# Lower Bounds on the Hardware Area Required to Process Signals in Real-Time

by

Carla Savage

Center for Communications and Signal Processing
Department of Computer Science
North Carolina State University

December 1985

CCSP-TR-85/21

**Abstract**

In this paper we describe a method of reasoning which can be used to establish lower bounds on the hardware area required to perform certain signal transformations in real-time. We use this method of reasoning to prove that to perform region labeling on a sequence of n by n raster scan binary images in real-time requires hardware of area at least $cn^2$ for some constant c.

## 1. Introduction

In an earlier research report [4] we discussed our motivation for studying special purpose hardware for real-time signal processing. The primary reason for our study is our conviction that, regardless of the current state of research or technology, there will always be current problems for which, because of constraints such as extremely high data rates or extremely small area requirements, special purpose hardware will be the only feasible solution. Processing images transmitted at video rates is in that position in today's technology. For that reason, we chose image processing as the initial focus of our studies on the design of special purpose hardware for real-time signal processing. Our major goal has not been to come up with specific designs (although we are doing work in that area), but to gain insight into the design process at the highest level.

From the work we have done over the past several years, one insight we have gained can be grossly generalized as follows. In designing hardware to perform real-time processing of signals, it seems there is no problem in meeting the real-time rate, even for, say, 512 by 512 images transmitted at 30 frames per second. That is, at least for every image processing problem we considered, we could come up with some architecture, usually pipelined, which when implemented in state of the art VLSI technology could be expected to process at transmission rates. However, it appears that to achieve these processing rates, we must pay a very high price in area.

Operations on images can be classified as point (Figure 1), local (Figure 2), or global (Figure 3) according to whether a pixel value $y_i$ in the output image depends only on the corresponding pixel value $x_i$ in the input image, or only on a neighborhood of values around $x_i$, or on every value in the input image. Examples of point, local, and global operations are, respectively, scaling, convolution with a 3 by 3 kernel, and region labeling. Point operations can be performed in real-time by hardware of area independent of the size of the image. It has been shown that convolution of an n by n image with a 3 by 3 kernel can be performed in real-time in area on the order of n [2,3]. We have shown that region labeling on an n by n image can be performed in real-time in area on the order of $n^2$ [1,5].

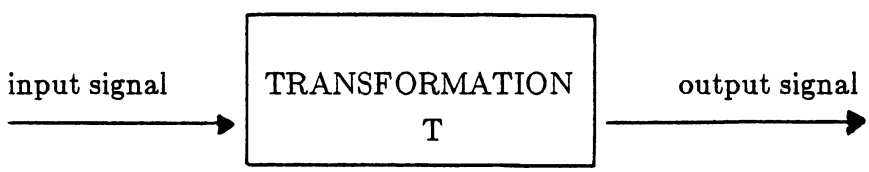
For 512 by 512 images, n=512 and although special purpose hardware of area $0(n)$ may soon be feasible, area $0(n^2)$ may remain unreasonable for special purpose devices for some time to come. We were curious whether a more clever hardware design would allow us to do real-time region labeling in area smaller than $n^2$. In this paper, we prove that $cn^2$, for some constant c, is a lower bound on the area of any finite state piece of hardware which does real-time region labeling. Since the region labeling problem, as we have

described it, is a relatively simple problem, we expect this lower bound to hold for many other problems involving global operations on images.

In Section 2, we describe the argument we use to establish lower bounds on the area required to perform signal transformations in real-time. In Section 3, we use the arguments to establish some lower bounds. Of particular interest, we show that to perform region labeling on an n by n binary image in hardware requires area at least $cn^2$ for some constant c. In Section 4, we suggest an investigation of ways to relax problem constraints or take advantage of special properties of the input signal data which would allow real-time signal processing to proceed in area less than that predicted by general lower bounds such as $n^2$.

## 2. How to Prove Lower Bounds

First we will try to formalize the notion of a signal transformation and what we mean by a hardware device which performs the transformation.



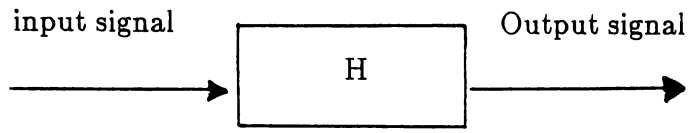

An *alphabet* is defined to be a finite set of symbols. For example, for any fixed k, the set of k bit integers is an alphabet of size $2^k$. An *input signal* is an infinite sequence of discrete values from some alphabet I, called the *input alphabet.* For example, an input signal could be an infinite sequence of bits or an infinite sequence of 8-bit numbers. Similarly, an *output signal* is an infinite sequence of discrete values from some alphabet O, called the output alphabet. If A is an alphabet, let $\omega(A)$ denote the set of all infinite strings consisting of elements from A. Then an input signal is an element of $\omega(I)$ and an output signal is an element of $\omega(O)$.

Define a signal transformation T to be a function (or possibly only a partial function) of the form

$$T : \omega(I) \rightarrow \omega(O)$$

for some alphabets I and O. Let x be an element of $\omega(I)$ and let $y=y_1\, y_2\, y_3 \cdots$ be an

element of $\omega(O)$, where each $y_i$ is in O. Then if $T(x)=y$, let $T_i(x)$ denote the element $y_i$, the ith symbol in the string y.




Finite state piece of hardware

Let H be a piece of hardware which operates, according to some clock, at discrete time intervals. Assume that during a time interval, H reads an input symbol, possibly performs some operations and modifies the contents of its storage elements, and then may or may not produce an output value. In case H does not produce an output during a particular time interval we assume that H in fact produces some special symbol "b" and that b is an element of the output alphabet, O, reserved for this purpose. We will call H a *finite state device* if

(i) H has a finite number of storage elements (for example, registers) $r_1,r_2,...,r_k$, each capable of storing some finite set of values.

(ii) The values stored in elements $r_1,r_2,...,r_k$ at time t are uniquely determined by the values stored in those elements at time t-1 and the input value to H at time t. (Note that this implies nothing about the architecture of H. It can be sequential, parallel, multiprocessing, or pipelined, and no restrictions are put on processor and/or memory organization.)

(iii) The output of H at time t is uniquely determined by the values stored in elements $r_1,r_2,...,r_k$ at time t-1 and the input value to H at time t.

Let $v_i(t)$ denote the value stored in $r_i$ at time t. The state of H at time t is the ordered k-tuple $<v_1(t),v_2(t),...,v_k(t)>$. Then we have the following result.

**Theorem 1.** If H is a finite state device with storage elements $r_1,r_2,...,r_k$ and if $n_i$ is the maximum number of elements which can be stored in $r_i$ then the maximum number of states which can be represented by H is

$$\prod_{i=1}^{k} n_i \qquad \square$$

We will make the following assumption: A storage device which is capable of storing

n distinct values must occupy area at least proportional to $\log_2 n$. (Note that for any j, $\log_j n$ is proportional to $\log_2 n$. ) Based on this assumption, we can draw conclusions about the area of H.

**Theorem 2.** If H is a finite state device with storage elements $r_1, r_2, \ldots, r_k$ and if $n_i$ is the maximum number of elements which can be stored in $r_i$ then the area of H is at least, for some constant c,

$$c \sum_{i=1}^{k} \log_2(n_i) \qquad \square$$

Let H be a finite state device with input symbols from a set I and output symbols from a set O. Let T be a signal transformation

$$T: \ \omega(I) \rightarrow \omega(0)$$

Say that *H computes T* if there exists a constant D (delay) such that for any $x=x_1x_2x_3\ldots$ in $\omega(I)$, if $x_i$ is the symbol read by H at time i, then $T_i(x)$ is the symbol produced by H at time i+D. That is, after an initial delay of D, H produces as output the string T(x), one symbol per time unit.

**Theorem 3.** Assume $T: \omega(I) \rightarrow \omega(0)$ is a signal transformation such that any finite state device which computes T requires at least q states. Let H be a finite state device which computes T. Then the area of H is at least $c\log_2 q$ for some constant c.

**Proof.** Let $r_1, r_2, \ldots, r_k$ be the storage elements of H and let $n_i$ be the maximum number of distinct values which can be stored by $r_i$. By Theorem 1,

$$\prod_{i=1}^{k} n_1 \geq q.$$

Take logarithms to the base 2 of both sides to get

$$\log_2 \prod_{i=1}^{k} n_1 \geq \log_2 q$$

Thus,

$$\sum_{i=1}^{k} \log_2 (n_i) \geq \log_2 q$$

By Theorem 2, then, the area of H is at least $c \log_2 q$ for some constant c. $\square$

Our goal is to show that for certain signal processing transformations, there are

nontrivial lower bounds on the area required by any hardware device which performs these transformations, assuming that the hardware device behaves as a finite state device. We will prove lower bounds on area by establishing lower bounds on the number of states and then making use of Theorem 3. In order to prove lower bounds on the number of states, we will use the line of reasoning described in Theorem 4 below. Although this line of reasoning seems intuitively clear, we have not given a proof that it is valid. In our continuing research, we plan to make more precise our model of a finite state signal processing device so that we can give a formal proof (and probably a generalization) of Theorem 4.

**Theorem 4.** Let H be a finite state device which computes transformation T: $\omega(I) \rightarrow \omega(0)$. Let S be a set of input sequences (that is, S is a subset of $\omega(I)$) satisfying the following: there exist integers i and N with $1 \leq i \leq N$ such that for each x in S

(i) $T_i(x)$ cannot be output before time N (that is, before reading the Nth symbol of x.)

(ii) At time N, $T_i(x)$, $T_{i+1(x)}$, ..., $T_N(x)$ are uniquely determined (that is, uniquely determined by the first N symbols of x.)
and

(iii) for any pair of distinct strings x and x′ from S,

$$T_i(x)\ T_{i+1}(x)...T_N(x) \neq T_i(x')T_{i+1}(x')...T_N(x').$$

Then we can conclude that H must have at least $|S|$ states. □

The intuitive reasoning here is that if $T_i(x)$ through $T_N(x)$ cannot be output before time N, but they are uniquely determined at time N then at time N, H must be storing enough information to distinguish between all sequences $T_i(x)T_{i+1}(x)...T_N(x)$ which are possible outputs. Since every element of S gives rise to a different sequence $T_i(x)\ T_{i+1}(x)...T_N(x)$, H must be able to distinguish between $|S|$ different sequences at time N and therefore must have at least $|S|$ states.

## 3. Lower Bounds

We first prove a lower bound on the area required to compute a rather artificial transformation as an example of how to use Theorem 4. Next we will consider the problem of region labeling.

Let I = {0,1} and let $x=x_1x_2x_3\ldots$ be an element of $\omega(I)$. Let $n>1$ be an integer and let T be the transformation $T: \omega(I) \rightarrow \omega(I)$ defined by

$$T_i(x) = \begin{cases} 0, & \text{if the sequence } x_i x_{i+1} \cdots x_{ceil(i/n)*n} \\ & \text{contains an even number of ones} \\ 1, & \text{otherwise} \end{cases}$$

Let S be the subset of $\omega(I)$ consisting of all elements $x=x_1x_2x_3\cdots$ such that $x_i=0$ for all $i>n$. Note that $|S|=2^n$ since there is a one-to-one correspondence between the elements of S and the set of n-bit binary numbers. Let H be a finite state device which computes T. Then, we establish the conditions of Theorem 4. It is clear that

(i) $T_1(x)$ cannot be output before time n and

(ii) at time n, $T_1(x)T_2(x)\cdots T_n(x)$ is completely determined.

To show (iii), let x and x′ be elements of S and suppose $T_1(x)\ T_2(x)\ldots T_n(x)=T_1(x')T_2(x')\ldots T_n(x')$. Then by definition of T,

$$x_n = x_n' = \begin{cases} 0 & \text{if } T_n(x) = T_n(x') = 0 \\ 1 & \text{if } T_n(x) = T_n(x') = 1 \end{cases}$$

and for $1 \leq i < n$,

$$x_i = x_i' = \begin{cases} 0 & \text{if } T_i(x)=T_i(x')=T_{i+1}(x') = T_{i+1}(x) \\ 1 & \text{otherwise} \end{cases}$$

Further, by definition of S, $x_i=x_i'$ for $i > n$. Thus it must be that $x = x'$. By Theorem 4, since conditions (i), (ii), and (iii) hold, H must have at least $2^n$ states. According to Theorem 3, then, any finite state device which computes T must have area at least cn for some constant c.

We now consider a simplified version of the region labeling problem. An n by n binary image can be regarded as an n by n array, A, in which each entry represents a pixel and is either 0 ("black") or 1 ("white"). We call two pixels (i,j) and (i′j′) *adjacent* if and only if either

$$i = i' \quad \text{and} \quad j = j' \pm 1$$

or

$$j = j' \quad \text{and} \quad i = i' \pm 1$$

Two pixels e and e′ are in the same region if either e=e′ or there exists a sequence of pixels

$$e = e_1, e_2, e_3, \ldots, e_k = e'$$

such that $e_i$ and $e_{i+1}$are adjacent for i=1, 2,...,k-1 and all of $e_1,e_2,...,e_k$ have the value 0 or all have the value 1. Being "in the same region" is an equivalence relation on the set of pixels and therefore partitions the set of pixels into equivalence classes which are the *regions* of the image. A *region labeling* is an assignment of labels from some set L to the pixels of an image in such a way that two pixels are assigned the same label if and only if they are in the same region. Figure 5 shows one possible region labeling for the binary image in Figure 4.

A region labeling transformation T can be defined as

$$T: \omega(\{0,1\}) \rightarrow \omega(L)$$

where L is some finite set of labels. An input sequence is an infinite sequence of n by n binary images, transmitted in raster scan order. Let $N = n^2$. If $x=x_1x_2x_3...$ is an input sequence, then $x_1x_2 \cdots x_N$ is the first image, $x_{N+1}x_{N+2}...x_{2N}$ is the second image, and so on. For i = 1,2,3,... and for j = 1,2,...,N, the value $T_{(i-1)N+j}(x)$ is the label of the $j^{th}$ pixel (in raster scan order) of the $i^{th}$ image. We make one further assumption, that the region labeling of the $i^{th}$ image $x_{(i-1)N+1}...x_{iN}$ is independent of the values $x_j$ for j<1)N+1.....ndj>iN.

**Theorem 5.** Any finite state device which performs a region labeling transformation on sequences of n by n binary images must have area at least $cn^2$ for some constant c.

**Proof:** Assume that H is such a finite state device. Let $N=n^2$ and let S be the subset of $\omega(\{0,1\})$ defined by

$$x_i = \begin{cases} 0 \text{ for } i > N \\ \qquad \text{(all but first image)} \\ 1 \text{ for } i = 1 \\ 0 \text{ for } i = 2 \\ 1 \text{ for } i = 3,4,\ldots,n \\ \qquad \text{(first row)} \\ 1 \text{ for } i = n+1,2n+1,\ldots,(n-1)n+1 \\ \qquad \text{(first column)} \\ 1 \text{ for } i = 2n,3n,4n,\ldots,(n-1)n \\ \qquad \text{(last row exluding last pixel)} \\ 0 \text{ for } i = n+2,n+3,\ldots,n+(n-1) \\ 0 \text{ for } i = n+2,2n+2,\ldots(n-2)n+2 \\ 0 \text{ for } i = 2n-1,3n-1,\ldots(n-1)n-1 \\ 0 \text{ for } i = (n-2)n+2,\ldots(n-2)n+(n-1) \\ \qquad \text{(inner border)} \end{cases}$$

That is, S consists of all strings in which all but the first image is zero, and the first image has a border one pixel wide of pixels of value 1 (except for pixel 2 which must be zero and pixel N, whose value is not constrained) and just inside this border, a border one pixel wide of pixels of value zero. (See Figure 6). The pixel values which are not fixed are pixel N and all the pixels in the (n-4) by (n-4) center of the image and these pixels can take on 0 or 1 values in any combination. Therefore,

$$|S| = 2^{(n-4)^2+1}$$

We now show that the conditions of Theorem 4 are satisfied with i=3 and N=N. Let x be in S. First, the label of pixel 3, $T_3(x)$, cannot be output by H until pixel N is input, because until the value of pixel N is read, it is unknown whether or not pixels 1 and 3 are in the same region. Second, by the time pixel N is input, all pixel values in the image are known, and so the regions and their labels are completely determined. Thus, $T_3(x)$ $T_4(x)...T_N(x)$ is uniquely determined by time N. Finally, we must show that for distinct elements x and x′ of S that

$$T_3(x)T_4(x)...T_N(x) \neq T_3(x')T_4(x')...T_N(x').$$

However, note that given the sequence $T_3(x)T_4(x)...T_N(x)$ and the fact that

$x_1 = 1$ and $x_2=0$, we can uniquely reconstruct the image x as follows:

for $1 \leq i < n$ and $2 \leq j \leq n$,

$$x_{in+j} = \begin{cases} x_{in+j-1} & \text{if } T_{in+j}(x) = T_{in+j-1}(x) \\ \text{otherwise,} & 0 \text{ if } T_{in+j-1}(x) = 1, \\ & 1 \text{ if } T_{in+j-1}(x) = 0 \end{cases}$$

(Recall that the first column of image x consists of all ones.) Thus, all conditions of Theorem 4 hold and we can conclude that H must have at least $|S|$ states. By Theorem 3, then, the area of H must be at least, for some constant c',

$$c' \log |S| = c' [(n-4)^2 + 1]$$

$$= c'(n^2-8n+17).$$

But we can show for some constant c,

$$c'(n^2 - 8n+17) > cn^2$$

for all but finitely many values of n. □

## 4. Further Research Directions

We would like to investigate whether there is any way to get around the lower bound of $cn^2$ on the area required to compute certain global functions on images in real-time.

One possibility is to settle for a solution which is, in some sense, approximate rather than exact. Such an approach allowed us to reduce area, but only by a constant factor, in the paper [2]. In the study of algorithms, there are many problems which seem to require exponential time to obtain an exact solution, but can be solved very quickly if only an approximate solution is required.

Another possibility is to somehow encode the image in a more compact form, for example, reducing $n^2$ data items to m data items. If this encoding could be done in real-time by hardware of area 0(m) and if the encoded image could then be processed in real-time by hardware of area 0(m), the original image will have been processed in time 0(m). In the worst case m will be $n^2$, but one may know ahead of time that the characteristics of the data of interest will guarantee that m is much less than $n^2$. In the study of algorithms, again, it can be shown that if an n-node graph is represented by its nxn adjacency matrix, solving any problem on the graph will require time at least $cn^2$. However, if only m entries of this matrix are nonzero, the graph can be represented more compactly by its adjacency lists and from this representation, many problems can be solved in time only 0(m) or 0(mlogn). Here, m may be as large as $n^2$, in which case we gain nothing.

However, frequently m is much smaller than $n^2$, for example on the order of n or nlogn.

One further possibility is to take advantage of some special information which may be available about the data. For example, it can be shown that cnlogn is a lower bound on the time required to sort n numbers. However, if it is known that the n numbers to be sorted are in the range 0,...,k-1, then the numbers can be sorted in time 0(n+k), essentially linear in n. (If k is very large relative to n, of course, this would not be a very good way to sort.)

**References**

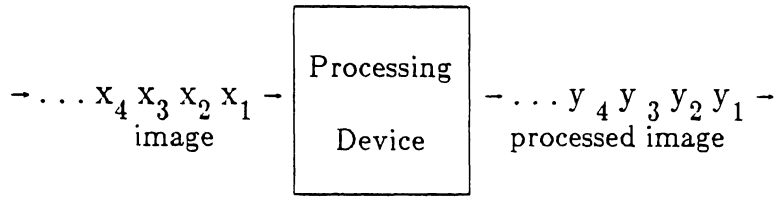
1. S. Ashtaputre and C. Savage, "Systolic Arrays with Embedded Tree Structures for Connectivity Problems," *IEEE Transactions on Computers*, Vol. C-34, No. 5, May 1985.

2. S. Ashtaputre, C. Savage, and W. E. Snyder, "Using an Approximate Multiplier in a One-dimensional Array Architecture for Real-Time Convolution," CCSP Technical Report TR 85/20.

3. H. T. Kung, L. M. Ruane, and D. W. L. Yen, "A Two-Level Pipelined Systolic Array for Convolutions," *VLSI Systems and Computations,* H. T. Kung, B. Sproull, and G. Steele, eds., Computer Science Press, 1981.

4. C. Savage, "Special Purpose Hardware for Real-Time Signal Processing: Motivation," CCSP Technical Report, WP-85/13.

5. W. E. Snyder and C. Savage, "Content-Addressable Read/Write Memories for Image Analysis," *IEEE Transactions on Computers*, Vol. C-31, No. 10, October 1982.



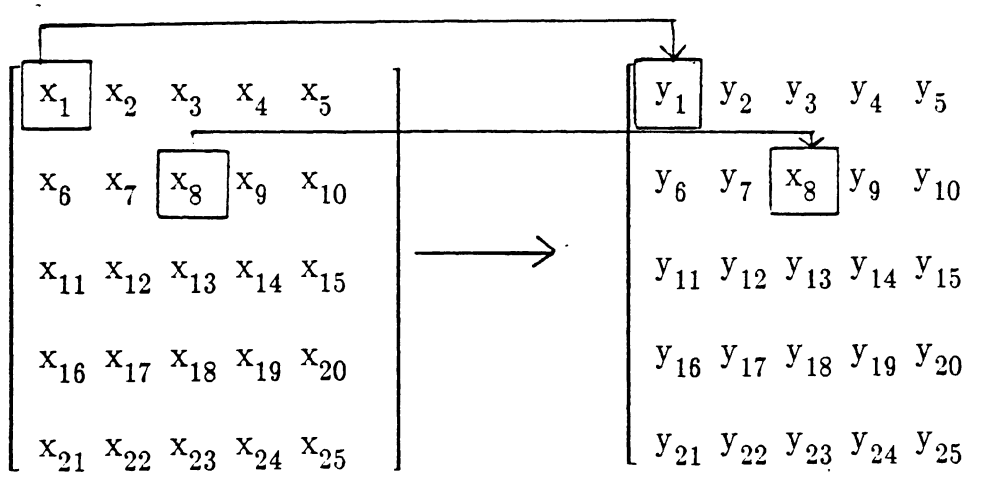
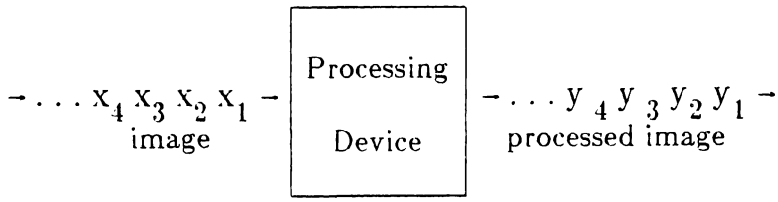




Figure 1. Point Operations



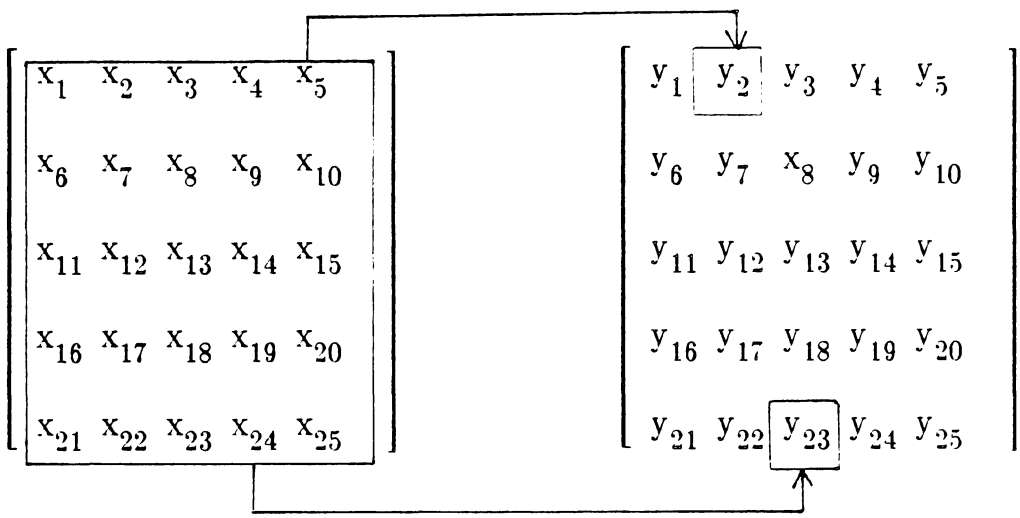
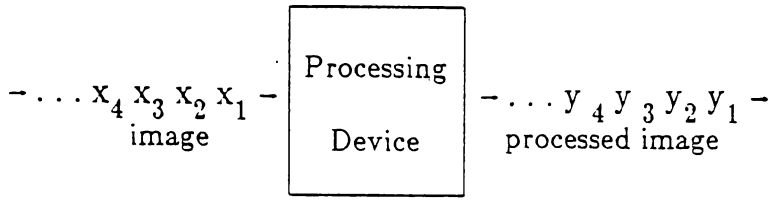




Figure 2. Local Operations



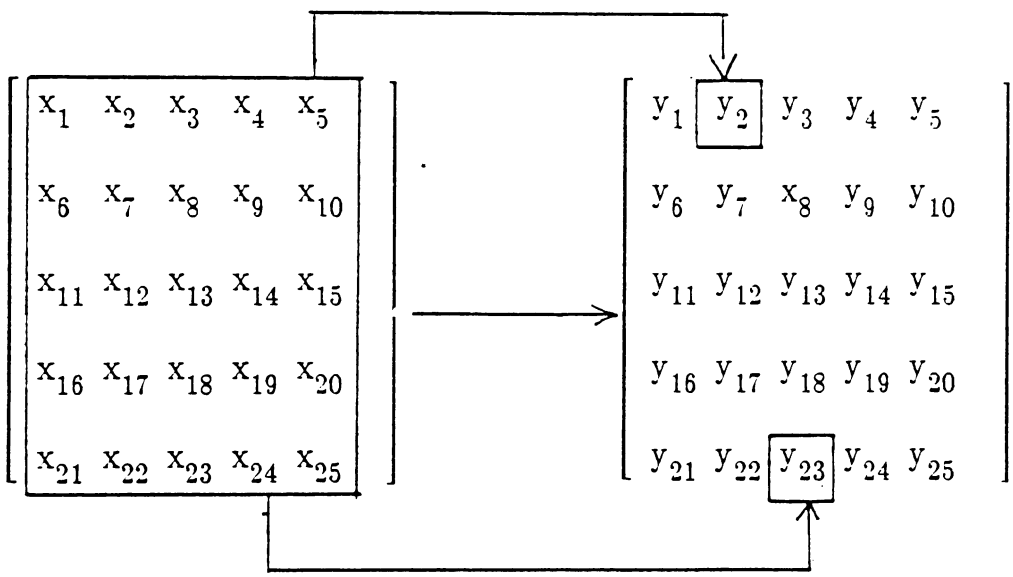




Figure 3. Global Operations

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Figure 4. A Binary Image

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| A | B | A | A | A | A | A | A |
| A | B | B | A | C | C | C | A |
| A | B | D | E | F | F | C | A |
| A | A | E | E | E | F | C | A |
| A | E | E | E | E | E | A | A |
| A | A | E | E | E | G | H | A |
| A | I | A | E | A | J | A | A |
| A | A | A | A | A | A | A | A |

Figure 5. A Region Labeling of the Image in Fig. 4

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | ? | ? | ? | ? | 0 | 1 |
| 1 | 0 | ? | ? | ? | ? | 0 | 1 |
| 1 | 0 | ? | ? | ? | ? | 0 | 1 |
| 1 | 0 | ? | ? | ? | ? | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | ? |

Figure 6. The first image in all sequences from S must have this form. The symbol "?" denotes entries which may be either 0 or 1. All other entries are fixed.