To the Graduate Council:

I am submitting herewith a thesis written by Alan G. Metcalfe entitled "Execution of a Parallel Counting Algorithm Using Array Logic (tse) Devices." I recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Robert E. Bodenheimer
Robert E. Bodenheimer
Major Professor

We have read this thesis and recommend its acceptance:

RC Gonzalez

J. W. Waller

Accepted for the Council:

Hilton A. Smith
Vice Chancellor
Graduate Studies and Research

Thesis
76
.M483
cop. 2

EXECUTION OF A PARALLEL COUNTING ALGORITHM USING ARRAY LOGIC (TSE) DEVICES

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Alan G. Metcalfe

December 1976

1305767

To Elizabeth

ACKNOWLEDGMENTS

The author would like to thank the National Aeronautics and Space Administration, Goddard Space Flight Center, for this research. Gratitude is extended to Dr. Robert E. Bodenheimer for his work as advisor during the research and preparation of this thesis. Appreciation is also extended to Dr. R. C. Gonzalez and Dr. J. W. Waller for serving on the author's committee.

ABSTRACT

A new family of digital logic elements, known as tse logic devices, have been proposed by D. H. Schaefer and J. P. Strong at the Goddard Space Flight Center, Greenbelt, Maryland. Tse logic elements are parallel computing elements which implement primitive logical functions concurrently at each position of a two-dimensional binary array. The purpose of this research was to examine different tse hardware structures for performing a certain task.

A parallel algorithm for counting the number of logic-1 elements in a binary array or image was developed at GSFC during preliminary investigation of the tse concept. After summarizing the research at GSFC, the counting algorithm is implemented using a basic combinational structure. Modifications which improve the efficiency of the basic structure are also presented. A programmable tse computer structure is then proposed, along with a hardware control unit, tse instruction set, and software program for execution of the counting algorithm. Finally, a comparison is made between the different structures in terms of their more important characteristics. To more clearly illustrate the projected advantages of tse logic, a program to perform the same task was written for a conventional binary processor and included in the comparison.

TABLE OF CONTENTS

CHAPTER		PAGE
1.	INTRODUCTION	1
2.	TSE LOGIC DEVICES	5
3.	A PARALLEL COUNTING ALGORITHM	14
	Modified forms of the counting algorithm	25
	Generating sums in sectors of an image	25
	Simplification of the algorithm for clustered elements	30
4.	TSE HARDWARE IMPLEMENTATION OF THE COUNTING ALGORITHM	34
	Combinational circuit approach	34
	Pipeline network implementation	38
	Implementation using a programmable tse processor	41
	Hardware considerations	41
	Software considerations	54
	Modified forms of the counting algorithm	64
5.	COMPARISONS AND CONCLUSION	66
LIST OF REFERENCES		74
VITA		76

LIST OF TABLES

TABLE		PAGE
1.	Summary of Modifications per Iteration	26
2.	Tse Computer Instruction Set	55
3.	Tse Processor Instructions	58
4.	Tse Program Control Instructions	59
5.	Tse Computer Program for Execution of the Counting Algorithm	60
6.	Program for Execution of the Counting Algorithm on the IBM 360	68
7.	Summary of Image Processing Times for Tse and Conventional Implementations	70
8.	Physical Characteristics of Tse Implementations of the Counting Algorithm	72

LIST OF FIGURES

FIGURE		PAGE
1.	A two tse Input, Digital AND Gate	6
2.	Use of DUPLICATORS to Increase Effective Fan-Out of a tse Device to Four	8
3.	Method of Implementing SLIDE Operation	9
4.	An Example of Elementary tse Operations on Typical Images	10
5.	Control Method Used for Switching of Image Paths	12
6.	A tse Plane in Which Each 1-Element Represents a Classified Region Whose Area is Desired	15
7.	The Result of Applying the Counting Algorithm to the Binary Image of Figure 6	17
8.	Result After Each Iteration of the Counting Algorithm, Indicating Partial Summing Method	18
9.	The Results of Applying the Steps in the Algorithm to Image A for One Iteration	20
10.	The Results and Significance of Applying the Algorithm to Image A After Each Iteration	24
11.	Example of Partial Summing Where an Error is Generated	28
12.	Example of Modified Counting Algorithm	31
13.	Example of Simplification of the Algorithm	33

FIGURE PAGE

14. Combinational Network Implementation 35

15. Contents of a Typical Box for the Combinational Circuit Implementations 36

16. Pipeline Network Implementation to Increase Image Processing Rate 39

17. Latch Implementation for the tse Register 40

18. A Microprogram, Microprocessor Control Unit Concept for the tse Processor 44

19. Block Diagram for the tse Processor 46

20. Organization of the tse Logical Operations Unit 47

21. Image Buss with Conditional and Monitor Devices 49

22. Organization of the tse Image Registers 50

23. Organization and Content of tse ROM Registers 51

24. Implementation of a Horizontal Sweep Device [5] 52

CHAPTER 1

INTRODUCTION

Since the advent of the first computers, a great deal of effort has been devoted to the task of developing processors with improved data processing rates. Technological innovations have been the primary contribution to the improvements which have been realized throughout the history of the computer. However, there is evidence that the speed at which present-day components can operate is fast approaching the limit at which electronic signals can propagate [1]. Thus, refinements in areas other than the speed of semiconductor devices will provide the probable source for significant increases in data processing rates in the future.

One such area which has gained interest in recent computer developments is the concept of parallel processing. The term "parallel processor" is used to describe a computer whose Arithmetic Logic Unit is structured to operate on each bit of an n- bit operand concurrently. This term is sometimes also used to describe computer architectures in which a number of different instructions may be executing at any one time. However, a description of such processors is beyond the scope of this investigation. Of particular interest in this research is the array processor, a special type of parallel-data processor. The array processor, in general, addresses and operates upon large blocks of bits, usually multidimensional in their arrangement.

The concept of the parallel processor is actually not new. Virtually all computers in use today exhibit some degree of parallelism in that they are word-oriented machines. This is justified by the fact that some of the most common forms of data handled by these machines (for example, ASCII or BCD information) occur most frequently as a group of bits. The processing of such data would become unnecessarily cumbersome if handled bit-by-bit. Therefore, the word-oriented processor is favored over a completely serial one. In fact, the word-oriented computer has become the most highly developed and widely used form of information processing machine in general use at this time.

Only recently have more highly parallel processors been given serious consideration as practical tools for information processing. Although the advantages of such processors are many, their development has been limited by such factors as cost, size, and maintenance considerations, which are due to the increased component count and number of connections. Although some array processors such as SPAC [2], SOLOMON [3], and ILLIAC IV [4] have been proposed, few have actually reached an operational status (the ILLIAC IV has been partially completed), and virtually none have found widespread general use. One factor which is expected to contribute to the development of parallel processors is the current state of integrated circuit technology. This technology allows the fabrication of large scale cellular component structures at a reasonable cost and small size.

Insofar as present-day computers are concerned, they are well organized to handle much of the data which they encounter, since these data occur mainly in a word-oriented fashion. However, one form of

data which becomes cumbersome to process is the digitized image. Even a low resolution, simple binary image would be digitized to a minimum of about 10^4 bits, and, at present, almost all images are processed in a highly serial manner using conventional processors. In order to optimize speed and efficiency, a processor capable of handling at least as many bits as there are image elements would be required. Hence, image processing is a very likely field for the expansion of parallel array processors. In particular, the organization of an image processor would necessarily be two-dimensional, implying communication between horizontal and vertical neighboring elements, instead of simply employing a large number of bits which are spatially unrelated.

The success of the Earth Resources Technology Program (ERTS-1, now known as LANDSAT-1) has led to the consideration of parallel processing, for the development of practical and efficient methods for identification and classification of earth resources. The fact that the LANDSAT-1 satellite images cover approximately six million square kilometers per day has provided the main challenge to the NASA Data Processing Facility for the retrieval and processing of these data. Among the most promising programs which have resulted from this challenge is one at the NASA Goddard Space Flight Center which has projected the development of a family of two-dimensional parallel logic devices. In essence, each of the logic devices represents a computing element whose array size is the same as the number of picture elements (or pixels) in the image and can perform a primitive logical operation concurrently at each image position. Currently, the utilization of fiber optics is being considered for the fabrication of these devices. Projected refinements in fiber

optics technology indicate that parallel computing elements could be constructed which are faster, less power-consuming, and possibly even smaller than conventional electronic components [5].

In the second chapter of this thesis, the work of Shaeffer and Strong related to two-dimensional logic devices is discussed. The third chapter presents Strong's counting algorithm, and the fourth chapter is devoted to the hardware implementation of the algorithm. In the fifth chapter, the merits of the different hardware implementations are compared, and conclusions are presented based on these comparisons.

CHAPTER 2

TSE LOGIC DEVICES

Consider an image composed of a 512 × 512 rectangular array of picture elements in which the gray level of each element is quantized to six bits. There are over 1.5×10^6 bits of information in this image. Another way to visualize a digitized image is as six binary image planes, each plane containing 512 × 512 bits. The binary image plane or bit plane is a two-dimensional binary data array called a "tse." The origin of the term "tse" is the result of an analogy drawn between binary bits and words of the English language. Just as the Chinese language makes use of single symbols which represent many English words, the binary array represents many binary bits. The term "tse" is the transliteration of the Chinese word for the pictograph character, and thus has been adopted as the word for the binary data array [5].

A family of tse logic devices which utilize electro-optical technology and which are capable of performing simple, parallel logical operations simultaneously on one or two tses has been proposed by Shaeffer and Strong [5, 6]. Figure 1 illustrates a tse gate capable of ANDing two binary image planes. A tse gate consists of two parts, an interleaver and an electro-optical threshold device. The interleaver is a passive device which consists simply of two bundles of n^2 optical fibers, where n × n is the size of the bit plane for which the gate is designed. These bundles are merged or interleaved such that corresponding positional elements in the Image A and Image B inputs are combined to the

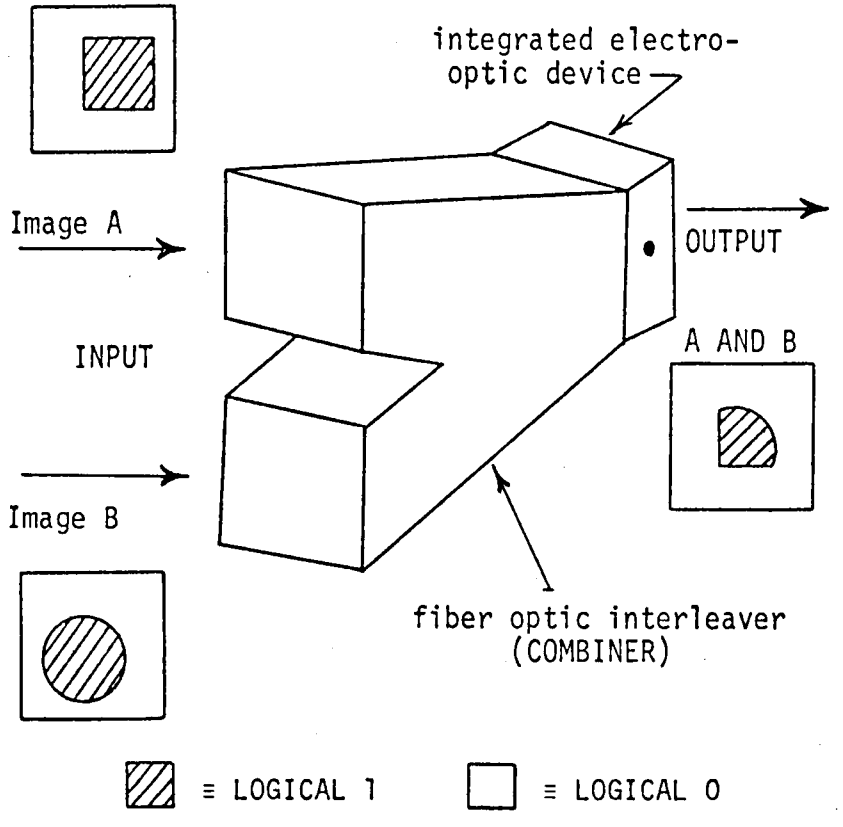


Figure 1. A two tse input, digital AND gate.

same elemental position at the interface of the electro-optical device. When used in this manner, the interleaver is referred to as a combiner. The electro-optical device is an active integrated circuit which converts the optical inputs to electrical signals which are logically ANDed in a conventional manner. Electrical signals at the output are converted to an optical output by an electro-luminescence process.

Since only one fiber bundle can be connected directly to the output, the fan-out of a tse logic gate is one. In order to increase the effective fan-out, one or more interleavers can be used, in a reverse manner, at the output of a tse gate. An interleaver is referred to as a duplicator when used in this manner. Since each output element of a duplicator is one-half the intensity of the input, the original light intensity must be restored before the outputs can be used. Therefore, each output from the duplicator must interface to a reformator, which is an active tse buffer device used to restore the proper optical signal levels. Figure 2 demonstrates how the effective fan-out from the AND gate can be increased to four.

In addition to the AND operation, other primitive operations can be implemented. The OR, EXCLUSIVE-OR, NEGATE and SLIDE operations are implemented in a similar manner as the AND gate, except that the single-operand devices need not include the combiner at the input. The SLIDE operation is an image translation in the UP, DOWN, RIGHT or LEFT direction. Conceptually, this operation is generated by interfacing two fiber bundles with a physical offset, as illustrated in Figure 3. The results of performing these primitive operations on typical images are depicted in Figure 4.

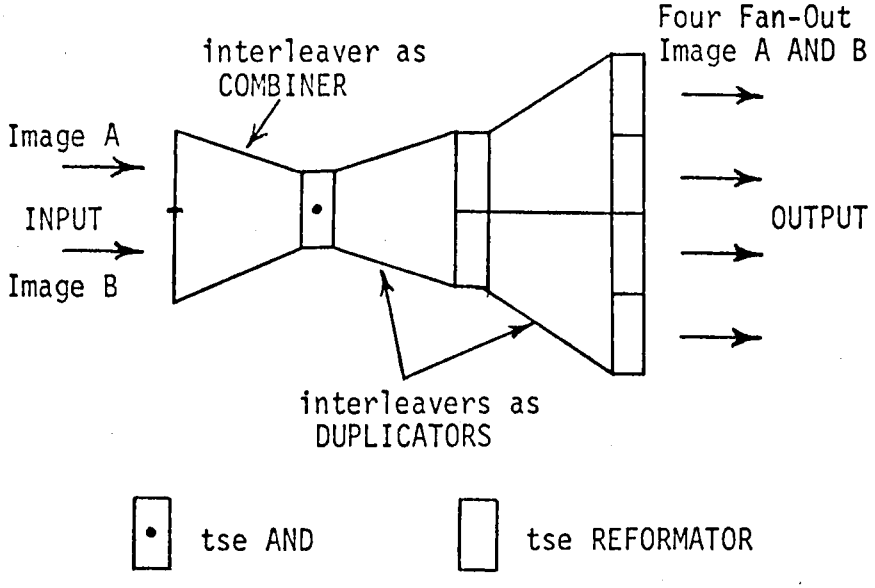


Figure 2. Use of DUPLICATORS to increase effective fan-out of a tse device to four.

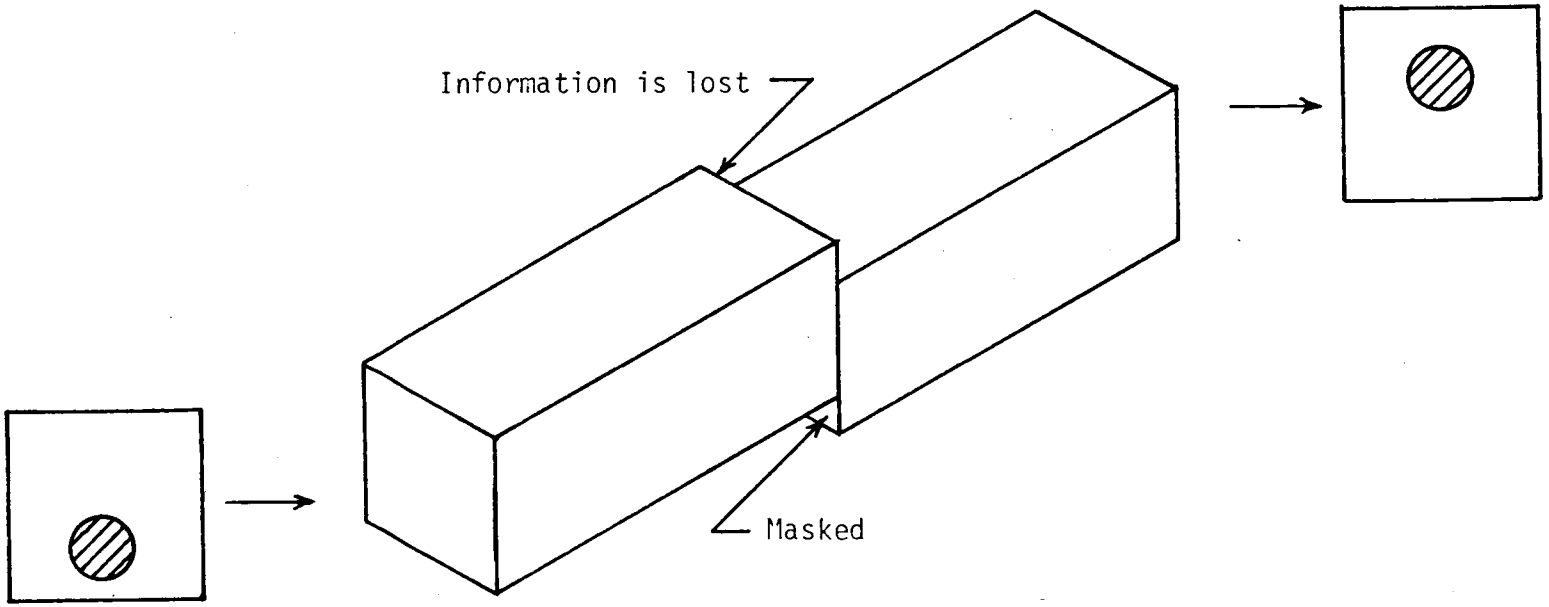


Figure 3. Method of implementing SLIDE operation.

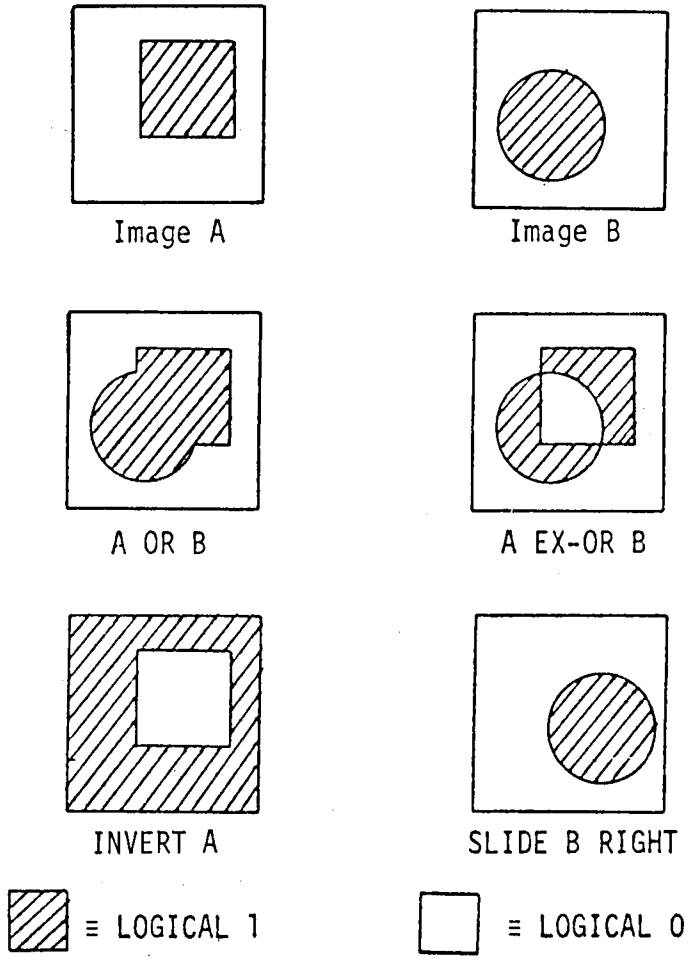


Figure 4. An example of elementary tse operations on typical images.

Tse logic gates which implement functions other than the primitive logical operations have been proposed. One of these gates, the contractor, has been found to be useful in most tse computer structures. The contractor is a control device which indicates the presence of any 1-elements in a tse. If there are no 1-elements in any position of the binary image, the output of the device is logic-0, otherwise the output is logic-1. This device is different in that the input is a tse, but the output is a single-bit logic signal. A device of this type is necessary for implementing conditional image operations. Other special tse logic devices can be found in Reference 5, Appendix A.

The basic tse gates can be interconnected in much the same manner as conventional logic components to form structures which perform useful functions. In order to realize more efficient utilization of components in a complex tse structure, some method of controlling the propagation of images must be provided. To facilitate the switching of paths along which an image will travel, all active tse devices are assumed to have a one-bit control line for turning the electro-luminescence on and off. In the off state, the output tse is a zero-tse; that is, all elements in the array are logic-0. The use of this control scheme is illustrated in Figure 5. Assuming that only one of the three control lines is active at any time, the circuit can execute a SLIDE UP, SLIDE DOWN, or a NO-OPERATION.

In the sections which follow, a parallel algorithm for counting the number of 1-elements in a binary image will be implemented using tse logic devices. To provide a basis for comparison, different tse hardware

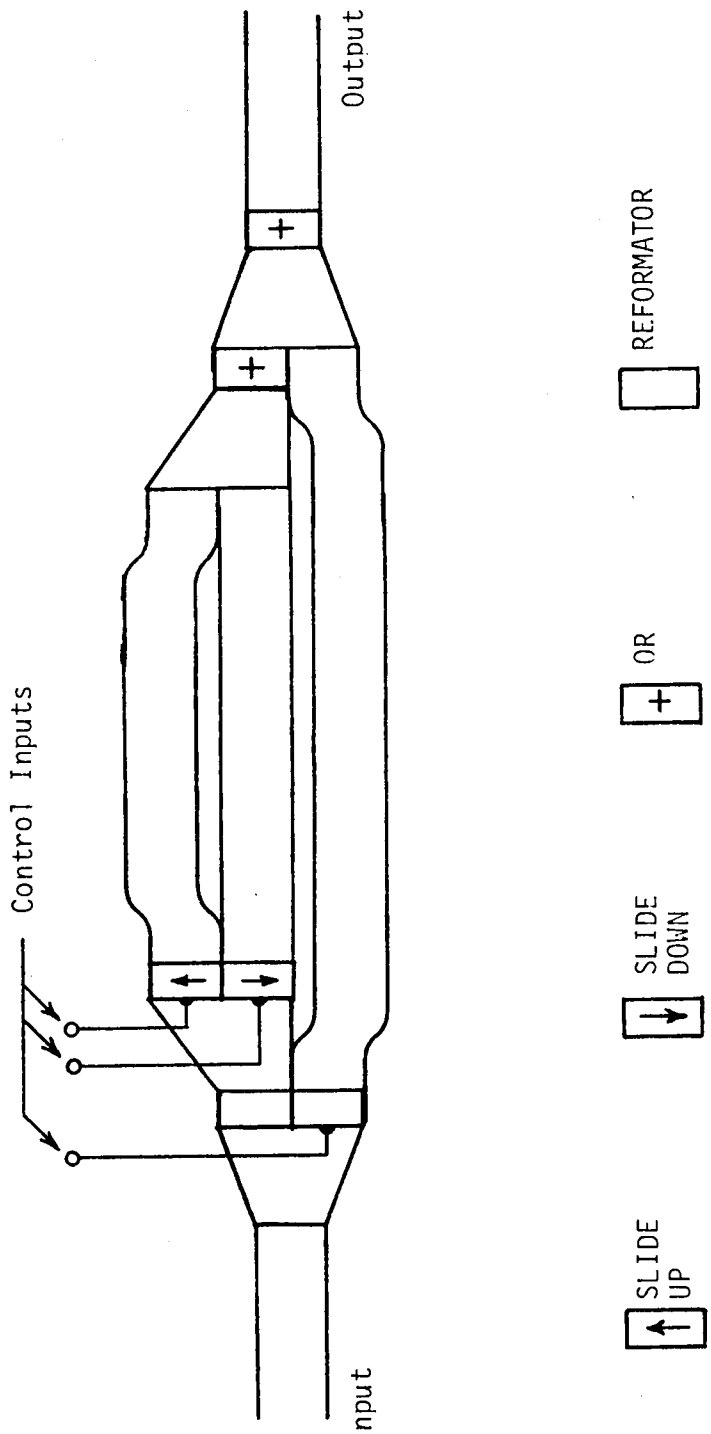


Figure 5. Control method used for switching of image paths.

structures are presented. In some cases, image processing rates show very significant improvement over those of conventional processors.

CHAPTER 3

A PARALLEL COUNTING ALGORITHM

In earth resource applications, techniques of pattern recognition are applied to the classification of terrain or surface features. After the classification process, the measurement of the area of an identified region is desired in many instances. For example, a typical application of earth resource technology might involve the mapping of the bodies of water in a certain land region. After the portion representing water is identified in each frame, the total area of the water surface is desired, since this information is important to the classification of the land region by percentage of water. Figure 6 illustrates a tse image plane of a typical frame after the identification of bodies of water. The desired area is represented by the 1-elements. Each element in the region classified contributes a partial area to the total. The problem of area measurement is solved by counting the number of 1-elements in the tse. In a conventional digital computer, this measurement is attained through a sequential decision process, one element at a time. The parallel counting algorithm of Strong [5, Appendix F] achieves the solution differently. A few of the descriptive characteristics of the algorithm are presented in this section.

The counting algorithm is applicable to any binary image (or tse), 2^m rows by 2^n columns, where m and n are any nonzero, positive integers. After the application of the algorithm, the result is also in the form

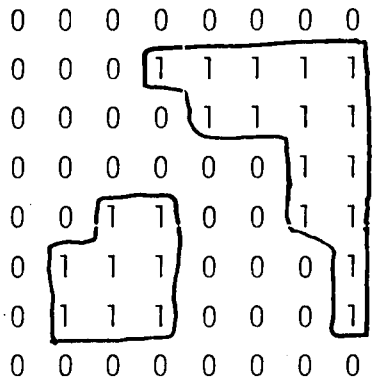


Figure 6. A tse plane in which each 1-element represents a classified region whose area is desired.

of an image. However, the desired information, which is a binary number indicating the number of 1-elements in the image, is found in the bottom row of the image. The remaining elements of the image are all zero. This is illustrated in Figure 7 for m,n = 3. The original image of Figure 6 is seen to contain 23 1-elements. Figure 7, which is the result of the application of the algorithm, has the number 00010111 in the bottom row. This is seen to be the binary representation of the number 23. The method by which the image of Figure 7 is generated from the original image is outlined as follows.

Basically, the counting process which is carried out by the algorithm consists of a number of iterations, each of which generates partial sums over the image. Each successive iteration generates these sums over larger areas, the final area being the entire image. To illustrate the process, consider the 4 × 4 binary image shown in Figure 8(a). Of course, the 1-elements and 0-elements in the image represent the logic level at each position. However, for this discussion, consider each as a one-bit binary number, a 1-element representing one unit of area to be contributed to the total, and a 0-element representing no area to be contributed. In the first iteration, each number in the first column and third column is added to the number immediately to its right. Thus, eight additions of two elements each are performed in parallel and the result is as shown in Figure 8(b). The eight groups or sectors over which the additions were generated are indicated in Figure 8(b), the encircled numbers being the two-bit first partial sums. In the second iteration, each group (partial sum) is added to the one immediately to its right, thus generating four second partial sums, as shown in Figure 8(c). Note that each group

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	1	0	1	1	1

Figure 7. The result of applying the counting algorithm to the binary image of Figure 6.

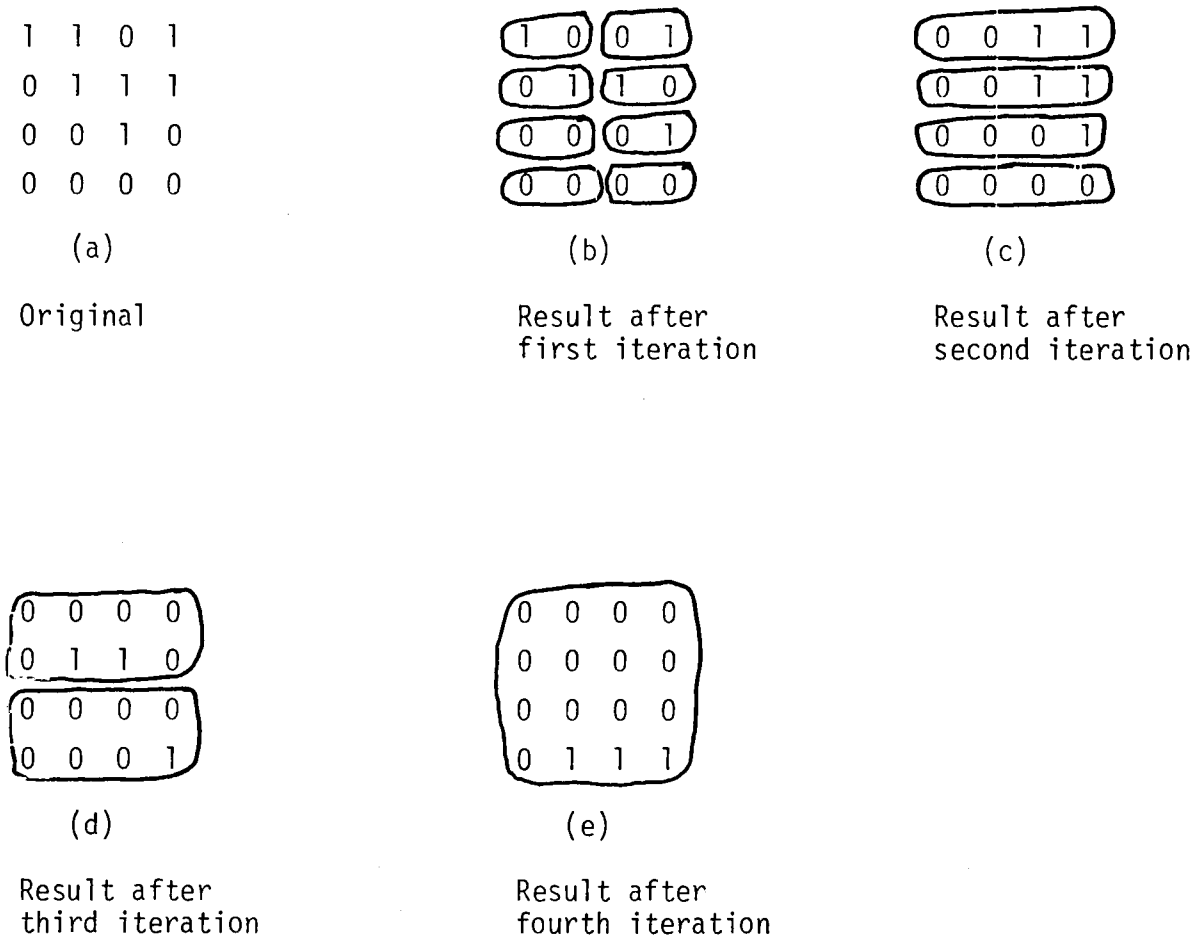


Figure 8. Result after each iteration of the counting algorithm, indicating partial summing method.

encompasses a full row, and the number in the group represents the number of 1-elements which were in that row of the original image (Figure 8(a)). Since the maximum number of 1-elements in any row is four, and only three bits are necessary to represent the sum of the elements in that row, the sum is right-justified. In the third iteration, the row sums are added to generate two partial sums, each over two rows, and, in the final iteration, the two partial sums are combined to form the sum over the entire image. Note here, that in addition to being right-justified, the sum in each group appears in the bottom row of the group.

As evidenced by the above description, the algorithm is inherently parallel. The algorithm could, of course, be implemented by various methods, such as by conventional programmed processors. However, the parallel nature of the algorithm makes it particularly well suited to the concept of tse logic. In the following paragraph, one iteration of the algorithm is reduced to a number of steps and described in terms of the necessary tse operations which were defined in the previous chapter.

To complement the illustration of the algorithm, an arbitrary image is used to show the effect of the execution of each step. For simplicity, the 8×8 (m,n = 3) image shown in Figure 6 (page 15) is chosen as the original, and is identified in the algorithm as Image A.

STEP 1. Create a new image, Image B, by performing a SLIDE 1 RIGHT operation on Image A. The result of this step is shown in Figure 9(a).

STEP 2. Mask the odd numbered columns of both Image A and Image B. This forces these columns to contain all

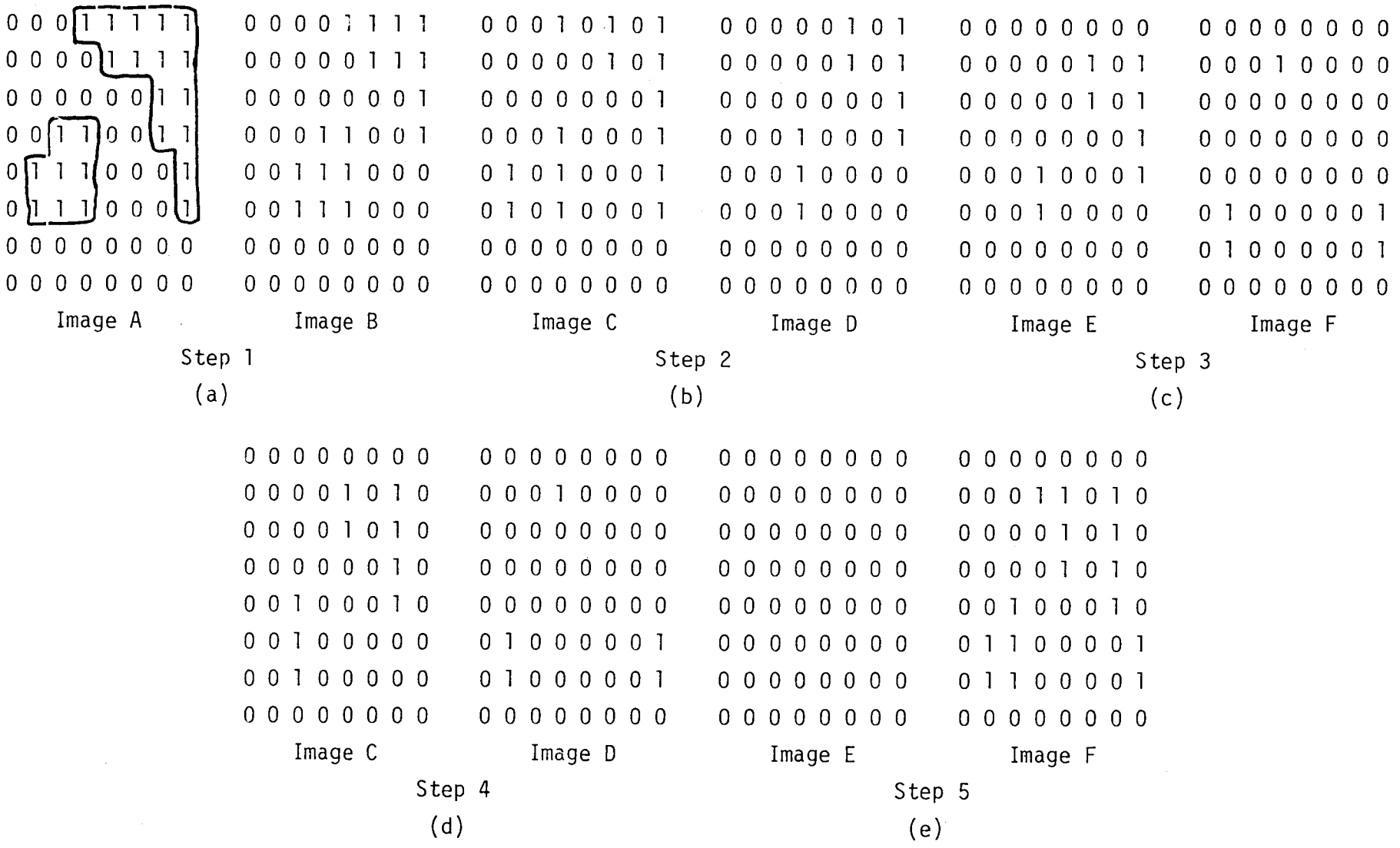


Figure 9. The results of applying the steps in the algorithm to Image A for one iteration.

zeros. The masked images are labeled Image C and Image D, respectively, and the result of this step is shown in Figure 9(b).

STEP 3. The AND and EXCLUSIVE-OR of Image C and Image D are generated and are labeled Image E and Image F, respectively. The result of this step is shown in Figure 9(c).

STEP 4. Image E is checked for all zeros. If Image E is all zeros, the iteration is complete. If Image E contains any 1-elements, a SLIDE 1 LEFT operation is performed on Image E. The result is labeled Image C, and Image F is relabeled as Image D. The result of performing this step is shown in Figure 9(d).

STEP 5. Repeat Steps 3 and 4 until Image E is all zeros. In the example, Steps 3 and 4 are repeated once. The result of performing this step is shown in Figure 9(e).

These steps describe the first iteration of the algorithm. All subsequent iterations use the same steps, with the exception that Step 1 and Step 2 are modified. The magnitude and direction of the Slide in Step 1 and the pattern of the masking in Step 2 are different for each successive iteration. These differences are outlined later. First, consider the effect of each step outlined above within the first iteration.

Step 1 creates a new Image B from the original Image A in which every element occupies the same position as the one to its right in the

original. By removing the odd columns from both Image A and Image B, Step 2 effectively creates Image C, which consists only of the even-numbered columns of the original image, and creates Image D, which consists only of the odd-numbered columns of the original image, shifted one position to the right. No information is lost in the masking process, as might first be concluded. When Image A is masked to create Image C, the effect is to retain the even-numbered columns of the original image. On the other hand, when Image B (same as Image A, displaced by one column) is masked to create Image D, the effect is to retain the odd-numbered columns of the original. Therefore, all information which was contained in the original image has been retained, and none is lost. The masking process actually removes redundant information from the Images. Step 3 adds, independently and concurrently, each element in Image C to the corresponding element in Image D. The sums (result of the EXCLUSIVE-OR operation) are placed in Image F and the carries (result of the AND operation) in Image E. Step 4 checks Image E to determine whether or not any carries were generated. If not, Image F is the result of the iteration. If any carries were generated, they must be added to the sums, using another EXCLUSIVE-OR and AND operation, thus generating another sum image and carry image (Image F and Image E). Step 5 indicates that the adding operation of Step 4 is repeated until the carry image shows all zeros, thus indicating that the addition has been completed. Note that Steps 3, 4, and 5 describe an addition process which is completely analogous to the operation of a conventional ripple-carry adder circuit [5, Appendix F]. Step 3 implements a function similar to that of the first half-adder in each cell of a conventional

adder. Steps 4 and 5 represent the method by which the second half adder of each cell adds the incoming carry to the sum from the first half-adder of the cell to generate an outgoing carry to be used by the next cell. Upon completion of all five steps, Image F is the result of the first iteration. This image is the original for the next iteration.

Each successive iteration performs the same basic operation over larger groups, the final iteration being the one which generates the sum over the entire image. Figure 10 illustrates the result after each iteration, along with the sectors over which summing is performed. The fact that any sector (encircled areas in Figure 10(a) through (f)) contains a binary number representing the number of 1-elements in the corresponding sector of the original (Image A) can be readily verified from the figure.

As previously stated, Step 1 and Step 2 must be modified for each iteration. For instance, the second iteration of the algorithm differs from the first, in that the slide operation is a SLIDE 2 RIGHT and the columns that are masked are the first and second, fifth and sixth, ninth and tenth, and so forth. Using the same approach as presented in the first iteration, the sum over horizontal groups of four will be generated when the second iteration is performed on the result of the first iteration, as shown in Figure 10(b).

Modification of the subsequent iterations is similar, until each horizontal group is the length of an entire row, as shown in Figure 10(c). At this point, the rows must be added to one another, in much the same manner as groups were added before. Therefore, Step 1 of the next iteration will be a SLIDE 1 DOWN, and the odd-numbered rows,

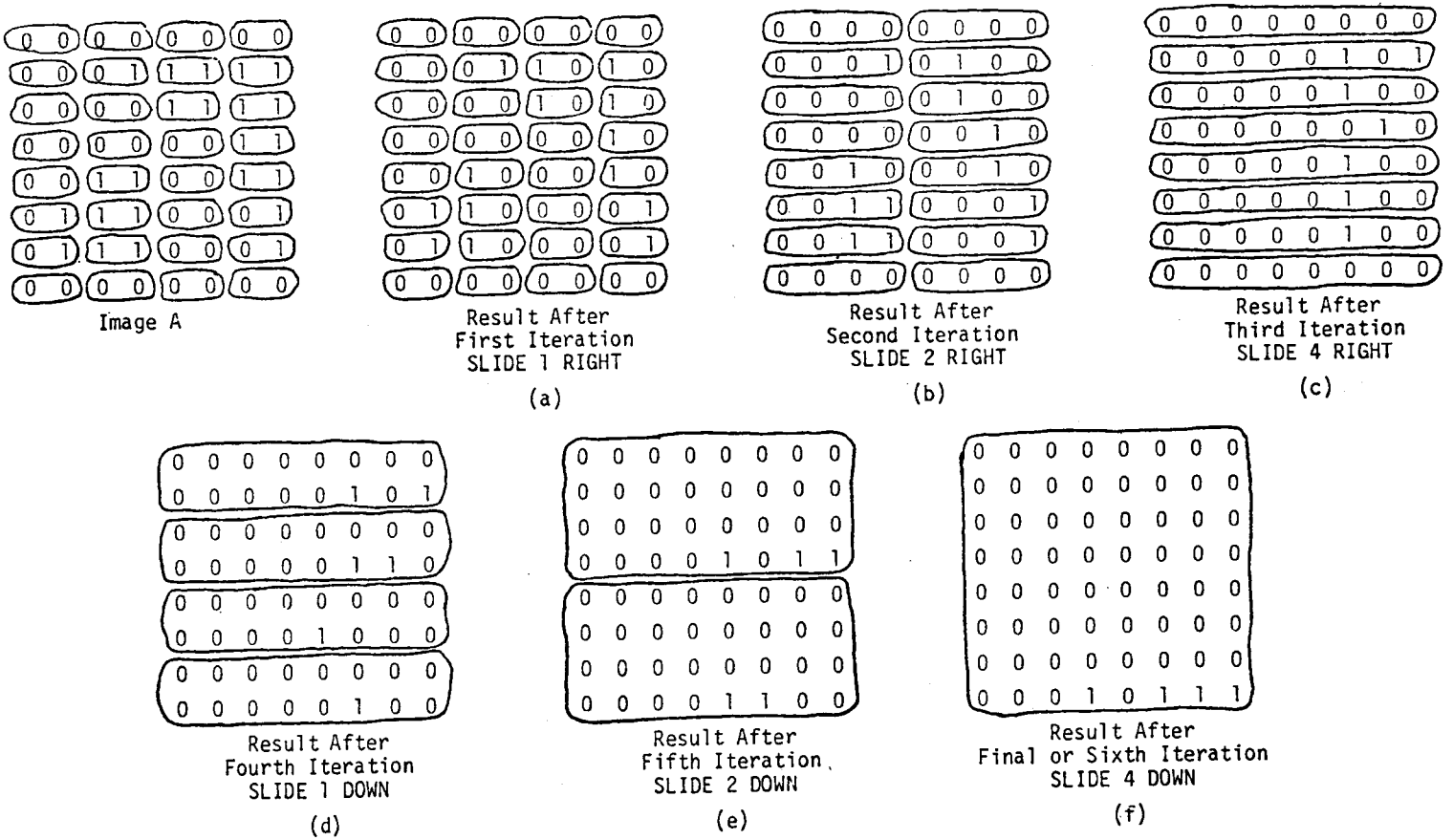


Figure 10. The results and significance of applying the algorithm to Image A after each iteration.

instead of columns, will be masked in Step 2. From that point, modifications are the same numerically as before, retaining the DOWN slide direction and row masking. Table 1 gives the necessary modifications as a function of the number of the iteration being performed.

Modified forms of the counting algorithm. The parallel counting algorithm is seen to be an efficient and potentially fast method of summing elements in a binary image. Results similar to those shown can be obtained by utilizing certain allowable variations in the basic steps of the algorithm. With these variations, the execution of the algorithm can possibly be greatly simplified for special types of input images. Some modified forms of the counting algorithm are presented below, using the image in Figure 6 (page 15) as an example.

Generating sums in sectors of an image. In the standard form of the algorithm, the magnitude and direction of the slide in Step 1 of each iteration is specified by the number of the iteration. For any image $2^m \times 2^n$, the order of the slide operations is 1, 2, 4 . . . , 2^{m-1} RIGHT, then 1, 2, 4 . . . , 2^{n-1} DOWN. Note, however, that the iterations need not be tied to this specific ordering. After performing any number of the iterations containing RIGHT slides, the iterations containing DOWN slides may be commenced. After any number of DOWN slides, more of the RIGHT slides may be performed, and so on. The iterations may be intermixed in any way, subject to only two restrictions. First of all, the actual order of the iterations containing slides of a certain direction should not be disturbed when

TABLE 1

SUMMARY OF MODIFICATIONS PER ITERATION

Operation	Modifications
Step 1 Slide Operation	SLIDE 2^{k-1} RIGHT, $k \leq n$ SLIDE 2^{k-n-1} DOWN, $k > n$
Step 2 Mask Operation	
Step 2 Mask Operation	(a) Mask following columns: For $k \leq n$ and $i = 1, 2, \ldots 2^{n-k}$ $(2^k i-2^k+1, 2^k i-2^k+2, \ldots, 2^k i-2^{k-1})$ (b) Mask following rows: For $k > n$ and $i = 1, 2, \ldots, 2^{m+n-k}$ $(2^{k-n} i-2^{k-n}+1, 2^{k-n} i-2^{k-n}+2, \ldots, 2^{k-n} i-2^{k-n-1})$

k = Number of the current iteration. 2^n = Number of columns in the image.

2^m = Number of rows in the image. $m + n$ = Total number of iterations per image.

iterations in the other direction are inserted between them. For example, if the iteration containing the SLIDE 4 RIGHT is performed at a certain time, then no matter how many iterations containing DOWN slides are performed, the next RIGHT slide iteration will be a SLIDE 8 RIGHT. Of course, the same applies when RIGHT slides are inserted between DOWN slides. The second restriction concerns the size of the partially summed sectors which may be generated as a result of performing the iterations in a different order. Certain orderings of the iterations will generate sectors whose row lengths may not be large enough to contain the binary number representing the number of elements in the sector. For instance, consider the 8×8 image in Figure 11 to which the algorithm will be applied. The order of the iterations is chosen to be the SLIDE 1 RIGHT iteration, followed by the SLIDE 2 RIGHT iteration, followed by the SLIDE 1 DOWN, SLIDE 2 DOWN, and SLIDE 4 DOWN iterations, as depicted in the figure. After the application of these five iterations, the result indicates partial summing over two sectors, each 8 rows by 4 columns. However, note that there are 19 1-elements in the left half of the original image, and that the binary form for 19 (10011) cannot be placed in the bottom row of the corresponding sector. The most significant bit is lost, thus introducing an error in the computation. Clearly, the result after the final iteration (SLIDE 4 RIGHT) is incorrect.

Within the restrictions, any rearrangement of the order of the iterations will generate the same final result. The real significance of the rearrangement is that by properly choosing the sequence, then omitting one or more of the iterations, the end result will indicate partial summing over a number of sectors of the original image. This

1	1	1	1	1	1	1	0
1	1	0	0	1	0	0	0
1	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0
0	0	1	1	1	0	0	0
0	1	1	1	1	1	1	1
0	0	1	0	0	1	1	0
0	0	1	0	1	1	0	0

Original

1	0	1	0	1	0	0	1
1	0	0	0	0	1	0	0
1	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0
0	0	1	0	0	1	0	0
0	1	1	0	1	0	1	0
0	0	0	1	0	1	0	1
0	0	0	1	1	0	0	0

First Iteration

0	1	0	0	0	0	1	1
0	0	1	0	0	0	0	1
0	0	1	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	1
0	0	1	1	0	1	0	0
0	0	0	1	0	0	1	0
0	0	0	1	0	0	1	0

Second Iteration

0	0	0	0	0	0	0	0
0	1	1	0	0	1	0	0
0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	1
0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0

Third Iteration

Figure 11. Example of partial summing where an error is generated.

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	1	0	0	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	1	1	1	1	0	0	1

Fourth Iteration

	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0
error here→	0	0	1	1	1	1	0	1

Fifth Iteration

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1 ← incorrect

Figure 11. (continued)

capability could be useful for some applications which require finding average densities in different partitions of an image.

As an example, consider the 8 × 8 image shown in Figure 12(a). In addition to finding the total number of 1-elements in the image, the distribution of these elements over the four quadrants is also desired. To achieve this, the order of the iterations is changed to SLIDE 1 RIGHT, SLIDE 2 RIGHT, SLIDE 1 DOWN, and SLIDE 2 DOWN. After these four iterations, the desired partial sums are available, as shown in Figure 12 (e). To complete the operation, the remaining two iterations, a SLIDE 4 RIGHT and SLIDE 4 DOWN, are performed.

Simplification of the algorithm for clustered elements. When the elements to be counted do not cover most of the total image frame, some of the iterations may possibly be omitted. Counting of clustered elements, those which lie totally within some smaller area of an image, requires only as many iterations of the algorithm as would the smallest $2^m \times 2^n$ (m,n are integers) image which will enclose the cluster. Once this reduced image size is determined, the partial summing procedure described above is applied to the image. The process is complete when the size of the partially summed sectors is the same as the reduced image size for the cluster. Provided that every element of the cluster was located within a single sector of the original, the result shown in that sector will actually be the desired sum. In order to ensure that the cluster lies entirely within the sector, a number of slides DOWN and RIGHT should be applied to the original to relocate the cluster to the extreme bottom right of the image. This will also cause the result to

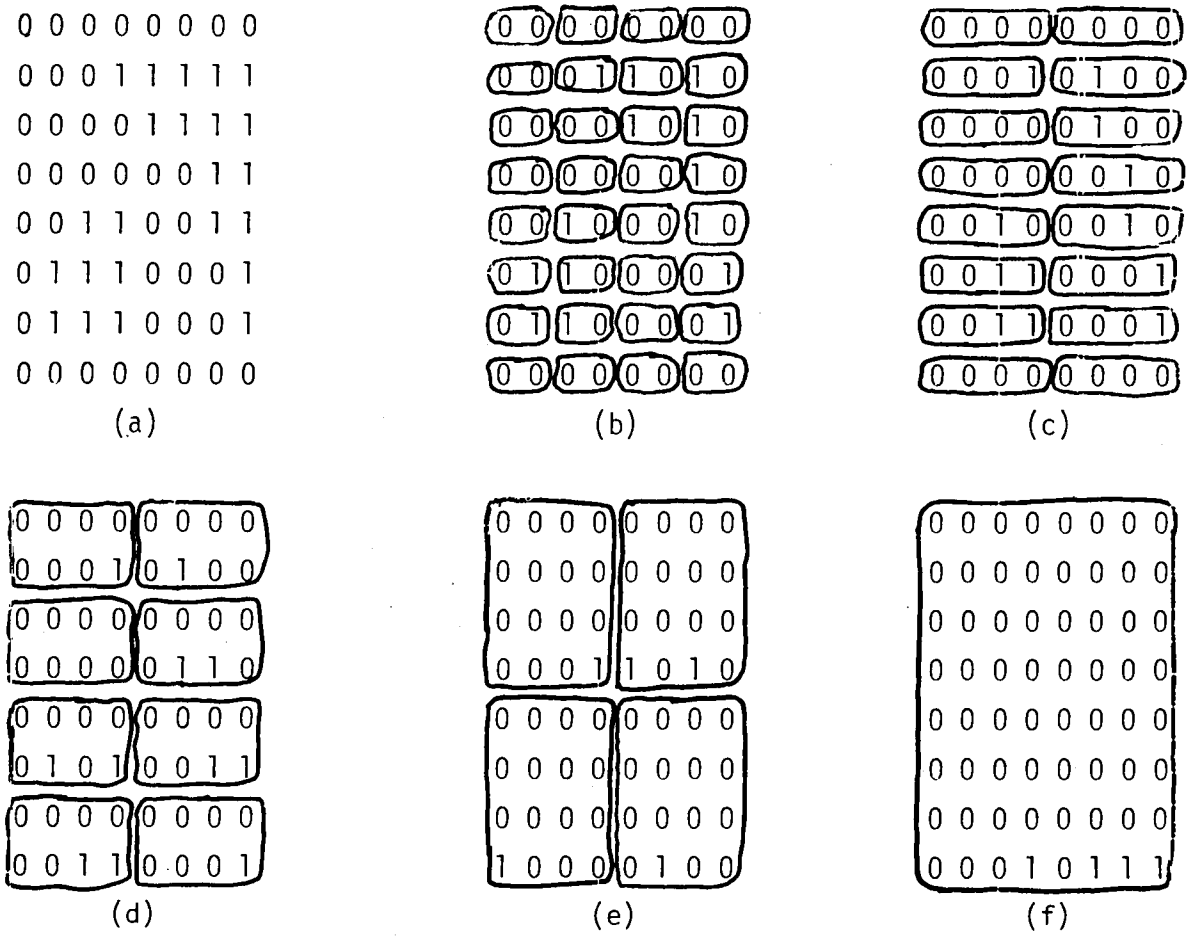


Figure 12. Example of modified counting algorithm.

always be located in the normal bottom right position. Thus, by elimination of some of the iterations, the processing time can be reduced.

The simplification is practical only if there is some method of determining the size of the cluster, adjusting the pattern of the iterations, and determining the number of slides needed to relocate the cluster. In particular, if the execution of the algorithm is under the control of some type of monitoring device, an overall reduction in processing time may be realized by checking every image before starting the counting process, then making the necessary adjustments, if any.

As an example, suppose that the binary image of Figure 13(a) is the result of some classification process, and the encircled areas have been found to be the areas of interest. However, for some reason, only the area at the lower left is to be measured. By some further classification process, the area at the upper right is removed, as shown in Figure 13(b). Upon checking the image, the monitoring device finds that the smallest area which is 2^m rows by 2^n columns and contains all of the elements to be counted is 4×4 $(m,n = 2)$. To relocate the image to the bottom right corner, simple slide operations are performed, resulting in the image of Figure 13(c). The partial summing procedure is then applied, with a final sector size of 4×4. Thus, the execution time for the algorithm will be shorter.

More significant reductions in processing time will be realized when image planes larger than 8×8 are used.

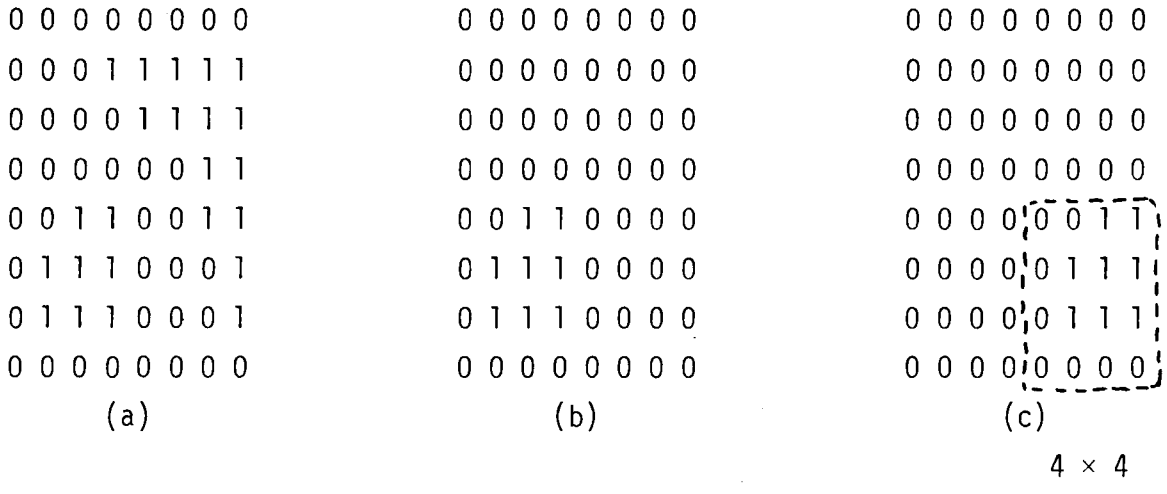


Figure 13. Example of simplification of the algorithm.

CHAPTER 4

TSE HARDWARE IMPLEMENTATION OF THE COUNTING ALGORITHM

Combinational circuit approach. As described in the chapter on tse components, tse gates may be interconnected to form more complex tse functions in much the same manner as conventional gates are interconnected to form more complex boolean functions. Using a straightforward approach, one method of implementing the counting algorithm is to simply connect the proper gates together in such a way that the desired steps are performed on the image as it propagates through the network. Conceptually, this is the simplest and most direct realization of the algorithm.

The complete circuit for implementation of the algorithm may be viewed as a group of cascaded "black boxes," where each box has only an input tse and an output tse, and performs a single iteration of the algorithm. This arrangement is illustrated in Figure 14. Since the magnitude and direction of the slide and the pattern of the mask is different for each iteration, each box will have different contents. However, the following description of the contents of one box is a general one, and the differences between each box can be summarized.

The hardware needed to perform one iteration of the algorithm is shown in Figure 15, and the steps within the iteration can be readily associated with certain gates in the structure. Tse Gate 1 is the slide gate which performs the slide operation in the appropriate mask pattern, as outlined in Step 2, by allowing information to pass through the AND

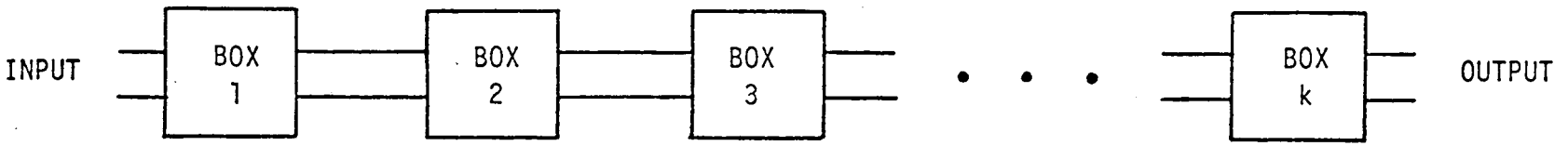


Image Size = $2^m \times 2^n$

$k = m + n$

Figure 14. Combinational network implementation.

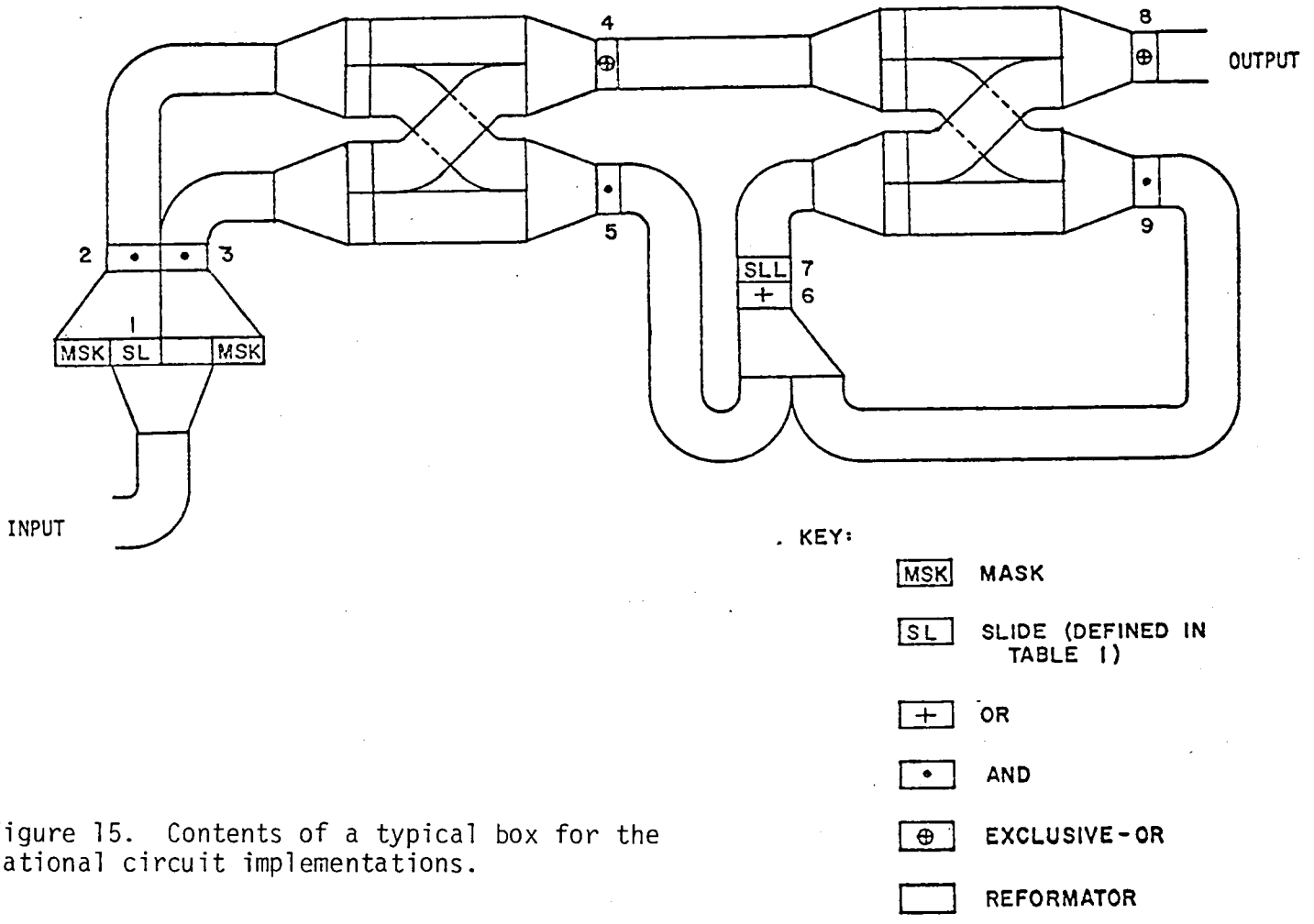


Figure 15. Contents of a typical box for the combinational circuit implementations.

gate where a 1 appears in the same position in the mask, and generating a zero elsewhere.

As described in Step 3, tse Gates 4 and 5 generate the sum and carry images, respectively. Tse Gates 6, 7, 8, and 9 effectively perform Step 4 and Step 5, but in a somewhat different manner. The carry image generated by Gate 5 is one input of the OR gate (Gate 6), whose other input is initially clear. The carry image then propagates through Gate 6 unaffected to Gate 7, where a SLIDE 1 LEFT is performed on the image. This will displace any carry bit generated to the left, as is expected when two binary numbers are being added. Remember, however, that many sets of two numbers are being added simultaneously in this case. The displaced carries are then fed back into one input of Gates 8 and 9, where they are added to the previously generated sums. Again, this follows directly from the case for binary numbers. When the carries are added to the sums, new carries may be generated. These new carries are, in turn, shifted left and added to the sum, which may generate even more carries. Thus, carries will propagate around the feedback loop until no new carries are generated. This is analogous to binary addition. In fact, the entire summing operation performed by Gates 4, 5, 6, 7, 8, and 9 is equivalent to having 2^{m+n} (the total number of image elements) one bit full adders connected in groups of p, where p is the number of image elements being summed per group in the particular iteration. Hence, the true potential of parallel processing is apparent.

After the feedback loop is stabilized (that is, no new carries are being generated), the output can be assumed to be correct. Actually, there is no indication as to when this condition has been reached;

therefore, the amount of time required for worst-case carry propagation delay must elapse before the output can be assumed to be stable and correct for the next iteration.

The number of gates required for this implementation of the counting algorithm for a 512 × 512 image is 360. Based on an initially projected power consumption of 3 watts per gate, the total power required for the circuit is 1080 watts. The processing time, taken to be the delay from the introduction of the original image to the input of the circuit to the time at which the output is stable (worst-case), is 756 tse gate delays. In terms of a projected delay of 5 milliseconds per gate, the processing time for this configuration is 3.78 seconds. This corresponds to an image processing rate of 0.26 images per second. These characteristics are summarized in the next chapter (see Chapter 5, page 70) where they are also compared to those of other implementation of the algorithm.

Pipeline network implementation. One of the most serious disadvantages of the combinational circuit implementation is the large propagation delay from the input to the output. Although this delay cannot be easily reduced, a higher rate of image processing can be realized by considering a pipeline structure, such as the one illustrated in Figure 16. The circuit is basically the same as for the combinational circuit implementation, except that intermediate tse registers have been placed between each iterative box. An implementation of the tse register is presented in Figure 17. These registers temporarily hold intermediate results so that data flow through the structure at a constant rate,

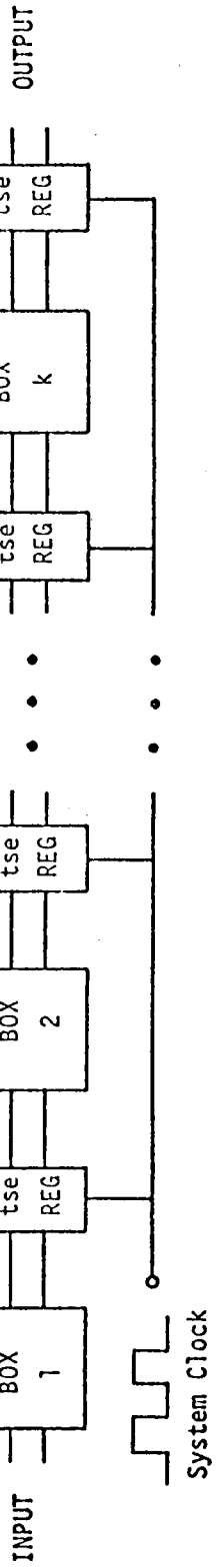
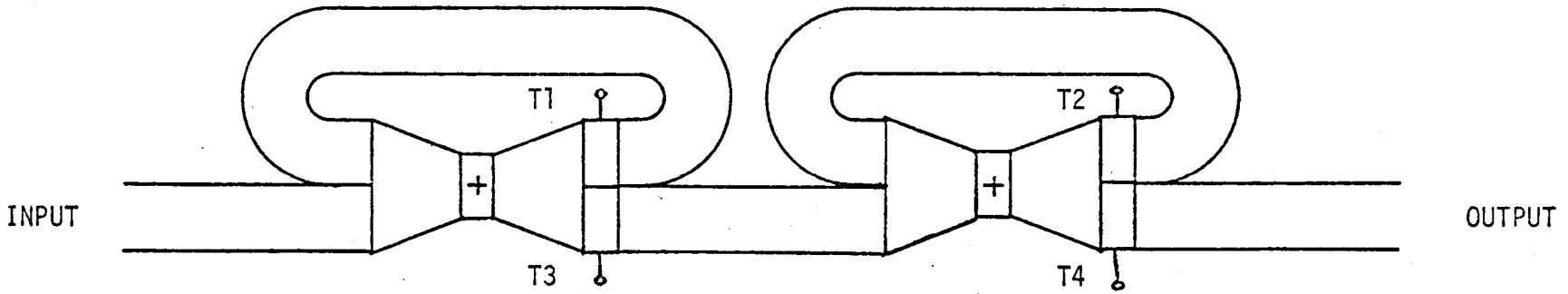


Figure 16. Pipeline network implementation to increase image processing rate.



T1, T2 Clear Image Control
T3, T4 Transmit Image Control

Figure 17. Latch implementation for the tse register.

controlled by the frequency of the clock. The clock is set at a frequency such that its period is slightly greater than the worst-case propagation delay of the slowest box. Between each clock pulse, a completely new image can be placed at the input. The result of that image will appear at the output after m+n-1 clock pulses. Although the actual delay from the input to the output for any single image is greater, the rate of processing (number of images per unit time) is increased.

The pipeline implementation of the algorithm requires a total of 468 gates, representing a power consumption of 1404 watts. For a single image, the number of gate delays is 1368, corresponding to a processing time of 6.84 seconds. However, the processing time per image for a number of images being input to the circuit at the clock frequency is 0.38 seconds. The image processing rate for this implementation is 2.63 images per second, an improvement over the combinational circuit implementation.

Implementation using a programmable tse processor. Present-day technology and projections to the near future indicate that the early tse components will be bulky, have large power consumption, and will not have a suitable degree of fiber alignment to allow easy interconnection of gates. Therefore, initial efforts will tend to favor structures which are as simple as possible, even though repetitive use of the structure may require considerably greater execution times per image.

Hardware considerations. As an illustration of the type of structure proposed, consider a unit which contains only the most

elementary gates AND, OR, NEGATE, SLIDE 1 RIGHT, SLIDE 1 LEFT, SLIDE 1 UP, and SLIDE 1 DOWN. Consider also a set of tse registers, as many as needed to hold intermediate results, and a control and bus scheme which allows any register to be directed through any gate and the result to be directed to any register. Using this machine, a SLIDE 64 RIGHT operation, for example, would be implemented by executing the SLIDE 1 RIGHT operation 64 times on the same register. The EXCLUSIVE-OR function of Register A and Register B, for example, would be implemented by performing the proper sequence of AND, OR, and NEGATE operations to generate AB' + A'B. Registers A and B, and three other registers for intermediate results would be used in generating this operation.

The machine described in the previous paragraph is the general form of a tse computer. Although inherently slow, the machine has the advantage of being structurally simple and versatile in that the control unit can be reprogrammed to execute virtually any function or algorithm.

As intermediate results are generated, they are stored in certain tse registers whose outputs must be directed along different paths according to the next desired operation. Therefore, some method for the switching of image paths is necessary. Recall that Figure 2 (page 8) shows a typical gate whose fan-out has been increased to four using image duplicators. The four outputs are then connected to four different paths through reformators. In order to cause the image to propagate through only one of the four paths, the control bit to the reformators in the other three paths is turned off. This will allow all-zero images to propagate through these paths, which is the same as having them disconnected from the source of the image. Hence, the method of control

in a tse structure is to switch the active tse elements on and off in the proper sequence. Since the control signals switch entire images and not individual elements within an image, the generation of these signals can easily be controlled by a small conventional binary computer or microprocessor.

Shown in Figure 18 is a microprogrammed control system for a tse computer organized around a microprocessor. The use of the Intel 8080 microprocessor is projected in this paper, although any other microprocessor would be suitable. The memory consists of conventional ROM and RAM organized as 8-bit words. The microprogram, which generates the control sequences for the tse processor, is stored in the ROM portion of the memory. For ease of modification, the main program is stored in the RAM portion of the memory.

The various operations performed by the tse processor (AND, OR, NEGATE, and so forth), along with program control functions (Branch, Halt, Register Transfer, and so forth) comprise the instruction set. These instructions, which are coded in some 8-bit format, are used to structure the program for the counting algorithm. This program is stored in the memory associated with the microprocessor. As each tse instruction is encountered during execution of the program, the microprocessor decodes the instruction under the direction of a system monitor program. The microprocessor then outputs the appropriate control words through the output port to the tse control lines in a proper sequence. When a program control instruction is encountered, its effect will be restricted to the microprocessor control system, and no control of tse components is generated.

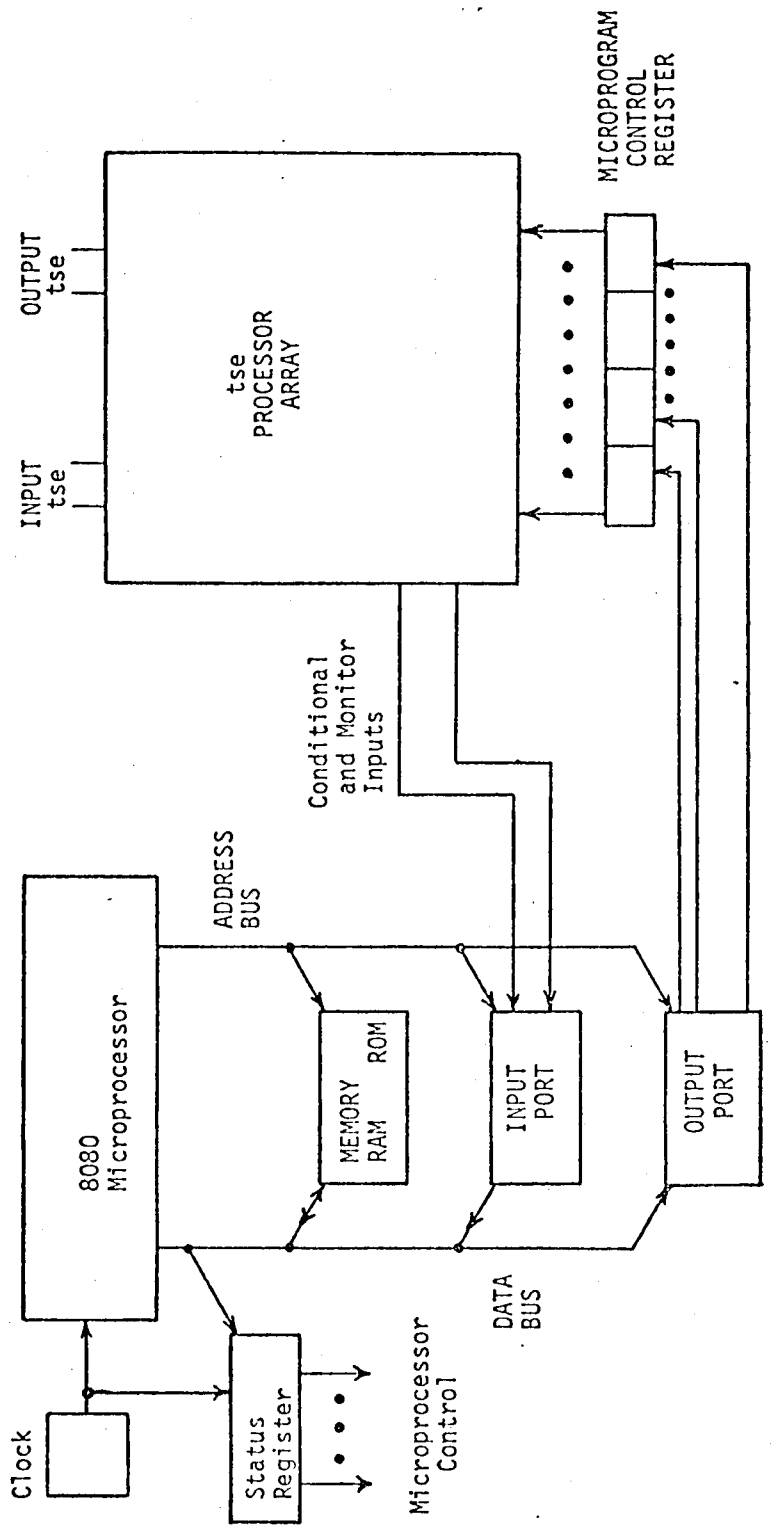


Figure 18. A microprogram, microprocessor control unit concept for the tse processor.

In tse conditional instructions, the status of the tse processor must be monitored by the control, as in checking an image for the presence of any 1-elements and branching if true. For this purpose, certain devices which monitor tse images and convert the status of these images to a few binary bits are required. The outputs of these devices are connected to the input port, where the information may be addressed by the microprocessor control.

Figure 19 presents a block diagram for the tse processor. The architecture includes these subsystems: a tse Logical Operations Unit, an Image Bus, tse Register, fixed (Read Only) tse Registers, and tse monitor devices. Control inputs to the logical devices are not shown; however, each active device which is involved in the switching of image paths represents an incoming control line. Each subsystem of this organization is described below.

The organization of the tse Logical Operations Unit is illustrated in Figure 20. An image placed at Input 1 will be directed through the SLR, SLL, SLU, SLD, NEG or NOP gates to the output. NOP is used in tse register transfer operations. When another image is placed at Input 2, the two images can be directed through the AND or OR gate to the output. Input 2 can be disabled and the SLR or SLD gate in the feedback path enabled to perform the Horizontal or Vertical Sweep operation [5] on the image at Input 1. Sweep operations are required to generate the 18 masks ($m+n = 18$) for an image size of 512×512 used in the counting algorithm without the requirement for a large ROM. The latch at the output is a register which retains the result of the operation until cleared.

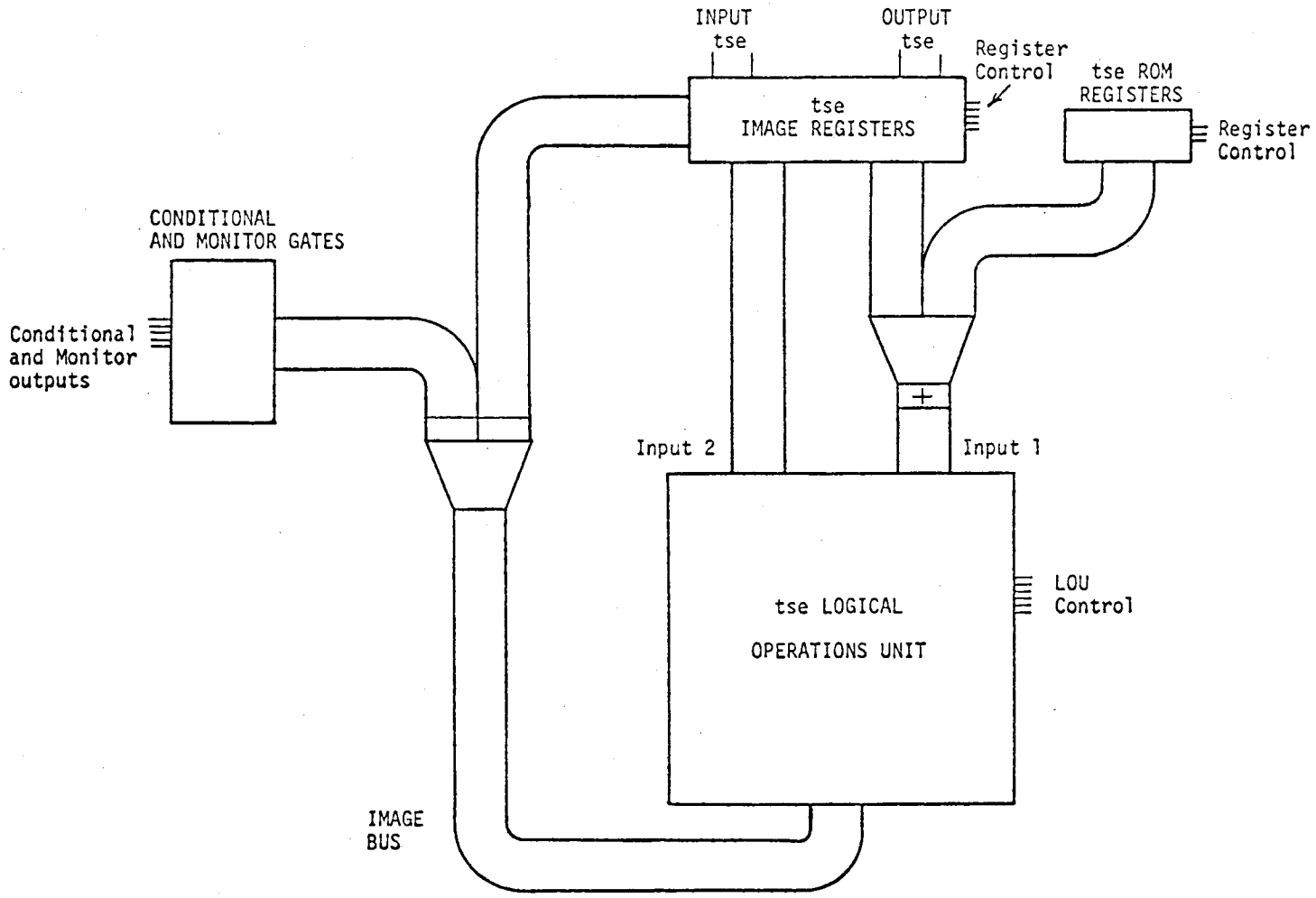


Figure 19. Block diagram for the tse processor.

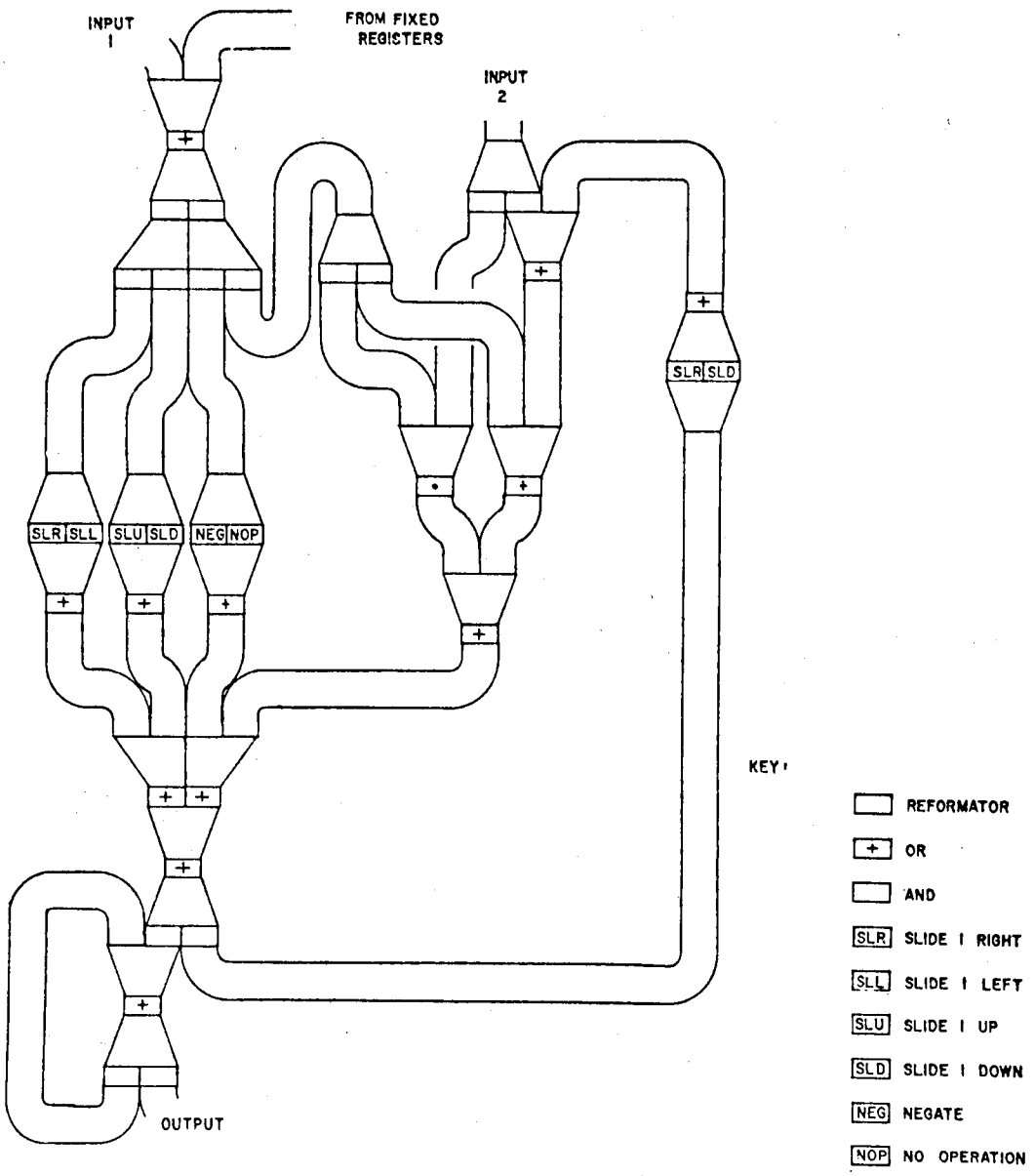


Figure 20. Organization of the tse Logical Operations Unit.

A primary purpose of the Data Bus is to transmit an image from the output latch of the Logical Operations Unit to the tse Image Registers, where the image is gated into a destination register. Also connected to the Data Bus are two special tse devices as shown in Figure 21. One of the devices is a tse contractor gate. This device serves as a zero-image or zero-tse detect bit when monitored by the control unit. Another special device on the Data Bus is a tse row output gate, which transmits to the control unit input port the information in the eight rightmost elements of the bottom row of the image latched on the Data Bus. This device allows the control unit to have access to the numerical result of the counting algorithm.

The tse Image Registers are connected to the Data Bus, as shown in Figure 22. For implementation of the counting algorithm, eight tse registers are required. The output of any one of these registers can be directed to Input 1 of the Logical Operations Unit, with the exception of one register which will be labeled Register R0. This register is connected directly to Input 2 of the Logical Operations Unit and is the only path connected to that input.

Two fixed (read-only) image registers are used to store certain pattern images, as presented in Figure 23, for an 8×8 image. These patterns are used in conjunction with the sweep operations to generate each mask required by the counting algorithm. The method used to generate these masks is outlined below.

Consider the simple tse circuit shown in Figure 24(a). Let the image of Figure 24(b) be placed at the input. The SLIDE 1 RIGHT gate in the feedback loop will cause the contents of any column in the input

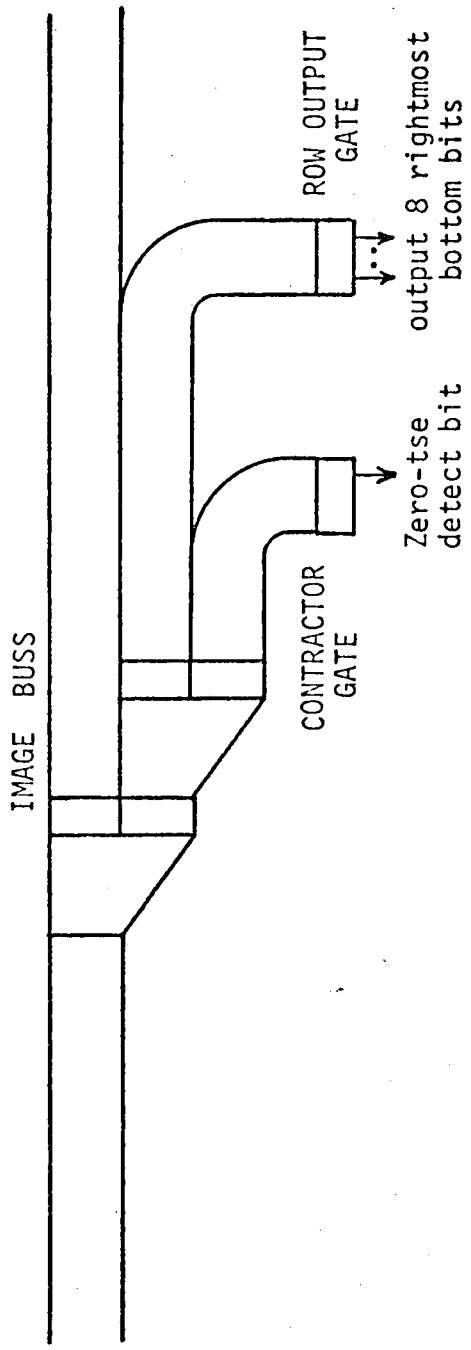


Figure 21. Image buss with conditional and monitor devices.

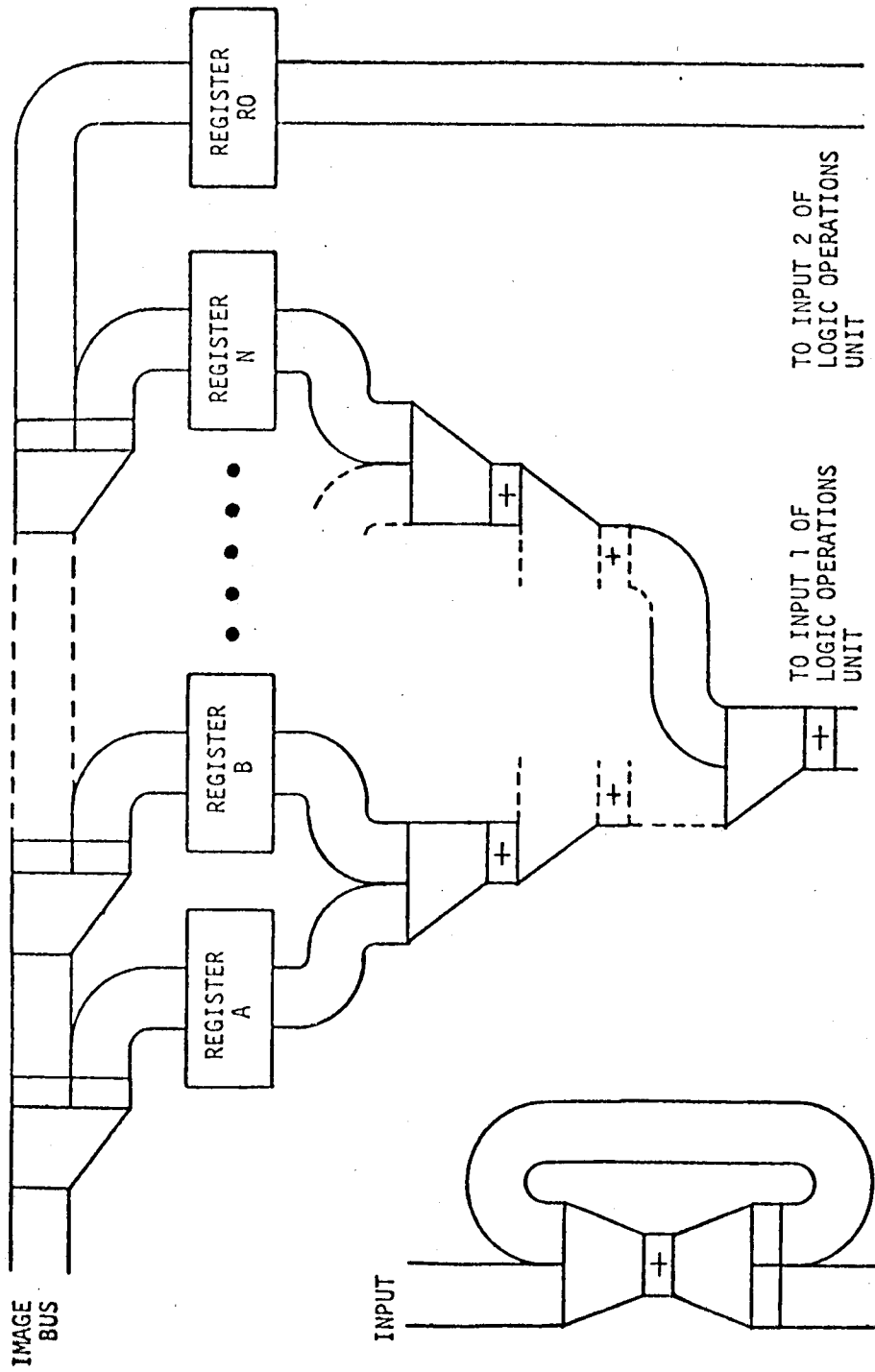
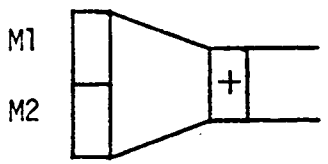


Figure 22. Organization of the tse Image Registers.



0	0	0	0	1	0	0	0
0	1	0	0	1	0	0	0
0	0	1	0	1	0	0	0
0	1	1	0	1	0	0	0
0	0	0	1	1	0	0	0
0	1	0	1	1	0	0	0
0	0	1	1	1	0	0	0
0	1	1	1	1	0	0	0

M1

0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	1
0	0	1	1	0	0	1	1
0	0	0	0	1	1	1	1
1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

M2

Figure 23. Organization and content of tse ROM registers.

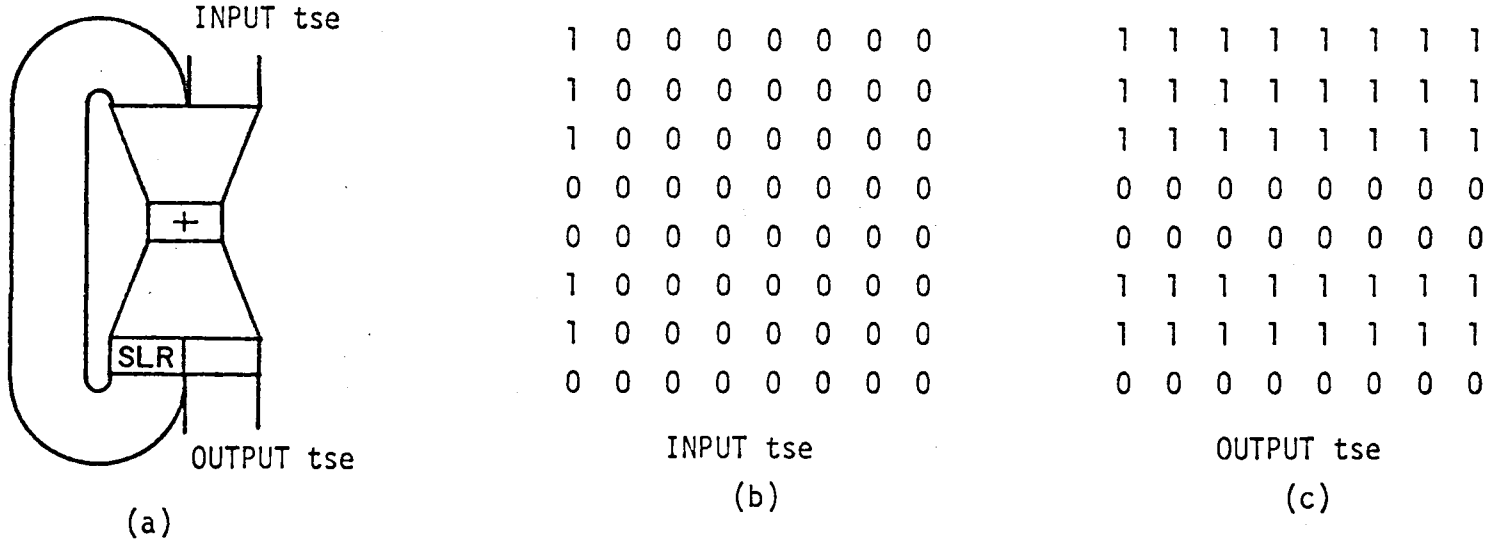


Figure 24. Implementation of a horizontal sweep device [5].

image to be duplicated in the column to its right. The duplicated column will propagate through the OR gate and be duplicated again. As shown in the output image of Figure 24(c), the column will continue to duplicate, or "sweep," across the image until all columns have been duplicated. This operation is called a HORIZONTAL SWEEP [5]. A VERTICAL SWEEP is implemented in a similar manner. When the OR gate in the Logical Operations Unit is enabled, and the SLR or SLD gate in the feedback path of the Logical Operations Unit is enabled, a circuit similar to that of Figure 24 is realized, and a sweep operation is performed on the image at Input 1.

The sweep operation is utilized in the generation of masks as illustrated in the following example. During the execution of the third iteration of the counting algorithm, the following mask is required for an 8 × 8 image:

0 0 0 0 1 1 1 1
0 0 0 0 1 1 1 1
0 0 0 0 1 1 1 1
0 0 0 0 1 1 1 1
0 0 0 0 1 1 1 1
0 0 0 0 1 1 1 1
0 0 0 0 1 1 1 1
0 0 0 0 1 1 1 1

To generate this mask, two images are formed by a SLIDE 3 UP and a SLIDE 4 UP on the contents of M2 (Figure 23). The results of these operations are ANDed. This places the desired mask pattern in the top row of the tse, and 0-elements elsewhere. A VERTICAL SWEEP operation is

then performed to duplicate the pattern downward, and the result is the required mask for the third iteration.

Software considerations. The instruction set for the tse processor includes the operations performed by the Logical Operations Unit. Since the processor was designed on the basis of a minimal hardware structure, operations such as EXCLUSIVE-OR and slides of magnitude other than one are not included, but must be programmed by the user when required. In addition to the tse instructions, there must be available to the programmer certain program control instructions or tse-microprocessor instructions. These instructions are used for branching, subroutine call and return, indexing, halting, and so forth. The primary distinction between tse processor instructions and program control instructions is that the latter do not involve any transfer or modification of information by the tse processor array. Their effect is confined to the microprocessor control system.

The complete instruction set derived for the tse processor is presented in Table 2. This set is partitioned into a tse processor instruction set and a tse program control set. All tse registers are indicated by upper case letters, and all microprocessor system registers are indicated by lower case letters. Note that some of the program control instructions have double-precision (16-bit) capabilities. These are indicated by an asterisk (*). Double precision operations are necessary to allow for numbers which may exceed 2^8-1, or 255, as in the case where image sizes may exceed 256 × 256 elements. Of course, the use of double precision operations implies the extension of the

TABLE 2

TSE COMPUTER INSTRUCTION SET

Mnemonic	Instruction	Description
	tse Processor Instructions	
AND A	Logical AND	(A) • (R0) → A
OR A	Logical OR	(A) + (R0) → A
SLR A	Slide 1 Right	SLIDE R (A) → A
SLL A	Slide 1 Left	SLIDE L (A) → A
SLU A	Slide 1 Up	SLIDE U (A) → A
SLD A	Slide 1 Down	SLIDE D (A) → A
NEG A	Complement	NEG (A) → A
VSW A	Vertical Sweep	V. SWEEP (A) → A
HSW A	Horizontal Sweep	H. SWEEP (A) → A
MOV A, B	Register Transfer	(A) → B
JMZ A, n	Conditional Jump on null Image to Location n	If (A) = 0, GO TO n

tse Program Control Instructions

Mnemonic	Instruction	Description
	Microprocessor Instructions	
JMP n	Unconditional Jump to location n	GO TO n
JMZ a, n	Jump on zero to a location n	IF a = 0, GO TO n
MOV a, b	Register Transfer	(b) → a
LDI a, n	Load Immediate	n → a
INC a	Increment	(a) + 1 → a
CALL n	Subroutine Call	Store Program Control Register, Go To n
RET	Return from Subroutine	GO TO (location stored by CALL)
*DAD a	Arithmetic Left Shift (Multiply by 2)	(a) + (a) → a

TABLE 2 (continued)

Mnemonic	Instruction	Description
	Microprocessor Subroutine Instructions	
*SUB a, b	Double precision Subtract	(a) - (b) → a

associated register to 16 bits using some auxiliary register. Other control instructions are implemented by microprocessor subroutines, as in the case of *Sub a, b.

In order to program the processor to perform any task, the proper sequence of instructions must be stored in the working memory of the processor. Therefore, a coding scheme or format must be provided to represent each instruction. Tables 3 and 4 show a coding scheme for the instructions in Table 2. Although there are many possible formats, the one shown has been structured to require a relatively simple micro-program for decoding. This simplification reduces the amount of read-only memory required to store the microprogram, and also reduces the decoding time for the tse instructions.

A tse computer program for execution of the counting algorithm is presented in Table 5. The size of the image in this case is 512 × 512 elements. Note that some subroutines are called upon which perform frequently-used functions. These functions were omitted from the basic instruction set for simplicity. The subroutines are EOR, VSLR, VSLL, VSLU and VSLD. Subroutine EOR performs the EXCLUSIVE-OR operation on the images in Registers A and B. Subroutines VSLR, VSLL, VSLU and VSLD perform variable-length slide operations in each of the four directions. The image upon which the slide is performed is stored in Register A and the magnitude of the slide in Register z prior to the call. Since the four variable slide subroutines differ only in the slide instruction at location IOTA, only one of the subroutines is shown in the table.

TABLE 3

TSE PROCESSOR INSTRUCTIONS

Instruction	Number of Bytes	Coding
VSW A	1	1 0001 xxx
HSW A	1	1 0010 xxx
SLL A	1	1 0100 xxx
SLR A	1	1 0101 xxx
SLU A	1	1 0110 xxx
SLD A	1	1 0111 xxx
AND	1	1 1001 RRR
OR	1	1 1010 RRR
NEG	1	1 1011 RRR
JMZ	2	1 1101 RRR aaaaaaaa
MOV	2	1 1110 xxx $[RRRR]_s$ $[RRRR]_d$

aaaaaaaa = Branch Address

xxx = Don't Care

s = Source Register

d = Destination Register

Register Format (RRR or [RRRR]):

RRR:

A	000
B	001
C	010
D	011
G	100
L	101
M	110
N	111

$RRRR_s$ or $RRRR_d$:

A	0000
B	0001
C	0010
D	0011
G	0100
L	0101
M	0110
N	0111
M1	1000
M2	1001
R0	1100

TABLE 4

TSE PROGRAM CONTROL INSTRUCTIONS

Instruction	Number of Bytes	Coding
MOV	2	0 0001 xxx xx $[rrr]_s[rrr]_d$
LDI	3	0 1011 $[rrr]_d$ nnnnnnnn nnnnnnnn
SUB	2	0 0011 xxx xx $[rrr]_s[rrr]_d$
JMZ r	2	0 1010 rrr aaaaaaaa
JMP	2	0 0010 xxx aaaaaaaa
CALL	2	0 0100 xxx aaaaaaaa
RET	1	0 1100 xxx
DAD	1	0 1110 rrr
INC	1	0 1111 rrr
HLT	1	0 1101 xxx

aaaaaaaa = Branch or Call Address

nnnnnnnn = Immediate Bytes

Register Format:

a 000
b 001
c 010
v 011
w 100
x 101
y 110
z 111

TABLE 5

TSE COMPUTER PROGRAM FOR EXECUTION OF THE COUNTING ALGORITHM

Location	Mnemonic	Comment
	LDI 2, 1	Initiate Microprocessor Counter
	LDI x, 1	
	LDI y, 0	
ZETA	MOV L, A	Test for Row Counting or Column Counting
	MOV y, a	
	SUB a, 1	
	JMZ a, ALPHA	
	MOV x, z	Slide Input Image Right
	CALL VSLR	
	MOV A, B	Create Mask
	MOV M2, A	
	MOV w, z	
	CALL VSLR	
	MOV M4, R0	
	AND A	
	VSW A	
	MOV A, C	
ALPHA	JMZ y, BETA	
	MOV x, z	Slide Input Image Down
	CALL VSLD	
	MOV A, B	Create Mask
	MOV M1, A	
	MOV w, z	
	CALL VSLL	
	MOV M3, R0	
	AND A	
	HSW A	
	MOV A, C	

TABLE 5 (continued)

Location	Mnemonic	Comment
BETA	MOV C, RO	
	AND B	AND input image with mask
	AND L	
	MOV B, RO	Generate Sum, Store in C
	MOV L, A	
	CALL EOR	
	MOV A, C	
	MOV L, B	Generate Carry, Store in B
	AND B	
	MOV B, RO	Test for additional carry bits If none, begin next grouping
DELTA	OR G	
	MOV G, RO	
	MOV D, A	
	CALL EOR	
	JMZ A, GAMMA	
	MOV G, D	Add Carry to Sum, Generate New Carry
	MOV D, A	
	SLL A	
	MOV A, RO	
	MOV C, A	
	CALL EOR	
	MOV A, L	
	AND C	
	MOV C, G	
	JMP DELTA	Repeat Carry test
GAMMA	DAD x	Increment Counters
	INC w	

TABLE 5 (continued)

Location	Mnemonic	Comment
	LDI v, 512	Test for Completed Row or Column Summing
	SUB v, x	
	JMZ z, EPSILON	If Column Summing Completed, Start Row Summing
	JMP ZETA	
EPSILON	JMN a, ETA	If Row Summing Complete, Stop
	LDI x, 1	Reset Counters, Start Next Grouping
	LDI y, 1	
	LDI w, 1	
	JMP ZETA	
ETA	HLT	

Subroutine VSLR

Location	Instruction
VSLR	LDI b, 0
IOTA	SLR A
	INC B
	MOV b, c
	SUB c, z
	JMZ c, THETA
	JMP IOTA
THETA	RET

Subroutines VSLD, VSLU, VSLL are similar to VSLR

Subroutine EOR

Location	Instruction
EOR	MOV A, M
	MOV B, N
	NEG A
	NEG B
	MOV A, R0

TABLE 5 (continued)

Location	Instruction
	AND N
	MOV B, R0
	AND M
	MOV M, R0
	OR N
	MOV N, A
	RET

Note: All registers are assumed to be cleared before execution of this program.

Original image is in Register L.

* Image size = $2^9 \times 2^9$.

Modified forms of the counting algorithm. Basically, the tse program shown in Table 2 (page 55) is used to implement the modified forms of the algorithm. Of course, some adjustments to parts of the program are necessary. These adjustments are minor and are presented without listing the entire program.

In one of the modified forms of the algorithm, recall that not all of the right-slide iterations or the down-slide iterations are performed, but are truncated. To accomplish this, the immediate bytes of the LDI instruction at program location GAMMA+2 must be altered, since the value they contain will signal the end of each series of iterations. Before execution, the immediate bytes of the LDI should be loaded with the number 2^m, where m is the number of right-slide iterations to be executed. After all desired right-slide iterations have been performed, the immediate bytes of the LDI should be loaded with 2^n, where n is the number of down slides to be executed.

In another modified form of the algorithm, which concerns partial summing by sectors, the right-slide iterations and the down-slide iterations can be intermixed. To accomplish this, the desired sequence of iterations must be specified in some portion of memory and addressed by the main routine. The program is not changed up to location GAMMA. At this location, however, a branch should occur to address the portion of memory in which the iteration sequence is located. When the next desired iteration is identified, registers w, x, and y should be modified accordingly. These registers control the magnitude and direction of the slide operation for each iteration. At this point, a branch to location ZETA should occur. This will start the iteration. Upon completion of

the iteration, the processor will again be at location GAMMA. Thus, the iterations are repeated in this manner until the desired sequence is completed.

A total of 92 tse gates are required for the programmable tse computer implementation of the counting algorithm. The corresponding power requirement for this number of gates is 276 watts. Total processing time for a single image is 68,775 tse gate delays, representing an image processing time of 343.9 seconds per image. This corresponds to an image processing rate of 2.9×10^{-3} images per second. Image processing times are based on an average carry-propagation distance of three positions per image. Unlike the combinational and pipeline implementations, the processor can indicate early completion of the algorithm. Maximum carry propagation in each iteration is unlikely, especially where this maximum is eight or more.

CHAPTER 5

COMPARISONS AND CONCLUSION

The concept of two-dimensional logic devices and the tse computer represents a new and different approach to the task of image processing. Although the basic ideas used in the development of tse operations are not particularly innovative and were conceived long ago, the use and refinement of these ideas have been severely limited by the lack of a technology required to implement them. Only now can such a highly parallel logic structure even begin to be considered as having practical applications in the future. At the present time, the first tse gates have been conceived and are under design and development. The specifications related to these gates give an indication that, in the near future, tse computers may replace conventional computers in certain applications. In order to gain a better perspective as to the relative merits of tse processors, the characteristics as presented in the previous sections will be compared with the characteristics of a conventional processor.

Of course, the execution of the counting algorithm cannot be considered as a proving ground for tse processors. There are many tasks that can be performed on a tse computer which could allow the processor to compete effectively with conventional processors, in terms of efficiency. Each task would generate a different set of specifications for comparison, which in turn would generate a different set of conclusions. Therefore, the results of consideration of the counting algorithm for tse

implementation must not be taken to rigidly apply to tse computers in general. Any projection based on the results generated here must be carefully evaluated.

All comparisons presented in this section are based on an image size of 512 × 512 pixels. This represents an excellent resolution, almost that of a standard television receiver. At present, the 512 × 512 is projected as the upper limit on the image size of tse components.

A good evaluation of the usefulness of the tse computer must include a comparison to a typical conventional processor. For the purpose of comparison, the counting algorithm will be implemented in terms of the language of the IBM 360. The total processing time will then be determined using the actual instruction cycle times of the 360/65, Level H.

The IBM 360 is not by any means the fastest processor available; however, its characteristics are intended to be representative of most computers in general use today. Also, the results generated by this example can be easily extended to fit almost any conventional processor by determining actual times for tasks performed.

A program written for the 360 to count the number of 1-elements in an image is shown in Table 6. The basic approach is to address each picture element and increment a counter if the element is a logic- "1." An image is stored as 8192 consecutive 32-bit memory locations in the computer. The image could also be stored externally, possibly latched at the output of a parallel-type camera, and addressed in much the same manner as the internal memory would be addressed. For the basic

TABLE 6

PROGRAM FOR EXECUTION OF THE COUNTING ALGORITHM ON THE IBM 360

Location	Instruction		Comment
	SR	8, 8	Clear Register 8
	SR	4, 8	Clear Register 4
	LA	6, 1 (0, 4)	8192⟶ Register 6
	SLA	6, (13, 0)	
	SR	5, 5	Clear Register 5
A3	L	3, 0 (5, 2)	Load Register 3 with one word from memory
	LA	7, 32 (0, 0)	32⟶ Register 7
A2	CR	3, 8	Compare MSB of Register 3 to zero. If zero do not increment Register 4
	BH	A1	
	LA	4, 1 (0, 4)	
A1	SLL	3, 1 (0)	Shift left one position Register 3
	BCT	7, A2	Go to A2 unless 32 shifts have occurred
	LA	5, 1 (0, 5)	Increment pointer to address new word
	CR	5, 6	Compare new word address to 8192
	BL	A3	Go to A3 unless 8192 words have been checked
	END		

Register 2 should contain the absolute address of the first word of the image before this program is executed.

Register 4 will contain the result after the execution of the program.

counting algorithm, the program of Table 6 is very efficient. The execution time for this program ranges between 1.06 seconds for an all-zero tse and 1.15 seconds for an all-one tse, where the average time is 1.10 seconds.

The execution times for the three tse implementations of this research depends upon the propagation delay per gate. Initially, a delay of 5 milliseconds per gate has been specified for tse gates which are being developed [5, page 5]. Of course, future technological projections indicate improvement in the propagation delay. Eventually, the delay is expected to be comparable to the delays of present-day binary logic gates.

Table 7 summarizes the processing time per image for the 360, along with number of gate delays per image of each tse implementation and the corresponding execution times for different projected values of the tse gate delay. A comparison indicates that tse hardware structures behave in much the same manner as binary gate structures. For instance, (1) a combinational structure will be much faster for any given task than a sequential structure (computer) because of repetitive use of fewer components in the latter, and (2) a pipeline structure will improve the image processing rate (number of images per unit time) over that of a combinational structure. However, the actual delay for any image through the pipeline structure may be longer.

Another indication which results from the comparison is that, for the basic parallel counting algorithm, the image processing rates for the combinational and pipeline methods are on the same order of magnitude as that of the binary processor, but the programmable tse computer is

TABLE 7

SUMMARY OF IMAGE PROCESSING TIMES FOR TSE AND CONVENTIONAL IMPLEMENTATIONS

Implementation	Gate Delays per Image	Processing time per image tse gate delay: 5 ms	5 μs	5 ns
tse-Combinational	756	3.78 sec	3.78 ms	3.78 μs
tse-Pipeline	76	0.38 sec	0.38 ms	0.38 μs
tse-Computer	68,775	343.8 sec	0.344 sec	0.344 ms
IBM 360		1.10 seconds		

slower, for the present time. This is not unexpected since tse logic is fast compared to conventional logic (due to the inherent parallelism), yet slow compared to delays to present-day binary gates fabricated by TTL or CMOS technologies (about 10 nanoseconds). However, as the speed of the tse gate is improved, the processing rate of the tse computer will surpass that of the binary processor. As the tse gate propagation delay approaches that of today's binary gates, the advantage of the tse computer is apparent in that the number of 1-elements in a 512 × 512 image can be counted in about one-third of one millisecond.

Other characteristics which could be used to compare the merits of the different implementations are total power, total size and weight, dollar cost, speed-power product, and gate count. However, most of these are physical characteristics which may be refined independently of each other. Therefore, no projections can be made as to when a certain characteristic will be refined to the point that the tse computer is feasible for a certain application. Some of the more meaningful characteristics are summarized in Table 8. The only conclusion that can be drawn concerning these characteristics is that any tse processor that could be built using current fiber optics technology would probably not be a practical replacement for the binary processor. Currently, alternatives to the power-consuming light sources and bulky optical fibers are being considered for the fabrication of tse logic components.

The need for improved data-handling capabilities in digital computers will inevitably lead to research into increased parallelism. The results of this section indicate that tse computers will not replace conventional processors presently, although they have a definite

TABLE 8

PHYSICAL CHARACTERISTICS OF TSE IMPLEMENTATIONS
OF THE COUNTING ALGORITHM

Implementation	Power Consumption	Speed-Power Product @ 5ms/gate	Gate Count
tse-Combinational	1080 W	4082.4 W-sec	360
tse-Pipeline	1404 W	533.5 W-sec	468
tse-Computer	276 W	94,888 W-sec	92

potential for future use. From the standpoint of the tse computer in orbit as an earth resources image processor, many of its characteristics are very promising. However, the size and power consumption of the processor must be brought to within limitations for practical spacecraft. The development of the tse computer will, of course, depend primarily upon the amount of research effort devoted to the concept in the near future. Until then, the expansion of today's digital computer will more than likely take the form of increased parallelism of conventional logic components. At some time in the future, however, the physical characteristics of tse devices will be refined to the point where they are more attractive than are the increasing number of inter-connections required for conventional gates, thus marking the advent of the generation of tse computers in digital processing.

LIST OF REFERENCES

LIST OF REFERENCES

1. Millman, J., and H. Taub, Pulse, Digital and Switching Waveforms. New York: McGraw-Hill, 1965.

2. Unger, S. H., "A Computer Oriented Toward Spatial Problems," PROC. IRE, vol. 46, October 1958, pp. 1744-1750.

3. Slotnick, D. L., W. C. Borch, and R. C. McReynolds, "The Solomon Computer," AFIPS PROC. FJCC, vol. 22, 1962, pp. 97-107.

4. Barnes, G. H., et al., "The ILLIAC IV Computer," IEEE TRAN. on Computers, vol. C17, August 1968, pp. 746-757.

5. Schaefer, D. H., and J. P. Strong, III, "tse Computers," X-943-75-14, NASA, GSFC, January 1975.

6. Schaefer, D. H., and J. P. Strong, III, "Two-Dimensional Radiant Energy Array Computers and Computing Devices," Patent Application Serial No. 468614, May 8, 1974.

VITA

Alan Gerald Metcalfe was born in Clarksville, Tennessee, on March 28, 1951. He graduated from Clarksville High School in 1969, at which time he entered the pre-Engineering curriculum at Austin Peay State University. In 1974, he received the Bachelor of Science degree in Electrical Engineering from The University of Tennessee. At this time, he entered The University of Tennessee Graduate School and earned the Master of Science degree in Electrical Engineering in December 1976.

In September 1975, he was employed as an Electrical Engineer with the Tennessee Valley Authority.