ( 227, 37) ( 228, 37) ( 229, 37) ( 230, 37)
( 231, 37) ( 232, 37) ( 233, 37) ( 234, 37) ( 235, 37) ( 236, 37)
( 237, 37) ( 238, 37) ( 239, 37) ( 240, 37) ( 240, 38) ( 240, 39)
( 240, 40) ( 240, 41) ( 240, 42) ( 240, 43) ( 240, 44) ( 240, 45)
( 240, 46) ( 240, 47) ( 240, 48) ( 240, 49) ( 240, 50) ( 240, 51)
( 240, 52) ( 240, 53) ( 240, 54) ( 240, 55) ( 240, 56) ( 240, 57)
( 240, 58) ( 240, 59) ( 240, 60) ( 240, 61) ( 240, 62) ( 240, 63)
( 240, 64) ( 240, 65) ( 240, 66) ( 240, 67) ( 240, 68) ( 240, 69)
( 240, 70) ( 240, 71) ( 240, 72) ( 240, 73) ( 240, 74) ( 240, 75)
( 240, 76) ( 240, 77) ( 240, 78) ( 240, 79) ( 240, 80) ( 240, 81)
( 240, 82) ( 240, 83) ( 240, 84) ( 240, 85) ( 240, 86) ( 240, 87)
( 240, 88) ( 240, 89) ( 240, 90) ( 240, 91) ( 240, 92) ( 240, 93)
( 240, 94) ( 240, 95) ( 240, 96) ( 240, 97) ( 240, 98) ( 240, 99)
...
```

Figure 16. Partial coordinate list of decoded contour from Figure 5A.

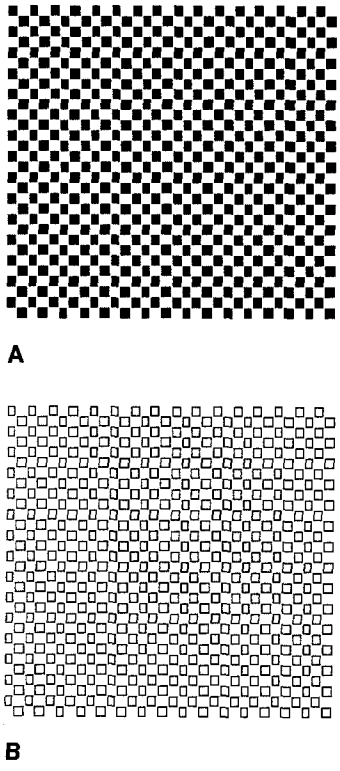


Figure 17. A binary image of size 256 by 256 pixels: (A) checkerboard without touching corners, (B) a plot of its coordinate lists. Number of queues: 960; number of GJCs: 14,430; number of contours: 480.

the starting coordinates reduces the compression rate appreciably.

5 Estimated VLSI Chip Complexity

The RASBOC architecture has been designed for realization as a chip set: a proposed new VLSI boundary encoder/decoder chip and the code storage unit of standard RAM. The chip set has been designed to encode a 4K-pixel by 4K-pixel binary image at video rates with a maximum of 64K possible queues. These quantities of data required relatively vast amounts of storage, from a VLSI design perspective. Thus, only the pixel line-delay buffer described subsequently was designed on the custom VLSI chip. All the other memory must be off-chip in the form of RAM in the code storage unit.

The code storage unit required 4.756M-bits of memory, as illustrated in Table 4. The equivalent pair table and queue pointer tables, amounting to 3M-bits, made up the majority of this RAM. The X and Y MAXPOINT memories, at 1.5M-bits, and the right- and left-hand queue memories, each a block of 64K-bit RAM, comprise the bulk of the remainder.

To implement the RASBOC architecture for such large images, 4K-element delay buffers were required. These function as shift registers, and a number of possible scheme were available: a static shift register design, a dynamic MOS shift register, a FIFO, or a RAM array with moving pointers. For

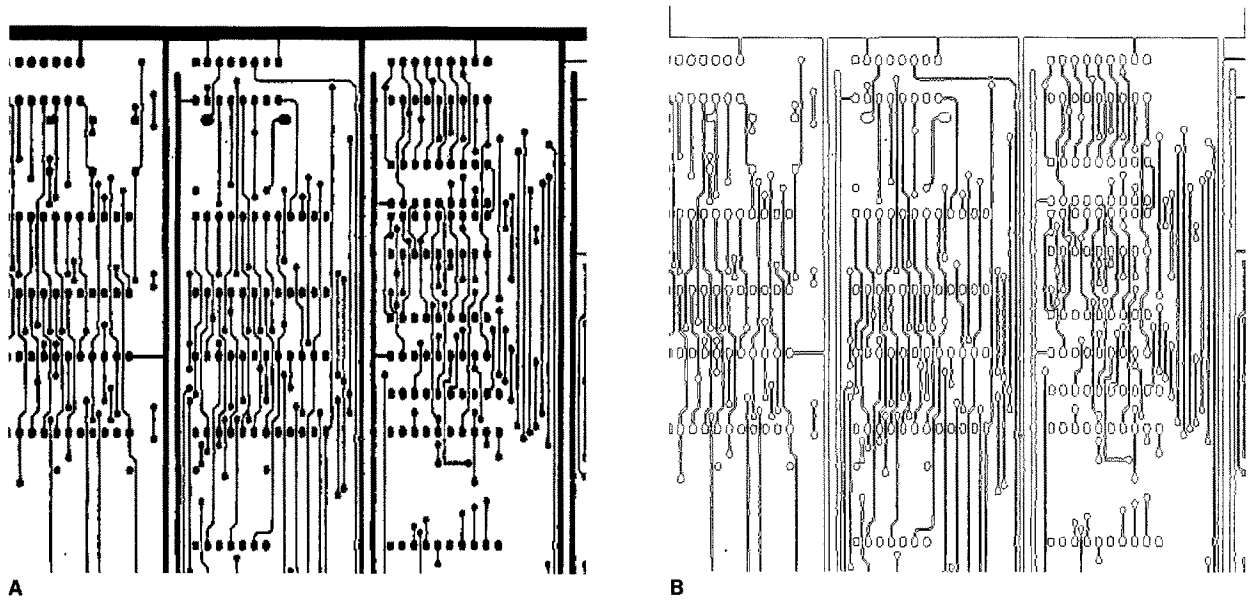


Figure 18. (A) A Binary image of size 512 by 512 pixels, (B) a plot of its coordinate lists. Number of queues: 5,660; number of GJCs: 49,422; number of contours: 271.

this application it was more area efficient to realize the pixel-buffer with a RAM array indexed by read/write bits from shift registers as described in Ruetz (May 1986). However, since the R and L label buffers each required 16-bit width to represent the 64K of possible queues, these were left off-chip as blocks of 8K-byte RAM.

Another design issue was the relative rates of coding versus decoding. The need for video rate encoding led to increased parallelism in the design, and inevitably to greater pin count. The lack of a requirement for high-speed decoding, on the other hand, permitted relaxation of time constraints in the modules associated with grid-joint decoding. For

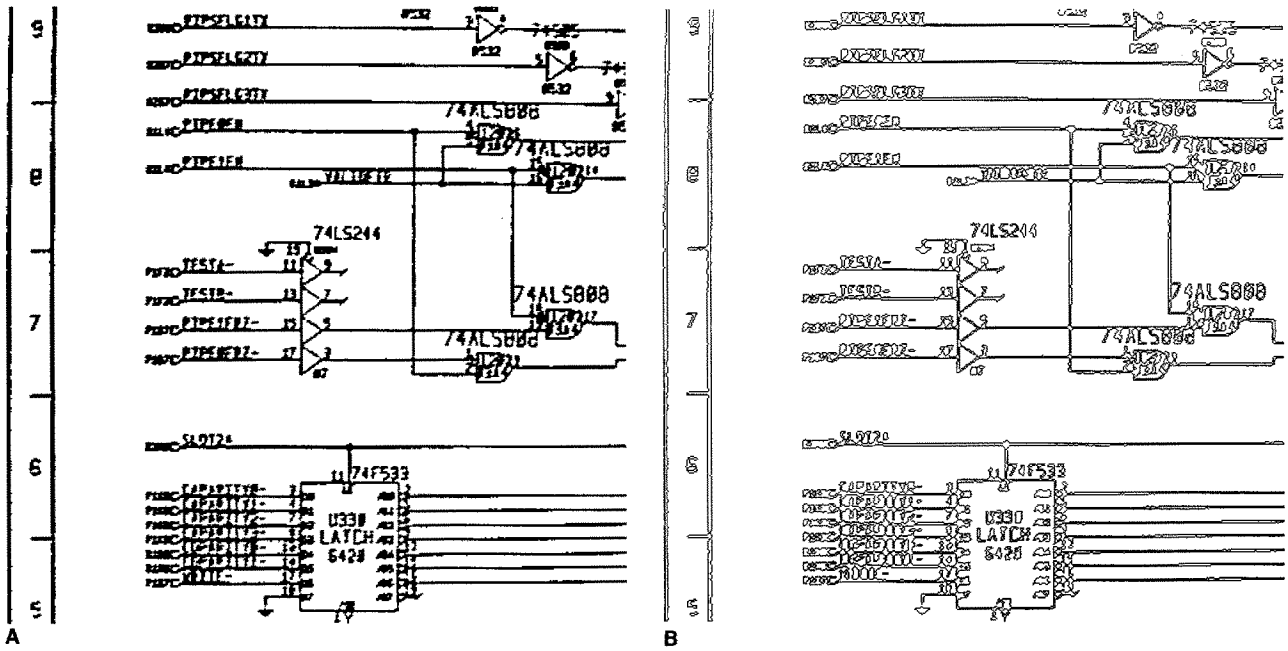
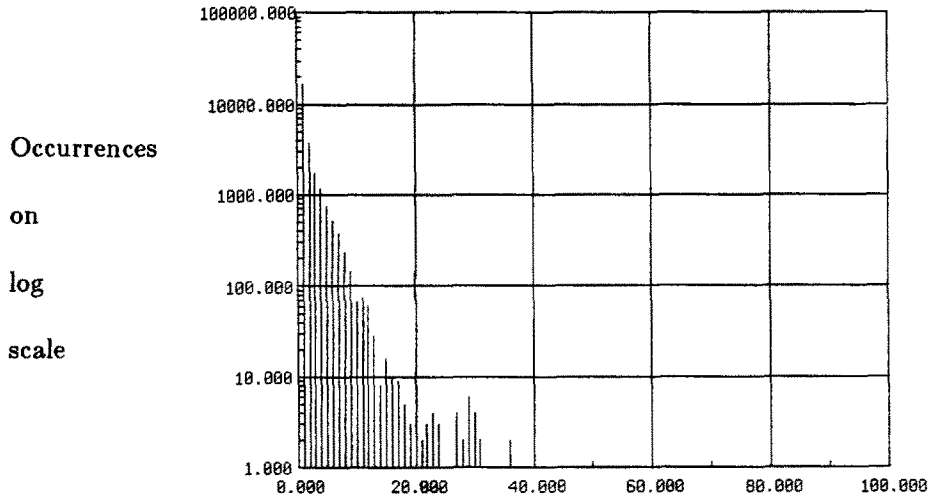


Figure 19. (A) A binary image of size 512 by 512 pixels, (B) a plot of its coordinate lists. Number of queues: 6,226; number of GJCs: 28,636; number of contours: 319.



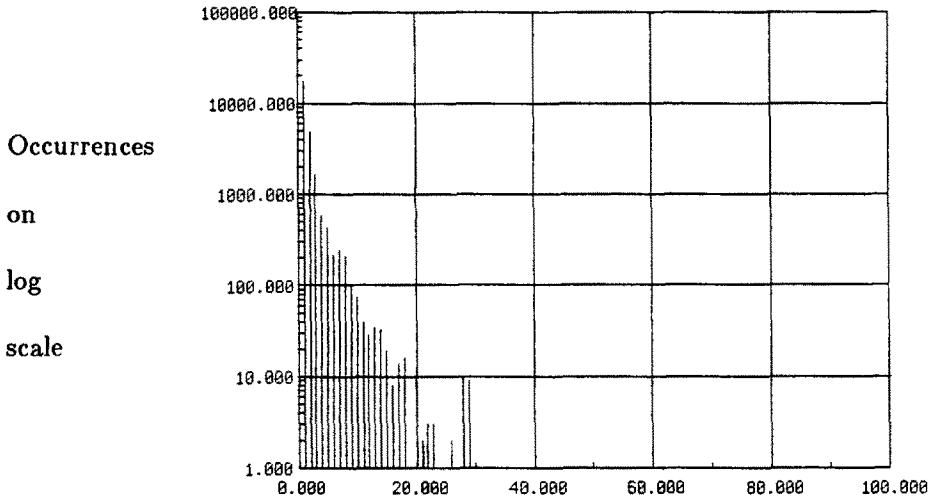


Figure 20. Histograms of runs of GJCs for CCITT image 1.

example, address comparisons in the QACs were slowed, and the duration of connectivity detection in the equivalent pair unit was relaxed.

Preliminary layout of our VLSI RASBOC boundary coder/decoder chip was done using a standard 2-micron, 2-metal CMOS technology. A 4K-pixel line-buffer designed using a 3-transistor dynamic RAM cell will occupy an area approximately 1.8 mm by 2.6 mm on the proposed chip layout. The chip will require about 14,000 additional transistors

Table 3. Compression rates for grid-joint coding the CCITT images

CCITT Image	Total Queues	Total Grid-Joint Code bits	Total Contours	Runs of 0's		Runs of 1's		Avg. bits per pixel	Maximum Compression Rate	
				Total	H_k	Total	H_k	**B**	GJC	RLC
1	25,750	96,614	1,586	25,865	1.83	25,865	1.62	0.034	29.6	18.0
2	15,790	54,466	199	14,293	1.74	14,293	1.61	0.014	70.3	21.4
3	34,550	168,268	5,828	36,377	1.55	36,509	1.69	0.060	16.7	10.6
4	46,236	180,550	9,063	38,889	1.55	38,572	1.74	0.080	12.6	9.5

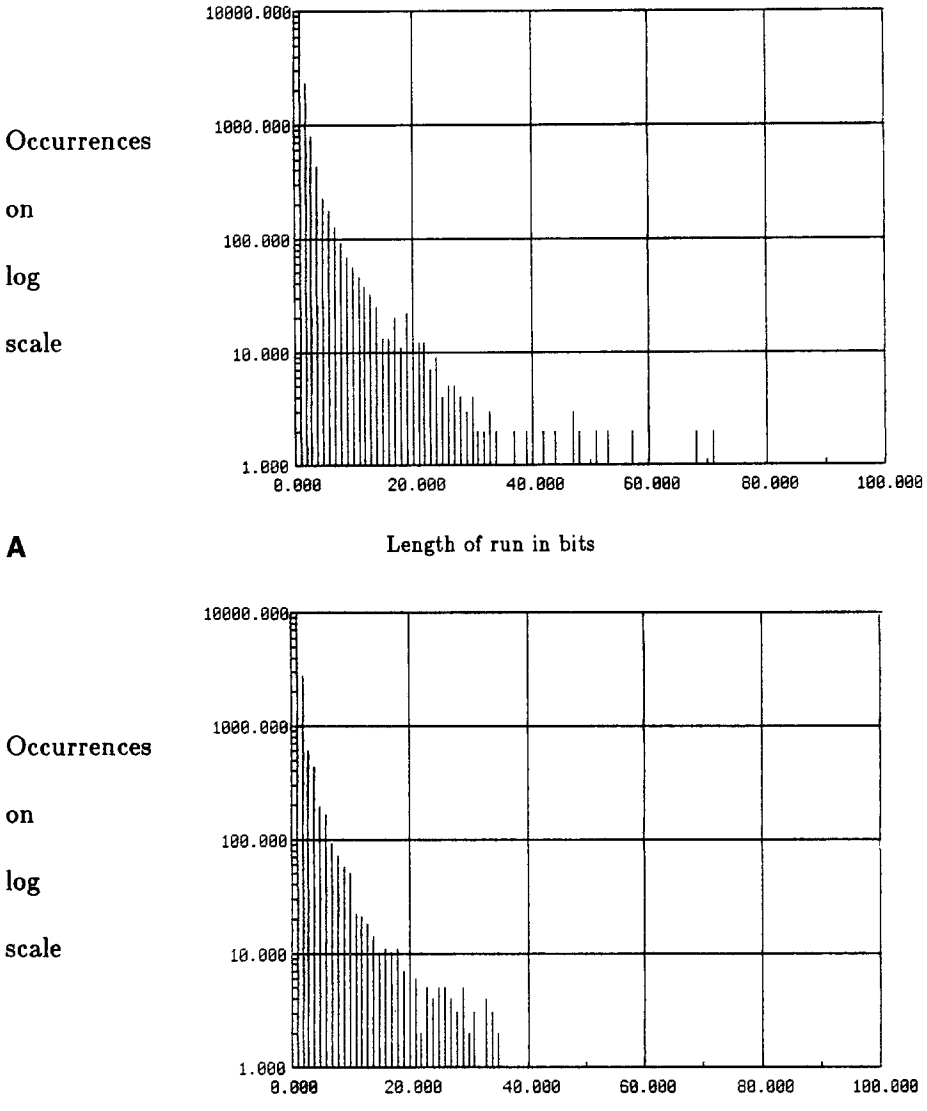


Figure 21. Histograms of runs of GJCs for CCITT image 2.

to realize registers, logic, and control functions, as indicated in Table 4. A small control ROM is alsu used. The total area of the proposed RASBOC chip would be approximately 4.8 mm by 5.3 mm including pads. In order to achieve encoding at video rates, the off-chip data buses were not multiplexed, and thus the chip would be packaged in a 128-pin pin-grid-array.

6 Summary and Conclusion

In this paper we have presented a new architecture for extracting contours of objects in bitonal images. The extraction algorithm being implemented performs two raster-scan passes through the source image before decoding the ordered list of boundary coordinates. The first pass is a statistic-gathering phase wherein data are collected for subsequent memory allocation. In the second pass through the image direction codes are built to represent object contours.

The coding algorithm, which derives from methods of raster-scan component-labeling, is attractive for pipelined hardware implementation due to its regularity in accessing pixels and the use of only local connectivity information. The issue of equivalent pair merging for subcontours was explained and shown to be more straightforward than that for raster-scan component-labeleing.

The compression rates of grid-joint coding the standard CCITT images were measured, and these

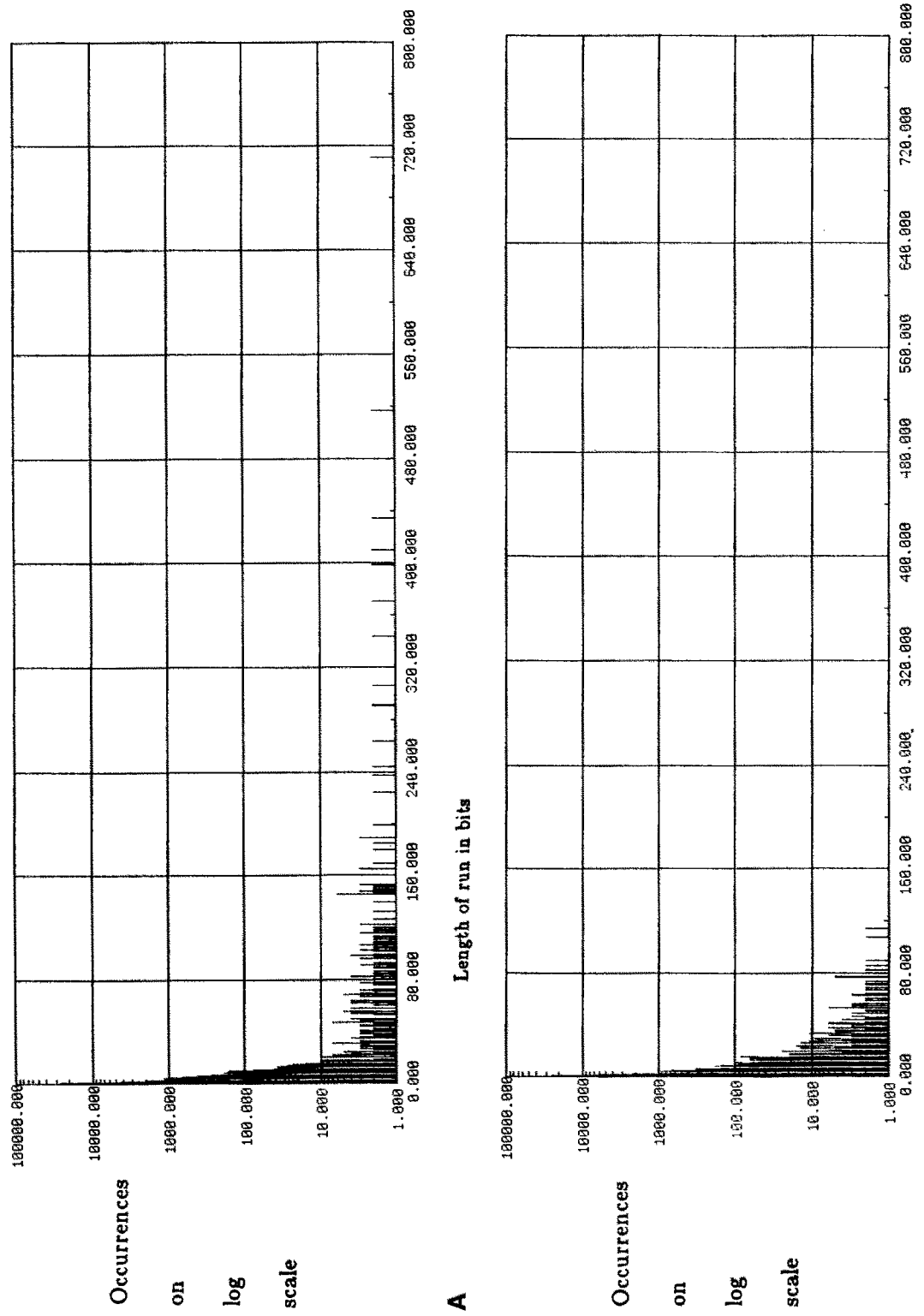


Figure 22. Histograms of runs of GJCs for CCITT image 3.

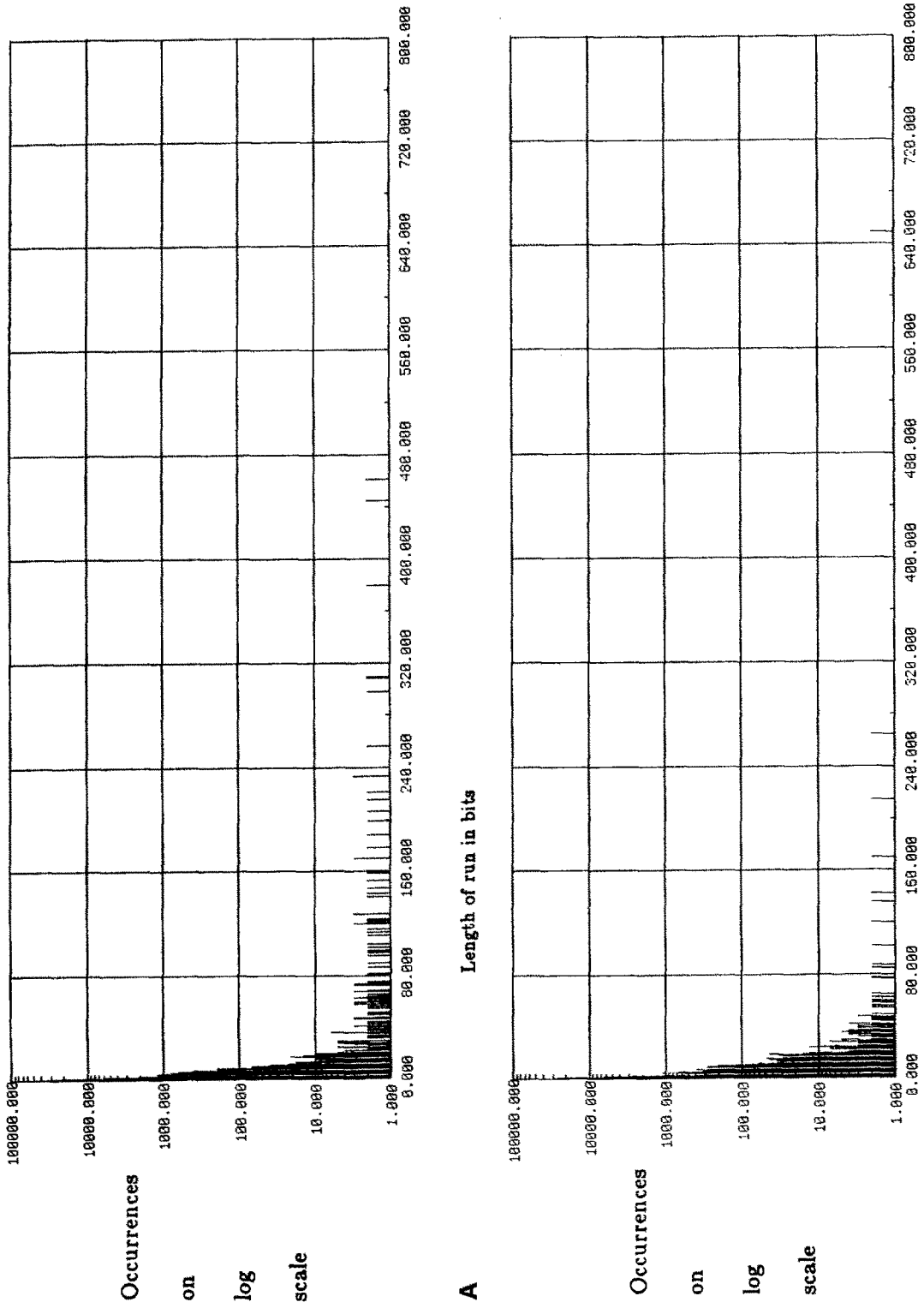


Figure 23. Histograms of runs of GJCs for CCITT image 5.

Table 4. Estimates of transistor logic and memory requirements for a 4K by 4K RASBOC

Component	On-Chip Logic	On-Chip Memory	Off-Chip Memory
Coordinate registers	1,280		
Next queue counter	400		
P buffer	1,536	4 Kbit	
Control	150	0.05 Kbit	
R label buffer			64 Kbit
L label buffer			64 Kbit
Left queue access controller	4,125		1 Mbit
Right queue access controller	4,125		1 Mbit
Equivalent pair unit	2,055		1 Mbit
Right queue memory			64 Kbit
Left queue memory			64 Kbit
X MAXPOINT memory			0.75 Mbit
Y MAXPOINT memory			0.75 Mbit
Totals	13,671 Trans.	4.05 Kbit	4.756 Mbit

were used to illustrate the significance of boundary coding. For the images shown the maximum compression rate of grid-joint coding outperforms that for standard run-length encoding. Examples of running the RASBOC simulator on a diverse set of binary images were also shown.

The detailed architecture of the RASBOC chip set was presented. This architecture contains features that make it attractive for high-speed, video rate boundary encoding of large binary images. It takes advantage of line-delay buffers, pipelined operation, and regular pixel and code access mechanisms to simplify much of the internal processing and external interfacing requirements. A novel implementation of a hardware memory allocation process is used to reduce the external memory storage requirements and to position properly the boundary codes in the code storage unit. The proposed new VLSI RASBOC boundary encoder/decoder chip was also described.

Acknowledgment. This research was supported in part by the U.S. Army Research Office under grant DAAG 29-82-k-0077, Naval Ocean Systems Center under grant N66001-85-D-0203 and also the UC MICRO Program sponsored by Hughes Research Labs, Malibu and Lockheed Aircraft Co., Palo Alto, CA.

References

Apffel JM (1987) An Architecture for Region Boundary Extraction in Raster Scan Images. Master of Science Thesis, March

Apffel JM, Sanz JLC, Jain AK, Current KW (1987) An Architecture for Boundary-Based Segmentation. SPIE Intelligent Robots and Computer Vision 848:586–593

Atkinson HH, Gargantini I, Walsh TRS (1985) Counting Regions, Holes, and Their Nesting Level in Time Proportional to the Border. Computer Vision Graphics and Image Processing 29:196–215

Caponetti L, Cliaradia ML, Distante A, Veneziani M (1984) A Track-following Algorithm for Contour Lines of Digital Binary Maps. In: Leviadi (ed) Digital Image Analysis, Pittman, pp. 149–154

Cederberg R (1979) Chain-Link Coding and Segmentation for Raster Scan Devices. Computer Graphics and Image Processing 10:224–234

Chakravarty I (1981) A Single-Pass Chain Generating Algorithm for Region Boundaries. Computer Graphics and Image Processing 15:182–193

Cohen J (1981) Garbage Collection of Linked Data Structures. Computer Surveys 13(3):341–367

D'Amato D, Pintsov L, Koay H, Stone D, Tan J, Tuttle K, Buck D (1982) High Speed Pattern Recognition System for Alphanumeric Handprinted Characters. In: IEEE Proc. Pattern Recognition and Image Processing, pp. 165–169, June

Danielson PE (1982) Encoding of Binary Images by Raster-Chain-Coding of Cracks. In: IEEE Proc. 6th Int'l Conf. on Pattern Recognition, pp. 335–338

Ellis TJ, Proffitt D, Rutkowski W (1979) Measurement of the Lengths of Digitized Curved Lines. Computer Graphics and Image Processing 10:333–347

Freeman H (1961) On the Encoding of Arbitrary Geometric Configurations. IRE Trans. EC-10:260–268

Freeman H (1974) Computer Processing of Line-Drawing Images. ACM Computing Surveys 6(1):57–97

Haralick RH, Shapiro LG (1985) SURVEY: Image Segmentation Techniques. Computer Vision, Graphics and Image Processing 29:100–132

Jain AK (1988) Fundamentals of Digital Image Processing. Prentice-Hall

Juvin D, Dupeyrat B (1982) ANIMA (ANalysis of IMAges): A Quasi-Real Time System. In: IEEE Proc. Pattern Recognition and Image Processing, pp. 358–361

Kundu MK, Chaudhuri BB, Majumder DD (1985) A Generalised Digital Contour Coding Scheme. Computer Vision Graphics and Image Processing 30:269–278

Landy M, Cohen Y (1985) Vectorgraph Coding: Efficient Coding of Line Drawings. Computer Vision Graphics and Image Processing 30:331–334

Lindgren T (1983) Programs for Handling RC-Coded Images. U.S. Dept. of Commerce: Nat'l Technical Information Service, vol. PB84-158260, December

Luhscher WHHJ, Beddoes MP (1989) Fast Binary-Image Boundary Extraction. Computer Vision Graphics and Image Processing, to appear

Milgram D (1979) Constructing Trees for Region Description. Computer Graphics and Image Processing 11:88–99

Mitiche A, Aggarwal JK (1985) Image Segmentation by Conventional and Information-Integrating Techniques: A Synopsis. Image and Vision Computing 3(2):50–62

Morrin T (1976) Chain-Link Compression of Arbitrary Black-White Images. Computer Graphics and Image Processing 5:172–189

Paglieroni DW, Jain AK (1985) A Control Point Theory for Boundary Representation and Matching. In: Proc. ICASSP, vol. 4, pp. 1851–1854

Paglieroni DW, Jain AK (1986) Contour Control Point Algorithms for Shape Measurement and Inspection. In: SPIE Conf. on Advances in Intelligent Robotic Systems, Cambridge, October

Pavildas T, Cherry LL (1982) Vector and Arc Encoding of Graphics and Text. In: IEEE Proc. 6th Int'l Conf. on Pattern Recognition, pp. 610–613

Persoon E, Fu KS (1977) Shape Discrimination Using Fourier Descriptors. IEEE Trans. Sys., Man., Cybern. 7:170–179

Price KE (1984) Matching Closed Contours. In: IEEE Workshop on Computer Vision: Representation and Control, pp. 130–134, Annapolis, Maryland, April

Rosenfeld A (1970) Connectivity in Digital Pictures. Journal of ACM 17(1):146–160

Ruetz PA (1986) Architectures and Design Techniques for Real-Time Image Processing. Ph.D. dissertation, Electronics Research Laboratory, U.C. Berkeley, vol. UCB/ERL M86/37

Ruetz PA (1986) A Custom Chip Set for Real-Time Image Processing. In: Proc. Int'l Conf. Acoustics, Speech and Signal Processing, pp. 801–804, Tokyo

Ruetz PA, Brodersen RW (1988) An Image-Recognition System Using Algorithmically Dedicated Integrated Circuits. Machine Vision and Applications 1(1):3–22

Samet H, Webber RE (1982) Line Quadtrees: A Hierarchical Data Structure for Encoding Boundaries. In: IEEE Proc. Pattern Recognition and Image Processing, pp. 90–92, June

Sobel I (1978) Neighborhood Coding of Binary Images for Fast Contour Following. Computer Graphics and Image Processing 8:127–135

Suzuki S, Abe K (1985) Topological Structural Analysis of Digitized Binary Images by Border Following. Computer Vision Graphics and Image Processing 30:32–46

Tanaka Y (1985) A VLSI Algorithm for Sorting Variable-Length Character Strings. New Generation Computing 3:307–328