Machine Vision and Applications (1990) 3:21–44

Machine Vision and Applications

VLSI Design and Implementation of a Real-Time Image Segmentation Processor*

Bir Bhanu
Honeywell Systems & Research Center, 3660 Technology Drive, Minneapolis, Minnesota, USA

Brad L. Hutchings and Kent F. Smith
Department of Computer Science, University of Utah, Salt Lake City, Utah, USA

Abstract: Image segmentation is a crucial part of machine vision applications. In this paper a system to perform real-time segmentation of images is presented. It uses a real-time segmentation VLSI chip that is based on a gradient relaxation algorithm and is designed using the Path Programmable Logic design methodology developed at the University of Utah. The system design considerations, system specifications, and an input/output format for the chip are discussed. The actual design of the chip is given that uses pipeline methodology to achieve real-time performance with a compact VLSI layout. The implementation of the segmentation system is presented and the segmentation chip and the overall system are evaluated with regard to real-time performance and segmentation results.

Key Words: computer aided design, image segmentation, Path Programmable Logic, real-time image processing, relaxation, VLSI

1 Introduction

The low-level image processing required by image analysis algorithms is computationally intensive. Images of size 512 × 512 bytes are commonly used in general image processing and larger image sizes are often required by LANDSAT, aerial, and biomedical imaging applications (Barbe 1981, Kruger and Thompson 1981). Military applications such as target recognition and tracking (Bhanu 1986) and commercial applications involving mobile robots will not be able to utilize fully image analysis techniques unless the system can provide information at the rates necessary to allow the image processor to respond in real time to visual stimuli. The large amounts of data coupled with the need for rapid processing make general purpose computers inappropriate for most image processing tasks. For a 512 × 512 image data rates of 30 frames per second (fps) are common. To achieve real-time performance with these data rates, the computer would have to process fully a picture element (pixel) every 100 nanoseconds (ns). Since most image processing algorithms require multiplication as well as other simpler operations to be performed on each pixel, specialized computer architectures are required if real-time performance is to be realized. Previously proposed special architectures have included array processors and multiprocessors. However, array processors and multiprocessors cannot meet the needs of many practical image processing systems due to size, weight, power, and environmental requirements.

The design and implementation of image processing algorithms in VLSI (very large scale integration) is an expanding area of research (Fouse et al. 1981, Fu 1984, Nudd et al. 1979, Offen 1985). Special purpose charge-coupled devices (CCDs) and MOS ICs (metal oxide semiconductor integrated circuits) have been custom designed for low-level operators on "smart" sensor projects. The impact and advantages of VLSI on image understanding have also been studied (Nudd et al. 1979). A recent example of a VLSI chip for image processing is the real-time video moment generating chip developed by Anderson (1985, Weste and Eshraghian 1985). The main obstacle blocking the wide-

Address reprint requests to: Bir Bhanu, Honeywell Systems & Research Center, 3660 Technology Drive, Minneapolis, MN 55418, USA.

* This work was supported in part by Grant ISI-856-0393 from the National Science Foundation.

spread use of VLSI in designing real-time image processors is the complexity of VLSI design. VLSI design complexities typically increase exponentially with the number of devices on the chip, making VLSI very costly to implement (Smith, Carter, and Hunt 1982). However, computer aided design (CAD) approaches to VLSI design help to manage this inherent complexity and can reduce design times by an order of magnitude when compared to full-custom design techniques. Since the design complexities required by real-time image processors also increase significantly with the number of functions implemented, CAD-based VLSI design methodology provides us with an excellent tool for the design of specialized real-time image processors. Further, these VLSI CAD tools can be used to update and make changes in the designs and to implement effective testing procedures.

This paper describes the development of a segmentation processor that uses computer aided VLSI design tools to produce a CMOS (complementary MOS) VLSI chip that segments TV Images in real time. The paper is organized into six sections: Section 1 is the introduction; Section 2 presents the CAD-based design methodology; Section 3 discusses algorithm selection in the context of VLSI design; Section 4 presents the system-level design considerations and details of the system; Section 5 describes the actual design of the chip; Section 6 discusses the issues related with system construction and integration; Section 7 discusses the evaluation of the real-time image segmentation chip and the evaluation of the image processing system; and finally, Section 8 summarizes the results and contributions of this paper.

2 CAD-Based VLSI Design Tools

CAD-based approaches to VLSI design help to manage the circuit complexity by hiding many of the circuit details from the designer. In a sense, using these CAD tools is analogous to using high-level computer languages such as Pascal or "C" instead of assembly languages. Generally, these high-level languages are not so efficient or so fast as hand-coded assembly language but are much easier to use and understand when writing complex programs. Similarly, CAD VLSI tools provide a framework for designing complex circuits that allows the designer to concentrate on functionality while leaving such details as process design rules and mask generation to the computer. Circuits designed using these methods are not so compact as full-custom designs but are easier to understand and have a better chance of working on the first pass because of the relative complexity reduction. The University of Utah has developed a CAD-based VLSI design tool known as Path Programmable Logic (PPL), which provides these improvements in circuit design efficiency (Smith 1983, Smith, Carter, and Hunt 1982, Smith and Israelson 1985). In this section the PPL design methodology is presented and contrasted against other CAD-based VLSI design tools.

2.1 The Path Programmable Logic Design Methodology

PPL is a cell-based design tool. Similar to the standard cell design method, the designer chooses the functions necessary to implement the design from a "cell library" of previously designed circuit modules. Cells are placed in the PPL grid and connected to other cells by butting cells together or routing wires directly between cells. One major advantage provided by PPL over standard cells is its inherent two-dimensionality. PPL cells can be placed at any row or column location. The cells are designed to fit together on all four sides rather than on two sides as in the standard family design. All four sides of each PPL cell have interconnecting wires running to the edges so interconnection between cells can be made in all four directions simply by placing a cell next to a neighboring set of cells. PPL cells can occupy a multiple of row and column locations. For instance, some cells, called "unit" cells, will only occupy a single row and column location, whereas other cells, called "multiple" cells, may occupy several locations. A flip-flop cell might occupy two columns and four rows but an inverter might only occupy a single column and two rows.

PPL cell layout is best visualized by assuming that we have horizontal and vertical metal interconnecting wires that cover the entire chip. The horizontal and vertical wires are grouped together so that each row and column may be represented as having N horizontal wires and M vertical wires. CMOS cells require two horizontal wires and three vertical wires in each row/column location. Power and ground wires run on the bottom level and interconnect to all grid locations, but the presence of these power supply wires are invisible to the user and are not included in the N by M wires. At each of the row/column intersections a unit or a multiple cell may be placed and the horizontal and vertical wires in that row/column may either connect to the cell, be broken, or simply pass the wires through the cell.

The main idea behind the PPL methodology is that circuits may be constructed by placing cells at these grid locations. The interconnect between cells

is formed by simply placing the cells next to each other and allowing connections to be made. Disconnections between cells can be made by forcing breaks in the existing interconnecting wires. Note that this technique is different from the standard cell approach where cells are positioned at predefined one-dimensional locations and interconnecting wires are placed between cells.

At this point the comments on breadboarding are worth mentioning. Breadboarding refers to the practice of building an operational prototype of the VLSI circuits using conventional discrete components. This step is used to validate further the design before committing to VLSI. Breadboarding hopefully avoids some of the pitfalls and bugs that may have been missed in the early designs and eliminates the time-consuming step of redesign at the full-custom VLSI level. However, prototypes built using discrete components cannot exactly duplicate the operation of VLSI circuitry and require a significant amount of time to implement. Further, it contributes little to the actual low-level design of the VLSI components. With the simulation tools that are available with VLSI design systems and the reduced manpower required when designing with PPL, the time-consuming breadboarding step is no longer necessary. In the time it takes to design and prototype a breadboard version of the chips the actual chips can be designed and sent for fabrication.

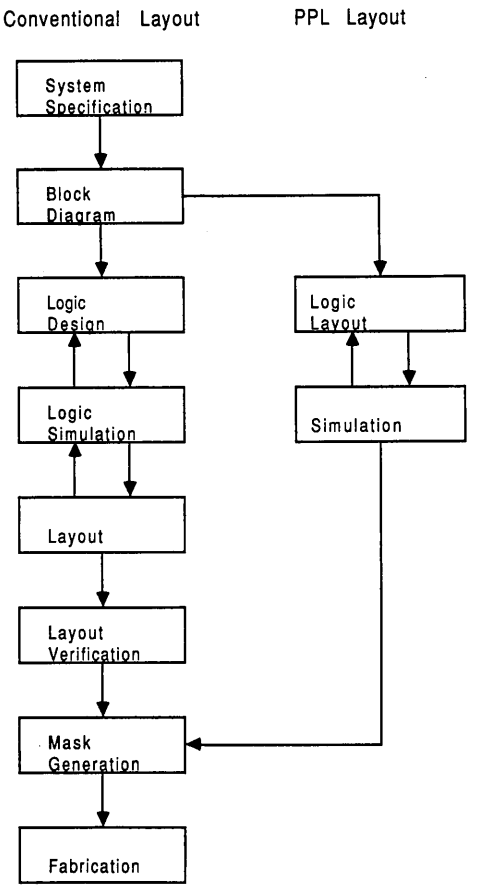

Figure 1. A pictorial comparison of PPL and conventional VLSI design techniques.

2.2 PPL Computer Aided Engineering Tools

Layout (design) of an integrated circuit using PPL methodology is done on a conventional alphanumeric terminal. The user places symbols that represent specific cells in the PPL library onto an array within a window representing an area on the silicon. Editing consists primarily of moving the cursor to certain locations and inserting a PPL cell at this location by typing a character corresponding to the cell. All standard editing features are available, such as being able to read and write to other files, movement of blocks of PPL programs, and so on. This editing (design) process represents the logical description of the circuit, the physical layout, and the interconnect. The layout editor understands the cell library being used by the designer and thus prevents him or her from creating illegal circuit configurations such as overlapping cells. Placement restrictions imposed by the IC technology being used are also enforced by the layout program. As a result, electrically and topologically "correct" circuits are produced by the layout editor. This means that the power and ground busing structure is complete and that all cells properly interface on the PPL grid as described in the previous section. After this first design iteration the only verification task required for PPL circuits is logic simulation and timing analysis (Smith 1983). Figure 1 shows a pictorial comparison between PPL and other more conventional VLSI layout techniques (Mead and Conway 1980, Weste and Eshraghian 1985). As can be seen from the figure, PPL not only reduces the number of steps required to design a circuit, but also it tightly couples the design sequence so that immediate design feedback is available.

Figure 2 shows an example of the PPL design of a full adder. The top portion of the figure is the schematic diagram of a full adder that consists of two half adders and and "or" gate. The PPL representation of the full adder is shown directly below this schematic diagram. The PPL representation can be directly mapped to its corresponding parts in the schematic diagram. For clarity, the PPL representation has been blocked out into sections. Each section is labeled according to the schematic function implemented by that PPL block. For example, the "*x*or" gate is represented by the box with the

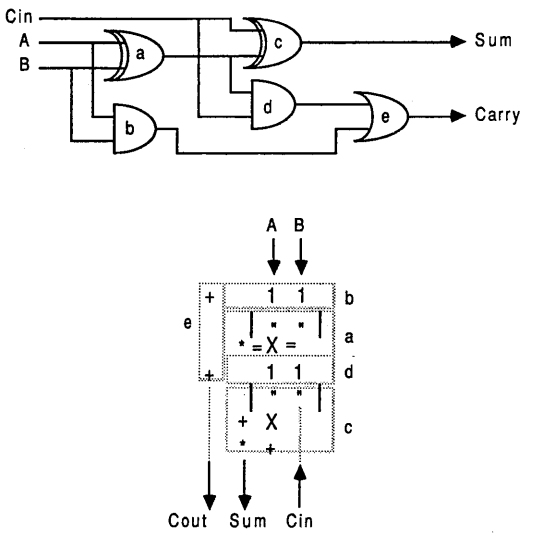

Figure 2. A full adder and the corresponding PPL representation.

large "*X*." Accordingly, other gates are noted and labeled. The inputs marked "*A*" and "*B*" are carried into the interior of the "*x*or" gate (*a*) and "and" the gate (*b*) on the two available column wires. The bottom "*x*or" gate is routed similarly with "C_{in}" coming in on a column wire and the output from the "*x*or" gate performing the "half-add" of "*A*" and "*B*" being routed through two column wires and one row wire. The "C_{out}" output (*e*) is created by "or"ing the outputs from the two "and" gates (*b*, *d*) through the "plus" cells. As shown from this example, the PPL structure allows cells to be placed so that the amount of required interconnect can be minimized, thereby allowing "macro" cells (cell modules created from other cells) to be designed that can be butted together to form automatically the interconnect between cells. By exploiting the two-dimensionality of PPL, the designer can reduce the length of interconnect wiring that in turn provides the potential for the circuit to operate at a higher frequency.

The PPL program made with the PPL editor contains all the information about both the logic and the physical topology of the entire circuit. Thus, it is possible to do performance analysis of the integrated circuit as the PPL program is defined. This means that, as opposed to conventional design techniques, PPL can give instant feedback to the logic designer about the performance of the particular design. Simulation of a PPL circuit is performed in two passes. First, the symbolic design done at the design step is converted to a logical description of the circuit, and a transistor-level net list is created that includes all active devices as well as all stray capacitance and resistance values. The circuit is also checked for such things as DC circuit problems, unknown paths, and syntax errors. Second, functional simulation is performed by specifying inputs to the circuit and computing the resulting outputs and internal node changes.

2.3 Mask Generation and Fabrication

Given the PPL design as created by the layout editor, mask data is generated in the CIF format (Mead and Conway 1980, Weste and Eshraghian 1985). These masks can then be converted to any of a number of other formats such as GDS-II, Computer Vision's EXDB, and Hewlett-Packard's IGS. A file in one of these formats is then delivered to a vendor who will make a mask set that is specifically keyed to the foundry to be used.

The PPL design specifies the use of a library of primitive cells that appear to be generic in nature to the design engineer. However, the specific fabrication technology that is to be used must be specified so that the resulting design is keyed to a cell library for that technology. To date, cell libraries have been created for technologies supported by the Department of Defense MOSIS (MOS implementation system) facility. MOSIS is a silicon broker that contracts with outside vendors for foundry services. It was set up to provide an inexpensive prototype service using the multiproject chip concept for DOD-supported research, and it has recently been opened to commercial contractors.

The CMOS cell set used in this project supports the MOSIS-scalable CMOS technology that is compatible with many existing silicon foundries. This cell set contains over 50 cells and is the result of the past three years of research. The library is compatible with P-well, N-well, or twin-tub CMOS processes. It was designed using a set of 3 micron design rules that can be scaled down to 1.2 micron processing technology. It implements two-level NAND gate logic and utilizes a single-phase clock for the storage elements. In addition to the basic PPL cells, a complete sublibrary of cells has been developed for the design of static RAM subsystems.

2.4 Benchmark Tests

The design of ICs using the PPL methodology is a major departure from the more traditional ways of doing design such as standard family or full custom. Thus, it was important to make a comparsion between circuits designed using the PPL methodology and more conventional approaches. A benchmark test (Smith and Israelson 1985) comparing custom design and PPL design of two circuits using an NMOS process was done in cooperation with a lo-

cal company. The results of the benchmark test showed that the sizes of PPL-designed circuits compared very favorably with full custom. PPL circuit sizes ranged from 38 percent larger than full custom to 5 percent smaller than full custom. More importantly, PPL design times were an order of magnitude less than the time required for full-custom design. Primarily because of these factors, PPL VLSI design methodology was used in the design of the image segmentation chip.

3 Algorithm Selection for Segmentation

The principal goal of the research project described in this paper is the VLSI design and implementation of an image segmentation algorithm. Image segmentation is one of the first steps in the image analysis process and refers to the grouping of parts of an image that are homogeneous with respect to one or more image characteristics. The results obtained from the segmentation step are very important since they form the beginning for the interpretation of the image. As such, the segmented image provides the input for further image analysis algorithms. Also, the maximum execution speed of the image segmentation algorithm sets the lower bound on execution time for the overall image analysis process. Any image analysis system striving toward real-time performance will require a segmentation subsystem that operates in real time. Due to the large amounts of data that must be processed in order to segment an image, segmentation algorithms, in general, require special purpose hardware for real-time performance.

3.1 Relaxation

Relaxation is a general computational technique that is very useful in computer vision for applications such as image segmentation and scene labeling (Bhanu and Faugeras 1982, Rosenfeld, Hummel, and Zucher 1976). When used as a method for image segmentation, the relaxation algorithms assign pixels into classes based on their gray values, and the gray values of the neighboring pixels. Relaxation algorithms are iterative in nature; each new iteration of the algorithm uses the results from the previous iteration so that the process is more informed as it proceeds. Each iteration of the algorithm proceeds incrementally until the algorithm converges to the final set of pixel label assignments that assign the pixels to regions. The iterative nature of relaxation algorithms gives the segmentation process a smoothing property that can provide excellent performance when using noisy images. An important question associated with relaxation algorithms is the number of iterations needed for the convergence of the process (Rosenfeld, Hummel, and Zucker 1976). In our gradient relaxation algorithm (Bhanu and Faugeras 1982) we use a well-defined mathematical criteria and provide parameters to control the rate of convergence. In practice, we use only a small fixed number of iterations based on an application. In the following we examine the appropriateness of relaxation algorithm for real-time image segmentation.

Relaxation algorithms fall into the class of parallel-iterative algorithms. Each iteration of the relaxation process can be executed in parallel due to the local execution of the algorithm. Local execution refers to the fact that the algorithm performs only local data accesses; the updating of any single value can be performed with only a small percentage of the overall data that, for image processing purposes, is usually spatially localized. For relaxation algorithms that perform image segmentation updating a single pixel value normally requires only the values of the eight neighbors immediately surrounding the pixel. Therefore, if a fully parallel implementation of the algorithm were available, all image pixels could be updated simultaneously. In many machine vision applications image acquisition devices such as TV cameras and scanners only allow access to an image in serial fashion (one line at a time) due to the rastering process that is used to create the image. Thus, practical systems that process image data in real time will gain little or no advantage from a fully parallel approach due to the serial nature of the imaging device. However, such systems can still exploit the localized execution of relaxation algorithms to great advantage. Since the pixel update depends only on its eight neighbors, it is not necessary to wait for the first iteration to finish before starting the next iteration. The next iteration can begin as soon as enough pixels have been processed to support a full neighborhood for the next iteration. The computations for each iteration can then overlap to a great degree, providing a significant speedup of algorithm execution. In contrast to a single relaxation iteration of the entire image being processed in parallel, the best approach for systems that use serial imaging devices is to process multiple iterations of a small portion of the image data in parallel.

The segmentation technique chosen for this project is the gradient relaxation algorithm developed by Bhanu and Faugeras (1982). This algorithm will be presented here so that it can be compared to other segmentation techniques based on its effectiveness as a segmentation algorithm and its suitability for VLSI implementation. Earlier Willet

(1978) has reported the use of bit-slice microprocessors to implement a nonlinear relaxation algorithm by Rosenfeld, Hummel, and Zucker (1976).

3.2 Gradient Relaxation Algorithm

Various approaches based on thresholding have been used by many researchers for the segmentation of both monochrome and color images. Normally, in the application of these techniques the histogram shows two or more peaks in at least one of the spectral features corresponding to various homogeneous regions of an image. Very often preprocessing is done to improve the histogram, and local properties are used to compute the local, global, or dynamic thresholds. However, if the intensity (or color) histogram of the image is unimodal, then the application of such methods gives a poor segmentation. Further, there are no criteria for automatic threshold selection. Unimodal histograms are typically obtained when the image consists mostly of a large background area with other small, but significant regions. This is true for most aerial, biomedical, and industrial images. Bhanu and Faugeras (1982) have presented a basic two-class gradient relaxation technique and have used its variations successfully for the segmentation of outdoor scenes, images of biological cells, and military targets (Bhanu 1986, Bhanu and Holben 1990, Bhanu and Parvin 1987). It controls the relaxation process and provides automatic selection of the threshold. A brief description of the basic two-class technique follows:

Suppose a set of N pixels $i = 1, 2, \ldots, N$ fall into two classes λ_1 and λ_2 corresponding to the white (gray value = 225) and black (gray value = 0) classes. Reduced inconsistency and ambiguity of pixels with respect to their neighbors are achieved by maximizing the global criterion (Bhanu and Faugeras 1982, 1984):

$$C(\mathbf{p}_1 \cdots \mathbf{p}_N) = \sum_{i=1}^{N} \mathbf{p}_i \cdot \mathbf{q}_i \tag{1}$$

subject to the constraint that $\mathbf{p}_i$'s are probability vectors. $\mathbf{p}_i$ is the probability that the ith pixel belongs to class λ_1 and λ_2. The compatibility vector, $\mathbf{q}_i$, is a function of the $\mathbf{p}_i$'s and is defined as

$$\mathbf{q}_i(\lambda_k) = \frac{1}{V_i} \sum_{j \in V_i} \sum_{l=1}^{2} c(i, \lambda_k, j, \lambda_l) p_j(\lambda_l)$$
$$k = 1, 2; \; i = 1, \ldots, N \tag{2}$$

where compatibility

$$c(i, \lambda_k, j, \lambda_l) = \begin{cases} 0 & \text{if } k \neq l;\ k = 1, 2;\ j \in V_i; \text{ for all } i \\ 1 & \text{if } k = l;\ k = 1, 2;\ j \in V_i; \text{ for all } i \end{cases} \tag{3}$$

and V_i is the size of the set of (eight) nearest neighbors. $\mathbf{q}_i(\lambda_k)$ is the average of $\mathbf{p}_j(\lambda_k)$ of the eight nearest neighbors. Compatibility c measures the likeness of a pixel with its neighbors and allows the reinforcement of pixel labels of similar classes.

The maximization of the global criterion (1) means that we are seeking a local maximun close to the initial labeling (see equation (4)) subject to the constraints that the $\mathbf{p}_i$'s are the probability vectors. The maximization of criterion (1) results in reduced inconsistency and ambiguity. Inconsistency is defined as the error between $\mathbf{p}_i$ and $\mathbf{q}_i$. Intuitively, this means the discrepancy between what every pixel "thinks" about its own labeling ($\mathbf{p}_i$) and what its neighbors "think" about it ($\mathbf{q}_i$). Ambiguity is measured by the quadratic entropy and results from the fact that initial labeling $\mathbf{p}_i^{(0)}$ is ambiguous ($\mathbf{p}_i^{(0)}$ are not unit vectors). We are therefore trying to align the vectors $\mathbf{p}_i$ and $\mathbf{q}_i$ while turning them into unit vectors. It can be easily seen that each term $\mathbf{p}_i \cdot \mathbf{q}_i$ is maximum for $\mathbf{p}_i = \mathbf{q}_i$ (maximum consistency) and $\mathbf{p}_i = \mathbf{q}_i$ = unit vector (maximum unambiguity).

Initially, at every pixel, the assignment of probabilities is done by

$$p_i(\lambda_1) = FACT \frac{I(i) - IBAR}{255} + 0.5 \tag{4}$$

where, $I(i)$ is the gray value at the ith pixel and $IBAR$ is related to the mean and variance of the image (Bhanu and Faugeras 1982, Bhanu and Parvin 1987). $FACT$ is a function of intensity that is taken to be equal to 1 if $I(i) > IBAR$, otherwise its value is between 0.5 and 1. $FACT$ is related to the expected number of white and black pixels in the image. It does not affect the rate of convergence very much, but it affects the segmentation results.

A projection gradient technique is used to solve the problem as stated in equation (1). The gradients of the criterion C in (1) with respect to λ_1 and λ_2 are $2q_i(\lambda_1)$ and $2q_i(\lambda_2)$, respectively. By computing the projection of this gradient and simplifying the equations for quick convergence, the approximate iterative equations for the relaxation process are obtained (Bhanu and Faugeras 1982). These are

$$p_i^{n+1}(\lambda_1) = p_i^n(\lambda_1)[1 - \alpha_1] + \alpha_1; \; q_i(\lambda_1) > 0.5;$$
$$0 < \alpha_1 < 1.0 \tag{5}$$

$$p_i^{n+1}(\lambda_1) = p_i^n(\lambda_1)[1 - \alpha_2]; \; q_i(\lambda_1) < 0.5;$$
$$0 < \alpha_2 < 1.0 \tag{6}$$

The magnitudes of α_1 and α_2 control the degree of smoothing at each iteration and their ratio controls the bias toward a class. The magnitude of *FACT* controls the initial assignment of probabilities. Since we use the projection gradient method to obtain iterative equations, we named this method the "gradient relaxation method" to distinguish it from the nonlinear relaxation method of Rosenfeld, Hummel, and Zucker (1976).

A few iterations of equations (5) and (6) result in the segmentation of an image by allowing the automatic selection of thresholds. The technique has been evaluated with respect to signal-to-noise ratio, region size, and contrast of the objects present in the image (Bhanu and Parvin 1987). It has been found to be very effective for the segmentation of low contrast images and the segmentation of natural scenes. Experiments with the parameter values and the quality of the segmentation have shown that the α's and *FACT* should be kept constant for each particular set of images belonging to a specific application. In section 7.2 we present some examples of segmentation to show the effects of parameter values.

3.3 Comparison of Segmentation Approaches

The gradient relaxation algorithm (Bhanu and Faugeras 1982) used in this research has been compared to another relaxation algorithm—the Rosenfeld, Hummel, and Zucker (1976) algorithm. The comparison assumed the same compatibility function and initial probability assignment as the gradient relaxation algorithm. While both algorithms provide a two-class image segmentation approach, the gradient relaxation algorithm has the added benefit of control over the segmentation process. The parameters *FACT*, α_1, and α_2 provide control over the segmentation process by allowing the user to adjust the convergence rate, class bias, and the distribution of the initial probability assignment. This advantage gives the algorithm the additional flexibility necessary for use in a wide range of applications.

There are two possible ways to extend the algorithm described in Section 3.2 to perform multiclass segmentation of natural scenes. In the first case we simply extend the technique to a known number of classes. However, such extension is not desirable since in practice we may not know the number of classes and also because of the complexity associated with the algorithm. In the second case we apply the two-class gradient algorithm recursively to segment the image into multiple classes. We have extended the gradient relaxation algorithm to perform multiclass segmentation of natural scenes (Bhanu and Parvin 1987). The extended algorithm proceeds by applying the two-class segmentation algorithm to the entire image, thus splitting the image into two classes based on both local and global characteristics. The connected components of the segmented image are isolated and each connected component is used as a binary mask on the original image for further partitioning. This process continues recursively until a region can no longer be partitioned or is of such a small size that it is no longer of interest. This extension provides significatn benefits for hardware implementation. Thus, the gradient relaxation algorithm can be used in its two-class form when applicable, and can be extended to multiclass when needed.

The multiclass algorithm has also been compared to two other well-known segmentation techniques—the Ohlander, Price, and Reddy (1978) algorithm and the Nagin, Hanson, and Riseman (1982) algorithm. In both comparisons it produced segmentations on natural scenes that are as good as these other techniques. Furthermore, because of its simplicity, it produced these segmentations at a much lower computational cost. It avoids the arbitrary division of a picture into quadrants or subimages that leads to unwanted boundary effects (Nagin, Hanson, and Riseman 1982, Ohlander, Price, and Reddy 1978). We have also extended the two-class gradient relaxation algorithm to include edge information for more robust segmentation of military targets (Bhanu and Holben 1990). Since the two-class gradient algorithm has been found useful for a variety of images, we selected it for implementation when intensity information is used to segment an image.

3.4 Relaxation and Pipeline Design Methodology

The relaxation algorithm supports overlapped execution of the computation of each iteration. This ability allows the algorithm to be implemented quite effectively using pipeline design methodology. Each algorithm iteration can be treated as a pipeline segment; the individual iterations are connected together with the output of each iteration immediately providing the input for the next iteration. The segmentation processor constructed in this project processes the image pixels in a line-oriented (raster) fashion and the succeeding iteration can begin processing as soon as the preceding iteration has finished processing the first three lines of the image. Since the iterations are all computed in overlap fashion, a real-time implementation of the algorithm with any number of iterations is feasible. In fact, the number of algorithm iterations does not affect the real-time performance of the system. The system

will always be capable of processing information at the same rate as it is presented. However, the amount of time required initially to fill and traverse the pipeline increases with the number of iterations, thereby increasing the system latency.

A pipelined implementation of the algorithm also provides benefits for the VLSI implementation of the algorithm. Each iteration of the algorithm will require its own identical processing hardware so that the computations for each iteration can be fully overlapped. A system that fully overlaps the execution of five algorithm iterations will require five identical processors, one for each iteration. VLSI design methodology is best used where the initial design and implementation results can be used multiple times within any system. The initial engineering costs of a VLSI implementation are quite high and the fabrication of the actual chips is most economical if done in large quantities. Thus, the implementation of the gradient relaxation algorithm matches well with the requirements for an effective VLSI implementation. For this project a single chip was implemented that performs the iterative portion of the relaxation algorithm. These chips were then cascaded to execute as many algorithm iterations as desired. An additional advantage of this approach is that the same chip can be used without modification in a multiclass segmentation system (Bhanu and Parvin 1987).

4 Design Considerations and Image Segmentation System

The design and implementation of a real-time image segmentation system requires making trade-offs among the various design considerations. In the following discussion, we examine these considerations at the system and chip level and provide system-level details to specify portions of the algorithm that are chosen for implementation in VLSI and those that are implemented with discrete components. The input and output formats for the chip are also presented.

4.1 Design Considerations

4.1.1 Silicon utilization. One of the first design considerations in a VLSI design is the amount of chip area available for actual implementation purposes. This consideration becomes even more important when a CAD-based VLSI tool is being used for the design. As previously discussed, PPL CAD-based VLSI deisgn tools are aimed at managing the complexity issues of a VLSI design so that the design can be achieved in one-tenth of the time required for a full-custom design. This complexity management is achieved at the cost of some efficiency in the utilization of the silicon. This situation introduces somewhat of a dilemma since faster algorithm execution speed is usually achieved at the cost of silicon area. The actual design of this chip involved making an informed and careful trade-off between chip area and the execution speed of the algorithm. This trade-off was arrived at by carefully analyzing possible implementations of the algorithm to see if they can meet both the real-time performance requirements and the silicon chip area requirement. The amount of area available for this project in PPL terms was 250 rows by 95 columns.

4.1.2 Real-time operation. Another principal design constraint was the real-time operation requirement. Since the real-time operation of the segmentation algorithm was central to this project, it is helpful to define precisely what is meant by "real time." In general, a process is said to be operating in real time if it processes data at the same rate as it is presented. For example, if a process was processing an image in real time and the data were presented at 30 frames per second (fps), the process would fully process one frame each 1/30 of a second. Specifically, the goal of this project was to segment images acquired from a video digitizer that is connected to a TV camera at real-time rates. The image size is 512 (horizontal) $\times$ 240 (vertical) bytes and the frame rate for the RS-170 standard is 30 fps. A 512 $\times$ 240 image will require that each pixel of the image be processed in 100 ns in order to meet the frame rate requirement. This sets the global clock rate for the system at approximately 10.0 MHz.

4.1.3 Algorithm speedup methodology. Pipelining and parallelism are two alternate methodologies for VLSI implementation of the gradient relaxation algorithm. A fully parallel implementation of the algorithm is realizable and could provide the performance necessary for real-time execution; however, this would require a separate processing element for each pixel. For a 512 $\times$ 240 image this would require 122,880 processors and would be very expensive. Also, since this project depended on MOSIS for the fabrication of the CMOS devices, only a limited number of chips was available for the final implementation of the system.

Pipelining methodology provides a definite complexity advantage over fully parallel systems for the gradient relaxation algorithm. A real-time implementation of the algorithm could be constructed with five to ten VLSI chips and a moderate amount of discrete logic. In the pipelined approach a single

chip would be responsible for the computation of the compatibility measure (equation (2)) and the pixel probability update [equations (5) and (6)]. The system would then be constructed in pipeline fashion with each chip acting as a single pipeline segment and connected to other chips so that the output from a preceding iteration provided the input for the next succeeding iteration. The number of chips would be equal to the number of iterations desired and any number of iterations could be handled in real time. Each iteration would add to the overall system latency but would still provide real-time performance due to the ability to overlap iteration execution. Due to the serial processing of each pixel in the pipeline, two horizontal video scan lines must be fully processed before the succeeding iteration can begin processing. This requirement is due to the 3×3 neighborhood that is used in the compatibility measure shown in equation (2). Thus, the minimal latency that will be incurred at each iteration is two horizontal scan line times, each of which is approximately 65.3 μsec or about 650 clock cycles for a total of about 1300 clock cycles (assuming a 10 MHz system clock).

4.1.4 Chip latency. Up to this point latency has been discussed with regard to the overall pipelined architecture of the algorithm. The system latency of 1300 clock cycles per iteration will be minimally required for pipelined algorithm execution regardless of the final VLSI chip implementation. However, pipelining is also effective at the chip level to provide the throughput necessary for real-time performance with a minimal amount of hardware. If pipelining is used at the chip level, then some additional latency will be introduced at each iteration. The total amount of latency per iteration would then be the minimal two horizontal lines plus the number of clock cycles required for each chip to process fully a single pixel.

The amount of latency that the chip requires to process a single pixel should be minimized. Designs with high amounts of latency generally occupy correspondingly high amounts of silicon area. This dependence occurs because each clock cycle of latency will require some number of registers to store the intermediate result. Rule-of-thumb estimates for this design place the amount of chip area per cycle of latency at about 2 percent. These estimates are arrived at by considering past experience with PPL, the size of the register used to store the result, and routing concerns. A practical upper limit on the chip latency is approximately 10 clock cycles. That way, no more than 20 percent of the overall chip area could be consumed by the registers used in the pipelining process. The amount of latency contributed by the chip to the overall image frame latency per iteration is not significant, being less than 1 percent of the 1300 clock cycles required for each iteration of the algorithm.

4.1.5 Discrete logic. Systems using VLSI chip as a principal component require some amount of discrete logic. This discrete logic performs necessary interfacing functions and is used where VLSI implementation would be difficult or ineffective. Discrete components for this project include ICs from the TTL family and random access memory (RAM). During the design of the VLSI component it is important to select those functions whose VLSI implementation will have the greatest impact on algorithm performance. Discrete components are used to fill the logic gaps that are not implemented in VLSI due to lack of available silicon space or other factors. VLSI designs are difficult to modify; those functions that are required to be flexible or application dependent should be implemented using discrete logic rather than VLSI. Part of the design process, therefore, includes the selection of those functions that are most important to the performance of the algorithm. The remaining functions are then implemented using discrete logic to complete the system construction.

4.1.6 Pin out requirements. Any VLSI design has only a limited number of pins available for electrical connections. The packages used to provide connections to the actual VLSI silicon chip come with 24, 40, 64, and 84 pins. Therefore, the chip design can have no more than 84 pins. This is an important consideration of the design since it places a limit on the amount of data that can be supplied during a single clock cycle.

4.1.7 Limitations of PPL design tools. The COMOS PPL tool set is relatively new and does not have the large number of cells that are available in other NMOS cell sets. In fact, this project is one of the first major designs completed using the CMOS cell set. As such, the functionality of the cells in the cell set is somewhat limited. This is not a terrible hardship since more complex functions can be composed using the available basic cells. However, the composite functions built using PPL cells will occupy more silicon than if the function had already been implemented directly as a separate cell using a full-custom design for that function. The implication of this is that the choice of the algorithm implementation is very critical due to the general lack of silicon area and the size of the implementation.

Hence, it will be difficult or impossible to use some innovative techniques that will require more silicon area than other simpler, straightforward approaches when using the CMOS cell set as supplied.

4.1.8 Input data format. Digital images are represented as a serial stream of 8-bit unsigned numbers. This digital format is formed by passing the signals received from a TV camera through a video digitizer. The RS-170 video format is the most commonly used video format and was used in this project so that all components could be easily integrated. The visible portion of each horizontal line in a 512 × 240 image is about 51 μsec making the sample rate required to produce the image approximately 10 MHz. The segmentation system is also provided with synchronization signals from the digitizer so that the system can determine the position of each pixel within the single horizontal line and the position of the horizontal line within the image frame. The final representation of the image that is passed to the segmentation system is a serial stream of 8-bit pixels and synchronization signals at a data rate of 10.08 MHz.

The number representation for this project is an 8-bit fixed radix-two number with the radix point placed to the left of the most significant bit. Since the gradient relaxation algorithm uses only probability values between zero and one, this number system is sufficient to represent all numbers that will be used by the algorithm. The smallest number that can be represented with this scheme is 0 and the largest number represented is $1 - 2^{-8}$. Although the value 1 cannot be represented exactly, the gradient relaxation algorithm quickly converges into two classes with a large gap between the two sets of numbers representing the classes, thus allowing the differentiation between the two classes. The image pixels as provided by the video digitizer are transformed from 8-bit pixel intensities that range in value from 0 to 255 into initial probabilities that range from 0 to 1 by the initial probability assignment circuitry.

4.2 Real-Time Image Segmentation System Specification

The segmentation system consists of six major components: the TV camera, computer terminal, display monitor, CPU board, digitizer board, and segmentation board. Each of the circuit boards uses the VME bus format and these boards are mounted in a VME card cage that provides cooling, power, interboard communications, and structural integrity. Each of the system components is explained subsequently with the interconnection of these system components shown in Figure 3.

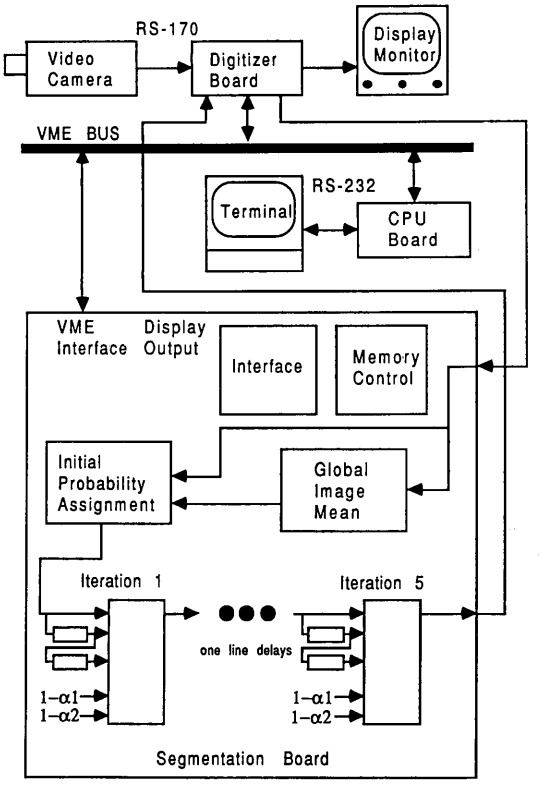

Figure 3. Segmentation system interconnection diagram.

4.2.1 TV camera. The video camera was selected on it capability to transmit images using a noninterlaced format. The relaxation algorithm performs 3 × 3 neighborhood operations, and therefore requires that three adjacent horizontal lines be available before processing can continue. Due to the alternating order of the horizontal lines with the interlace format, the entire odd frame would have to be transmitted and stored in a buffer so that the odd lines could be lined up with the even lines when they arrive in order to form the 3 × 3 neighborhood. This would greatly increase the latency incurred per image frame and would degrade the overall system performance. Therefore, the interlaced format was abandoned and a camera that uses a noninterlaced format was acquired for this project. The noninterlaced camera does not divide the image into its odd and even components. Rather, the horizontal lines are transmitted such that successive adjacent lines are in order allowing the system to proceed immediately upon the receipt of the first three lines. The resulting improvement in system latency as well as the reduction in system hardware (no need

for the odd field buffer) makes the noninterlaced camera the best choice for this system. The noninterlaced camera supplies 240 horizontal lines instead of the 480 normally transmitted via the RS-170 format. This lowered number of horizontal lines is required so that the camera is compatible with RS-170 control signals. However, the system sampling frequency remains the same since the horizontal resolution is unchanged.

4.2.2 Computer terminal. The computer terminal allows the user to communicate with the segmentation system through the CPU board.

4.2.3 Display monitor. The display monitor is connected to the digitizer board and displays the data provided by the user in real time. It displays the image data using the same noninterlaced format as the TV camera. The segmented image results can also be viewed on this display monitor.

4.2.4 CPU board. The CPU board, purchased from Plessey Corporation, consists of a VME component that implements a single board computer. The single board computer provides the entire environment for program execution that includes 512 kilobytes of RAM, 128 kilobytes of ROM, 68000 microprocessor, and an RS-232 serial interface. The CPU board serves as the supervisor to the segmentation system, providing the user with control over the system by interpreting commands typed at the computer terminal that are transmitted via the RS-232 port to the CPU board and subsequently executed on the segmentation board. The CPU board provides a simple command interpreter allowing the user to set all segmentation parameters as well as provide testing procedures for the VLSI chips themselves. The CPU board does not enter the actual segmentation process itself. Rather, it initializes various control hardware on the segmentation board to start the segmentation process. Once the segmentation process is started, it runs continuously on the segmentation board until interrupted by the CPU board. The software for the CPU board was written in both "C" and 68000 assembly languages. Low-level input/output routines were coded in assembly for efficiency and the remainder of the system was programmed in "C" for simplicity. The entire software for this project resides in two EPROMS placed on the CPU board.

4.2.5 Video digitizer board. The video digitizer board is also a VME component and was purchased from Recognition Technologies Incorporated. It provides real-time digitization at a rate of 10.08 MHz, which allows a maximum image resolution of 512 × 512 bytes. This board sends the digitized image data out as a byte-serial stream at the sample rate of 10.08 MHz. The digitizer also contains additional circuitry to allow the user to display digital images on a conventional display monitor. This display function operates as the inverse of the digitization function and the user only needs to provide the image data to be displayed in the same format as the digitizer uses along with the proper synchronization signals. The user-provided image data is displayed on the display monitor connected to the digitizer board.

4.2.6 Segmentation board. The segmentation board consists of two major partitions, the CMOS VLSI chips and the discrete circuitry necessary to perform those functions not implemented in the VLSI circuitry.

1. Discrete Circuitry: The segmentation board uses discrete logic to implement those functions that were not implemented in VLSI. These functions included test circuitry, initial probability assignment (equation (4)), calculation of the global image mean, various control and interfacing circuitry, and digital delay elements to align three adjacent horizontal scan lines for the 3 × 3 neighborhood processing.

There are two principal reasons for not implementing some system functions directly on the VLSI chips. First, only those functions that will contribute directly to the operation of the segmentation algorithm in a general purpose environment should be implemented on-chip. The general purpose parts of the segmentation algorithm include all the equations used to perform the relaxation algorithm. Conversely, the control and interfacing circuitry are parts of the system that are specific to this particular implementation. This includes the interfacing circuitry that provides communications between the video digitizer, CPU board, and other parts of the segmentation system and the control circuitry that handle memory control and supervision of the segmentation system. Second, due to restrictions on the available VLSI chip space for implementation, not all functions of the segmentation system could be implemented on-chip. The initial probability assignment and calculation of global image properties are both required for operation of the gradient relaxation algorithm but have lower computational requirements than the iterative parts of the gradient relaxation algorithm. Thus, these functions were implemented with discrete logic. This was also true for the digital delays required to align the three horizontal lines for neighborhood op-

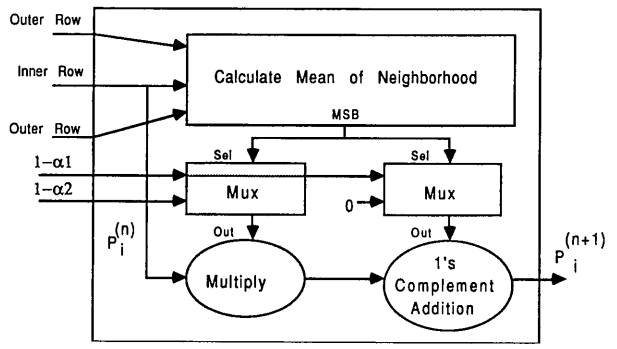

Figure 4. Block diagram of the input/output format of the real-time segmentation chip. Mean of the neighborhood is the value of $q_i(\lambda_1)$ given by equation (2).

erations. Therefore, the discrete circuitry serves to fill in the gaps that result from the lack of available silicon space for a full algorithm implementation and provides interfacing and control functions that are specific to a given application. Further details of the discrete circuitry used on the segmentation board will be referred to in Section 6.

2. VLSI segmentation chip: The equations chosen for VLSI implementation are equations (2), (5), and (6). These equations represent the bulk of the processing performed during the relaxation process due to their iterative nature. Each chip performs a single iteration. Five iterations of the algorithm were implemented that required five real-time segmentation chips. These are located on the segmentation board. The other operations, like global mean [equation (4)], required by the algorithm are performed once per image frame and represent a much lower computational burden. It was originally estimated that equations (2), (5), and (6) and the neighborhood alignment circuitry could be implemented as a single VLSI component. However, the implementations of equations (2), (5), and (6) required all the available silicon; therefore, the digital delays that performed the neighborhood alignment had to be placed off-chip.

The input and output format of the real-time segmentation chip is shown in Figure 4. Since the lack of available silicon space required the neighborhood alignment circuitry be external to the chip, each clock cycle must provide three pixels of data at a time to the VLSI chip. The three pixels that are input on each clock cycle represent the three horizontal lines after they have been aligned. Since each of these pixel values is represented by an 8-bit binary number, 24 pins are required to input the three pixels simultaneously during each clock cycle. The segmentation parameters α_1 and α_2 are also represented by 8-bit values and require a total of 16 pins for input. The actual inputs to the chip are $1 - \alpha_1$ and $1 - \alpha_2$ instead of α_1 and α_2 since doing so saves some circuitry internally and $1 - \alpha_1$ is easily converted to α_1 when necessary. The output of this chip is a single 8-bit value that represents the updated pixel probability value and requires 8 pins. These input and output requirements will require a total of 48 pins not including power and ground pins. The addition of these 2 pins and a pin reserved by MOSIS raises the total to 51 pins. The 64-pin package was chosen since it meets the needs of the 51 pins for input and output while leaving the remaining 13 pins for testing purposes. The detailed design of the real-time segmentation chip will be discussed in the next section.

5 Real-Time Image Segmentation Chip Design

The design of the segmentation chip was accomplished by dividing up the segmentation algorithm into logical sections as shown in Figure 5. Each logical partition is a representation of an algorithm block that was implemented physically with VLSI circuitry (referred to as a circuit module) and as such, met two important VLSI design criteria. First, communications between logical sections are restricted so that the physical implementations of

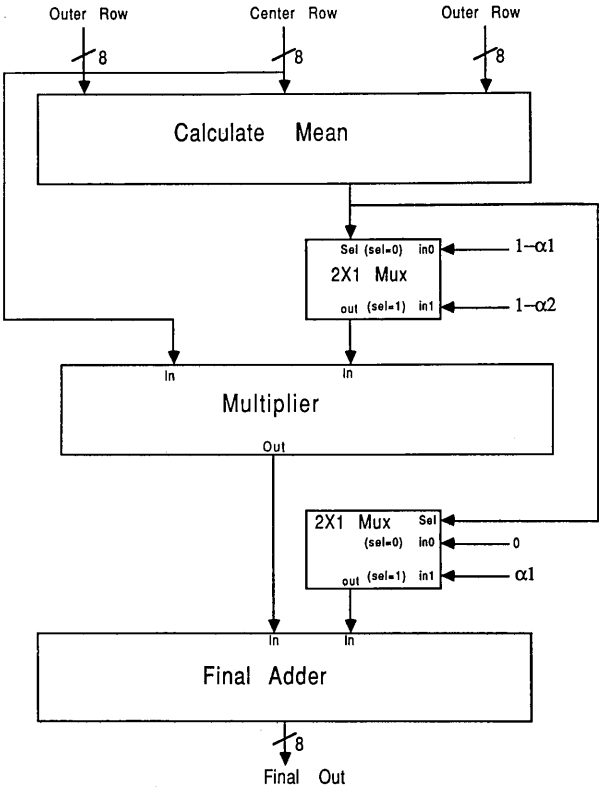

Figure 5. Logical partitions of the gradient relaxation algorithm. Mean is the value of $q_i(\lambda_1)$ given by equation (2).

these sections require 8 bits or less of interconnect. By minimizing interconnection, more effective use of chip area is achieved and the parasitic capacitance associated with interconnection is reduced (Kung and Foster 1980). Second, partition of the segmentation algorithm allows a regular layout of VLSI circuitry that makes effective use of chip area. Structuring the design process so that each circuit module could be considered separately simplified the overall design of the segmentation chip.

Circuit module synthesis began by defining the basic functions that were repeated throughout the module. Identifying these basic functions served two important purposes: 1) more time could be spent designing these functions to be efficient since they would be used many times throughout the module and 2) it allowed more accurate simulations [i.e., the use of SPICE (Weste and Eshraghian 1978)] due to the lower relative circuit complexity. Circuit simulators trade computer time for model complexity and accuracy. Therefore, simulators used for large networks of devices (1000 transistors or more) are not so accurate as those used for smaller networks. By simulating small portions of the circuit module (the intrinsic functions), much more accurate timing information was obtained. These simulations were extremely important as they formed the basis for establishing the real-time capability of the circuit.

The simulations of the designed circuit modules were performed with SPICE using data that was extracted by hand from the simulation files generated by the PPL tools. Due to the newness of the PPL tool suite, there was no automatic way of generating this data. All timing information was multiplied by a factor of 4 to allow for inaccuracy of the simulation and process variability. This factor of 4 allowed the design to have an extra margin of safety in order to guarantee meeting the timing parameters for real-time performance. This figure may seem extreme or overly pessimistic, but the CMOS PPL cell set was completely uncharacterized and no timing or other information was available. By choosing a safety factor of 4, it was hoped that a necessary safety margin could be maintained throughout the design process.

Since the system clock frequency was known, the number of basic functions that could be computed during a clock cycle were known. If there was not enough time in a single cycle, multiple cycles could be combined using pipeline methodology. Thus, the simulation results provided the information necessary to divide circuit modules into one or more pipelined stages. Pipelining is a particularly effective method for achieving speedup with low-level vision algorithms due to the regularity and repeated execution of these algorithms. The synthesis and low-level design of each circuit module will now be presented.

5.1 Mean Calculation Circuit Module

The mean of the neighborhood intensities is calculated by adding the intensities of the eight nearest pixel neighbors and dividing by 8. Since 8 is an integer power of 2, division can be performed by a shift of the operand. Consequently, the most significant bit (MSB) becomes the module output, indicating whether the mean is above or below 0.5. The inputs to this circuit module are the three lines representing the intensities of the pixels and their respective neighborhoods. The basic operation used in this module is addition. A 2-bit adder circuit was designed for use in the mean calculation module, simulated using a switch-level simulator to prove correct functionality, and processed by SPICE to extract timing information. The critical path of the adder, the carry out signal, has about 1.5 ns of propagation delay. This was multiplied by a factor of 4 to provide a safe margin in case of the CMOS process variation or some error in the simulation values. The PPL representation of the adder has been shown in Figure 2.

Maximum speedup of the module is obtained by eliminating redundant addition operations. Ignoring the constant factor of 8, equation (2) can be written in the following explicit form:

$$q_i = \left(\sum_{k=i-1}^{i+1}\sum_{l=j-1}^{j+1} p_{k,l}\right) - p_{i,j} \tag{7}$$

If equation (7) is compared with

$$q_{i+1} = \left(\sum_{k=i}^{i+2}\sum_{l=j-1}^{j+1} p_{k,l}\right) - p_{i+1,j} \tag{8}$$

in expanded notation we obtain:

$$q_i = \sum_{l=j-1}^{j+1} p_{i-1,l} + \sum_{l=j-1}^{j+1} p_{i,l} + \sum_{l=j-1}^{j+1} p_{i+1,l} - p_{i,j} \tag{9}$$

$$q_{i+1} = \sum_{l=j-1}^{j+1} p_{i,l} + \sum_{l=j-1}^{j+1} p_{i+1,l} + \sum_{l=j-1}^{j+1} p_{i+2,l} - p_{i+1,j} \tag{10}$$

It can be seen that there are two identical partial sum terms in both the previous and current mean calculation. Therefore, only one partial sum needs to be calculated for each neighborhood mean operation if the previous results are stored. This module makes use of this fact by using the explicit form in equation (9) for an execution model. The final configuration of the mean calculation module, shown in

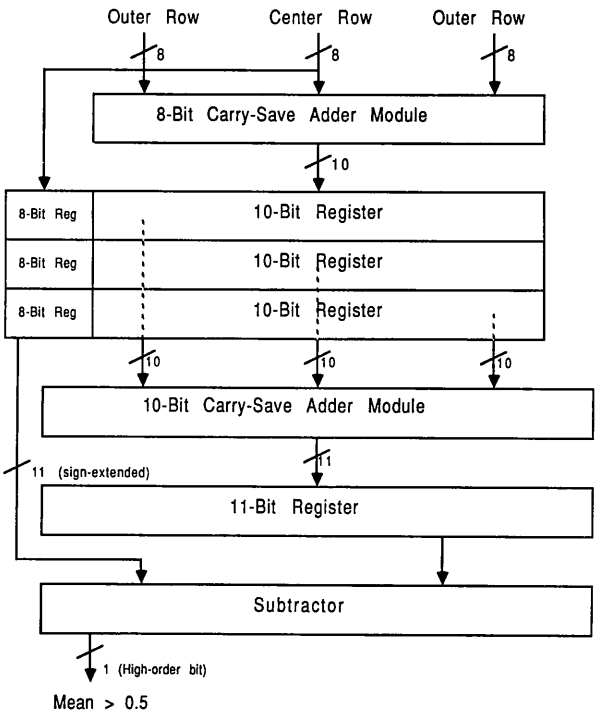

Figure 6. Neighborhood mean calculation module.

Figure 6, uses a set of carry-save adders to sum up the three pixel values that are presented at the input to the chip (Figure 7). The carry-save adder was implemented using a set of full adders without chaining the carry-out carry-in signals together. Note that the carry chain (or critical path) of the carry-save adder and full adder is shorter than if the three pixel values were added using two full adders. If two full adders are used, the carry chain propagates through nine adders. For the carry-save implementation the carry chain propagates through eight adders. The adder used to sum up the three values contained in the pipeline registers is the same carry-save adder presented earlier. There are three operands to be summed and the output is again converted using a full adder. A full adder is then used to subtract the central pixel value from the total using 2's complement arithmetic. Full adders were used for the conversion and the subsequent subtraction for the same resons of register count as previously explained. The final module design required a maximum of 11 carry propagation delays for a total execution time requirement of 66 ns, which falls well within the 100 ns maximum allowed by the 10 MHz operation parameter.

5.2 Tree Multiplier Circuit Module

The second operation required by the segmentation algorithm is an 8 × 8 multiplication. The inputs to the multiplier are the central pixel intensity and the operand produced by the mean calculation module (see Figure 5). Again, the basic function to be performed is addition, and therefore a 1-bit full adder was designed and simulated using both a switch level and linear simulator. A full adder was chosen because of its relatively compact layout when compared with carry-lookahead. Carry-save adders were not used since the intermediate partial sums will be placed in pipeline registers and a carry-save adder would require twice as many registers. Simulations of the adder circuit showed that approximately 1.5 ns must be allowed for each carry out propagation. This propagation time is the same as the adder used in the mean calculation module and is in fact quite similar to it except for the way some of the signals were routed. The propagation time was multiplied by a factor of 4 to provide a safety margin and the actual figure used was 6 ns. The number of carry propagations required to complete an asynchronous 8 × 8 multiplication exceeded the time allowed by the design specification (100 ns). Therefore, a pipelined multiplier was chosen to meet the real-time performance requirement. A tree multiplier was selected since it exhibits low latency and exploits much of the parallelism inherent in multiplication. Figure 8 describes the multiplier in detail.

Binary multiplication consists of a bytewise "anding" of the proper bit of the multiplicand term with the multiplier term. The results of these "and" operations are shifted according to the bit with which they were "anded" and then added together. The dotted lines shown in Figure 8 show the relationships among different parts of the circuit module. The eight operands involved in the multiplication are divided into four groups as shown. Each of these four groups is added together in parallel and placed into a 10-bit register. These four partial sums

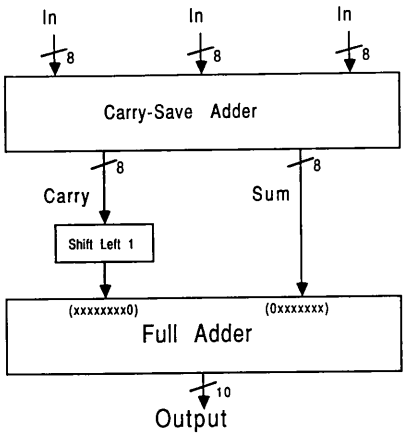

Figure 7. Addition of three pixel values using carry-save adders and a full adder.

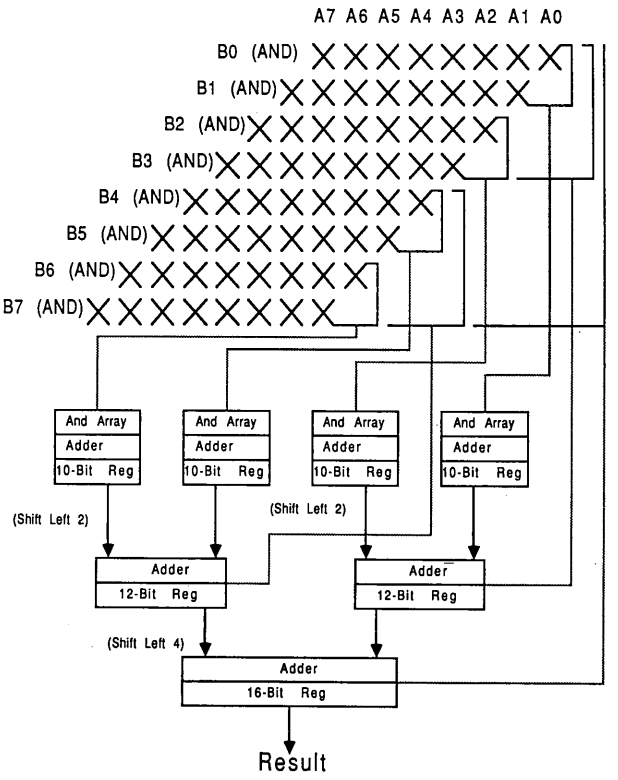

Figure 8. Tree multiplier.

are added together after being aligned properly to form two sum partials that are clocked into a 12-bit register. These, in turn, are aligned and added to form the final result. This circuit has 12 carry propagations (72 ns) and meets the design specification for operating speed (10 MHz). The tree-shaped multiplier as shown in Figure 9 is too wide to fit within the PPL array without modification. Therefore, the multiplier was folded as shown in Figure 10 to fit within the chip. This folding increases the amount of wire used for interconnect of the multiplier and will have some detrimental effects on the overall speed of the circuit due to parasitic capacitance. Since the calculated critical path propagation delay is less than 75 percent of the allotted time (100 ns) and the capacitance of the metal layer is low when compared with the poly and diffusion layers, the multiplier should still meet the real-time specifications.

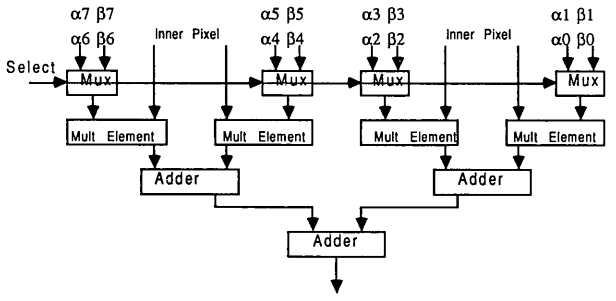

Figure 9. Functional and topological view of the multiplier.

5.3 Final Adder Circuit Module

The final adder module consists of a full adder and a multiplexer. The multiplexer selects α_1 or zero, based on the output from the mean calculation module. The adder then adds the selected constant to the result from the multiplier. This becomes the final output of the chip. The full adder used here is the same as that used in the multiplier module, and since it has only seven carry propagations (42 ns), it also meets the minimum operating frequency requirement of 10 MHz.

5.4 Test Circuitry Design

A crucial part of any VLSI design is the need to incorporate testing procedures early in the design process. If testing strategies are left until the chip is manufactured, the designer will have few choices for verifying the functionality of the part. Exhaustive testing is impractical for VLSI; the number of inputs combined with the total number of possible states make it impossible to exercise properly the chip in a reasonable amount of time. For the image processor built here testing capability was provided in an ad hoc fashion by using a multiplexer to select different internal portions of the circuit for determination of their status. Through the multiplexer, individual inputs and outputs for each module were observed and checked for correctness. The internal chip signals available through the test multiplexer

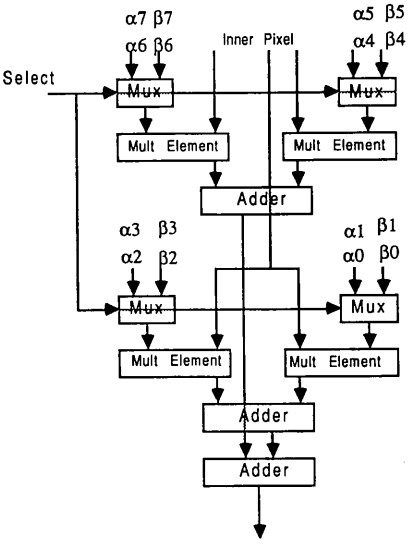

Figure 10. Folded multiplier for implementation.

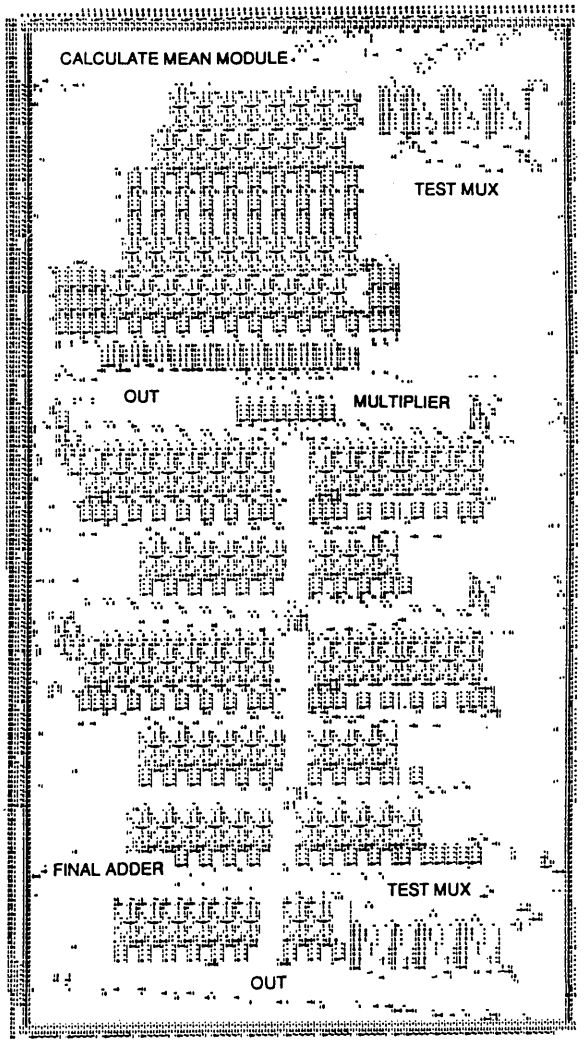

Figure 11. PPL representation of real-time segmentation chip.

include all the inputs to the chip as well as the outputs from the center pipeline and the input and output to the multiplier. An additional 1-bit output, the output of the calculate mean module, was made available directly to an output pad.

5.5 Fabrication

The final PPL design of the real-time segmentation chip is shown in Figure 11. The final design contained 9440 transistors and required 217 rows and 95 columns with 63 pads. Approximately 50 percent of the total chip area was occupied by active circuitry. The real-time segmentation chip requires eight cycles of latency to process fully a single pixel. The pin out of the VLSI chip is shown in Figure 12. The functionality of each pin of the chip is described in Table 1. The total fabrication time for the chip was about 14 to 16 weeks. A microphotograph of the fabricated chip is shown in Figure 13. Regularity and density of the chip design is clear from this figure.

Table 1. Table of segmentation chip pin functions

Pin symbol	Type	Pin function
A0–A7	Input	Outer neighborhood row
B0–B7	Input	Outer neighborhood row
C0–C7	Input	Center neighborhood row
Tstsel0–Tstsel2	Input	Test mux select control
TstMux0–TstMux7	Output	Test mux output
Al0–Al7	Input	α_1
Be0–Be7	Input	α_2
Out0–Out7	Output	Final chip output
Mean	Output	Neighborhood mean
Clock	Input	Global input clock

6 System Construction and Integration

A photograph of the completed system is shown in Figure 14. A photograph of the constructed segmentation board is given in Figure 15. The segmentation board was constructed in modular fashion. Each module was independently wired and tested and then connected to any previously constructed modules that it needed to communicate with and then

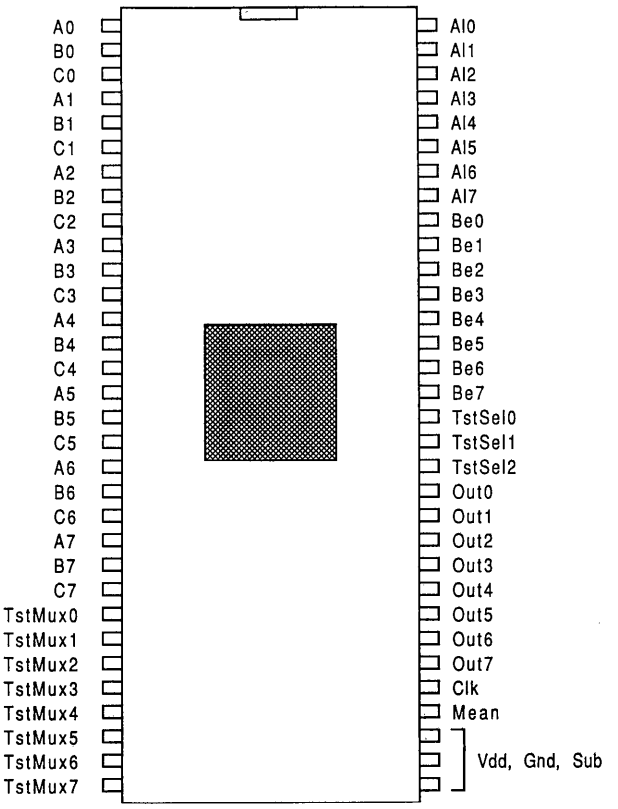

Figure 12. Chip pin connections.

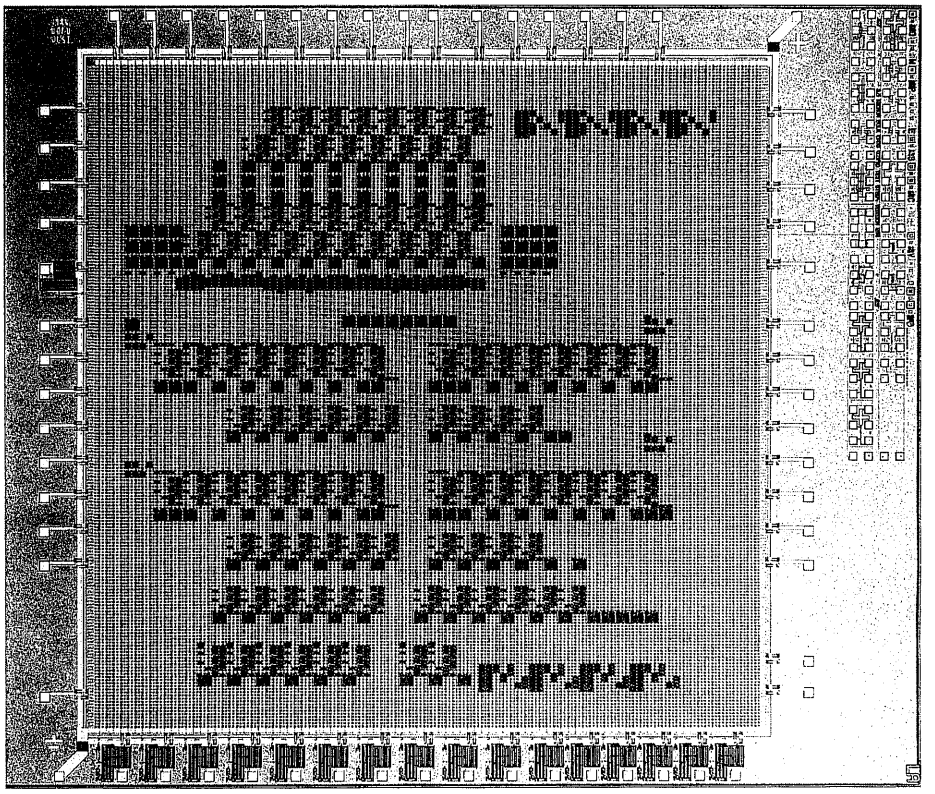

Figure 13. Photomicrograph of the fabricated segmentation chip.

the system was tested to that point. All module and system testing was performed using an oscilloscope and a logic analyzer. In the following section the function of each module on the segmentation board is presented.

6.1 Segmentation Board Circuitry

6.1.1 Memory control circuitry. The control circuitry serves as the local supervisor on the segmentation board. It was responsible for all memory timing and addressing. The control circuitry consisted of an asynchronous state machine that provided the read and write signals to the RAM used in the digital delays that aligned the three horizontal lines so that neighborhood processing could be performed. It also supplied signals to a set of counters used to address the RAM in the digital delays.

6.1.2 Interfacing circuitry. The interfacing circuitry provides communications between the video digitizer and the CPU board. Communications between the CPU and the segmentation board were memory mapped directly into the address space of the VME bus. Thus, all commands consisted of writing a specific memory location, and the results of a command or operation, if any, were received by reading a specific memory location. The interface circuitry monitors the bus until a valid request is pending for the segmentation board.

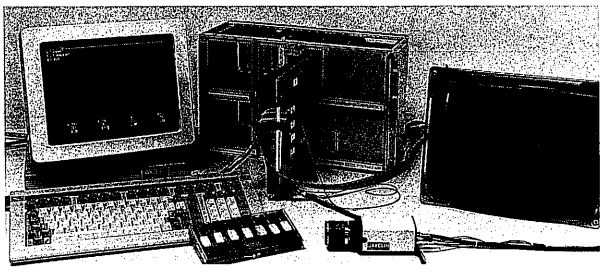

Figure 14. Photograph of real-time image segmentation system.

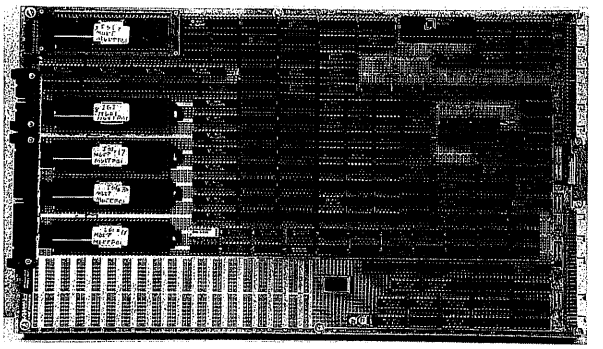

Figure 15. Photograph of the segmentation board.

Once a valid request is detected, the interfacing circuitry acknowledges the request and performs the required action. Parts of the memory used for test purposes on the segmentation board were also memory mapped in the VME bus and when accesses were performed to these locations, it communicated with the memory control circuitry.

6.1.3 Global image mean calculation circuitry. The global mean of the image was required as an input to the probability initialization process. The most straightforward way to compute the mean of the image is to add up all the pixels and divide by the number of pixels. However, due to the large number of pixels involved in a 512×240 image, the adder required to perform the addition would be too wide for real-time performance. Another way to compute the mean is to add up all the pixels on a horizontal line and to take the average of the line by dividing by the number of pixels in a horizontal line (512). Since 512 is an integer power of 2, the division can be accomplished by shifting the operand left by nine. Shifting can be hardwired so that it takes zero computation time and is therefore very efficient. All the line averages are added together and the total is divided by the number of horizontal lines (240). Unfortunately, 240 is not quite an integer power of 2; however, shifting the total by eight still provides a good estimate of the global image mean. By breaking up the addition between two adders, the carry propagation chain is much shorter, making it much easier to design the adder to run in real-time.

6.1.4 Initial probability assignment circuitry. The initial probability assignment (equation (4)) requires three operations: a subtraction, an addition, and a multiplication. This was implemented in a straightforward way with discrete multipliers and adders.

6.1.5 Neighborhood alignment circuitry. The neighborhood alignment circuitry aligns three adjacent horizontal lines so that neighborhood operations can be performed. This requires that the first of the three lines must be delayed two full horizontal line times and the second must be delayed one full horizontal line time. The delays are simply long shift registers that serve as digital delays and are implemented with high-speed RAM, registers, and address counters. The registers serve as a holding place for data, one register being used as an input and one register used as an output. During each operation the digital delay operates by reading the value of the current RAM location into the output register, writing the current input value from the input register into the current RAM address and incrementing the address counter. The address counter can be set to operate in modulo n fashion, where n is a parameter set by the user. In this way, the depth of the shift register could be easily adjusted to any value. The actual latency of the pseudo shift register is equal to $n + 2$ due to the input and output registers that are in the serial path of the shift register. The amount of delay needed depends on the size of the image and by making it adjustable; the system can be used with images of various resolution. This parameter can be programmed by the user through the command interpreter.

6.1.6 Chip testing circuitry. The chip-testing circuitry provided input stimulus and output data capture for the VLSI chips in real time. The input stimulus was provided in real time by connecting three high-speed RAM chips to the three pixel intensity inputs of the VLSI chips. The output data was captured in real time by connecting the output of the chip to a single high-speed RAM chip. During the test procedure the three input and single output RAM chips are sequenced in real time, providing and capturing the data. The output data can then be compared against some known good standard after the chip has finished processing. Rather than adding redundant circuitry, the memory control circuitry and address counter used for the neighborhood alignment circuitry were extended to provide the control necessary for the testing process. The testing circuitry has been verified to operate at up to 16 MHz.

6.2 System Software

The main purpose of the system software is to provide the user with a cohesive interface to basic system functions. This interface hides much of the underlying system detail from the user while providing a mechanism for controlling segmentation system operation. The system software consisted of two principal components—the command interpreter and the segmentation chip simulator. The command interpreter provided an interface so that the user could easily experiment with the segmentation system. The chip simulator was provided for testing purposes.

The test software provides the interface to the chip-testing circuitry on the segmentation board, whereas the image segmentation software sets up the segmentation and digitizer boards for image acquisition and segmentation. The digitizer board is set up for image acquisition by programming the

digitizer gain, setting noninterlace function bits, programming the lookup tables, and setting image resolution to 512 bytes per horizontal scan line. The segmentation board is set up by disabling the test logic and setting the segmentation bit. At this point the digitizer is acquiring images, sending these images to the segmentation board via the image bus, receiving segmented images on the same bus, and displaying these images.

The system software executes on the 68000 Single Board Computer (SBC). The system software was written in "C" and 68000 assembly language. Only the low-level routines that directly access the hardware were written in assembly for efficiency purposes. More than 95 percent of the system software was written in "C." This code was "burned" into an EPROM and resides on the SBC board.

6.2.1 Command interpreter. The command interpreter was designed to simplify user interface for the segmentation system. As explained earlier, the actual low-level command format to the segmentation board consisted of reads and write to memory locations contained on the segmentation board. If this were the only interface to the segmentation board, it would be error-prone and difficult to use. Therefore, the actual series of low-level reads and writes to the segmentation board were programmed as separate commands. To request any of the programmed operations of the segmentation system, the user need only type in the command to the interpreter. Currently, there are several commands available in the interpreter. Most of these commands deal with low-level operations used to debug the system. Two of the commands in the interpreter are used when experimenting with the segmentation system. The first is called "adjust" and initializes the real-time segmentation process. It is called adjust because it allows the user to adjust any of the segmentation parameters (*FACT*, α_1 and α_2) while viewing the screen to see the effect of the adjustment. The adjustment of the segmentation parameters is quite simple. Each of the parameters resides in a separate field on the computer terminal screen. To change the value of one of these parameters, the cursor is moved to the desired field by using one of the arrow keys located on the terminal keyboard. Once the cursor is in the proper location, the value can be incremented by pushing the up arrow and decremented by pushing the down arrow. This interface is extremely easy to use and allows the user to adjust and view the unsegmented image or the effects of any of the segmentation parameters on the image display monitor. There is one additional parameter that can be adjusted from the computer terminal, the gain of the video amplifier. This video amplifier is located on the video digitizer board and allows the user to have control over the intensity of the image through software.

6.2.2 Chip simulator. The second useful command in the interpreter is for the purpose of chip testing. It controls the chip-test circuitry and automates the chip-testing process. On the segmentation board a specific socket has been dedicated for testing the real-time segmentation chips. Testing of the VLSI chips requires only that the user place the chip to be tested in the dedicated socket and type "chip-test" to the command interpreter. The test software automatically generates test vectors, feeds them to the chip-testing logic, and captures the results. In parallel with this process, those test vectors fed to the segmentation system are also input to a chip simulator that is logically identical to chip function. The testing process automatically proceeds and reports statistics about the chip being tested.

The actual chip-testing process consists of two steps—test input data generation and output data verification. The software on the CPU board generates a pseudo-random test sequence that is written to the test input RAM located on the segmentation board. It then starts the chips operation in real time while capturing the output of the chip in the RAM connected to the output of the VLSI chip. The output data is then compared to the output from a software simulator. The software simulator is functionally identical to the real-time segmentation chip, and if the output values match those from the simulator and the segmentation chip, the chip passes the test. The results from the chip simulator are compared with those captured via the chip-testing logic located on the segmentation board. Any discrepancies between the simulated and actual results are reported to the user on the terminal monitor. The real-time test sequence is limited to 2048 bytes of data. However, this is extended by looping through the test sequence each time with different generated input data for a total of 255 times. A second version of the test that loops a total of 65,000 times called "comp-test" has also been implemented and takes approximately 24 hours per chip to complete.

7 Evaluation

7.1 Real-Time Image Segmentation Chip Evaluation

The chips were tested in two different ways before installing them in the segmentation system. A preliminary test that used a logic analyzer, oscillo-

scope, and high-speed pattern generator (constructed out of discrete components) was performed while the segmentation system was under construction. This test was somewhat crude; all patterns had to be checked by hand, thereby limiting the number of tests that could be performed. However, this did give some indication as to the correct function of the chips and was useful to verify some of the early bugs that existed in the CMOS PPL cell set. The preliminary test using the logic analyzer and oscilloscope tested the chips at frequencies up to 40 MHz. The second test consisted of the comprehensive test performed on the segmentation system and was capable of testing chips at frequencies up to 16 MHz.

Three lots of chips were received from MOSIS at various times throughout this project. The first and second lots were received within one week of one another. Of these, one lot was totally nonfunctional due to a bug in the COMS PPL cell set and was not included in the test or yield figures. Ignoring this bad lot of chips, a total of 44 chips was received from MOSIS. These 44 were tested and 33 chips were fully functional up to 16 MHz. This gives an overall yield of 75 percent. During the preliminary tests one of the chips functioned at 20 MHz, which would allow real-time segmentation of 1024 × 1024 images. This chip also passed the comprehensive test at 16 MHz. Five of these operational chips were placed on a circuit board with the required support circuitry and the entire system was evaluated with regard to real-time segmentation performance.

Each chip occupied an area of 7928 microns × 9225 microns. About 50 percent of this chip area was active area. The technology used was 3 micron CMOS, double-layer metal and the estimated steady-state power consumption per chip was less than 50 milliwatts.

7.2 Overall Segmentation System Performance Evaluation

The overall system evaluation was based on two criteria. The first criterion was with regard to the real-time operation of the segmentation system. Once the chips had been shown to be fully operational at the required system clock rate, the system was thoroughly tested to ensure that all discrete circuitry functioned properly at the speeds necessary to segment the 512 × 240 image (10 MHz). This testing was performed using a logic analyzer and other standard laboratory instruments.

The second criterion was with regard to the segmentation process (Bhanu 1986) itself and is more difficult to evaluate than the real-time performance of the system. While test instruments can indicate, in quantitative fashion, whether or not the system functions in real time, the only way used here to judge the segmentation results was through observation. This part of the qualitative evaluation was done by presenting scenes to the camera and sequentially observing both the segmented and unsegmented images on the display monitor. This allowed a qualitative judgment to be made as to the effectiveness of the gradient relaxation technique for real-time segmentation.

Two classes of images were used for evaluation of the segmentation system. They consist of complex outdoor scenes and photomicrographs of precancerous human cells. The outdoor scenes were obtained by using the TV camera to image buildings near our research facility. The photomicrographs of human cells were obtained from 35 mm color slides. In all cases the images were noninterlaced and of resolution 512 (horizontal) × 240 (vertical). The pictures in this paper were obtained by directly photographing the display monitor. All segmentation was performed in real time and no frame buffering of the results was performed.

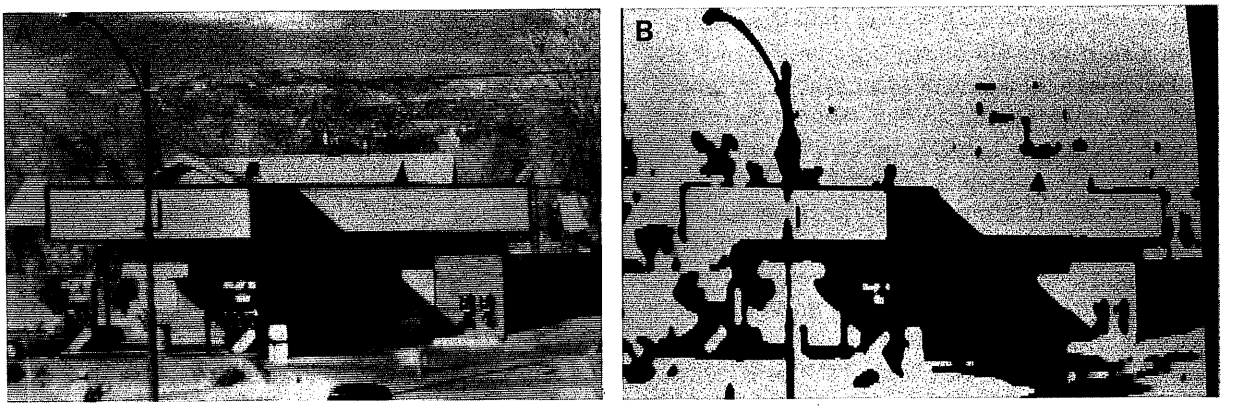

Figure 16. (A) An outdoor image; (B) segmented outdoor image, $FACT = 0.7$, $\alpha_1 = 0.63$, $\alpha_2 = 0.52$.

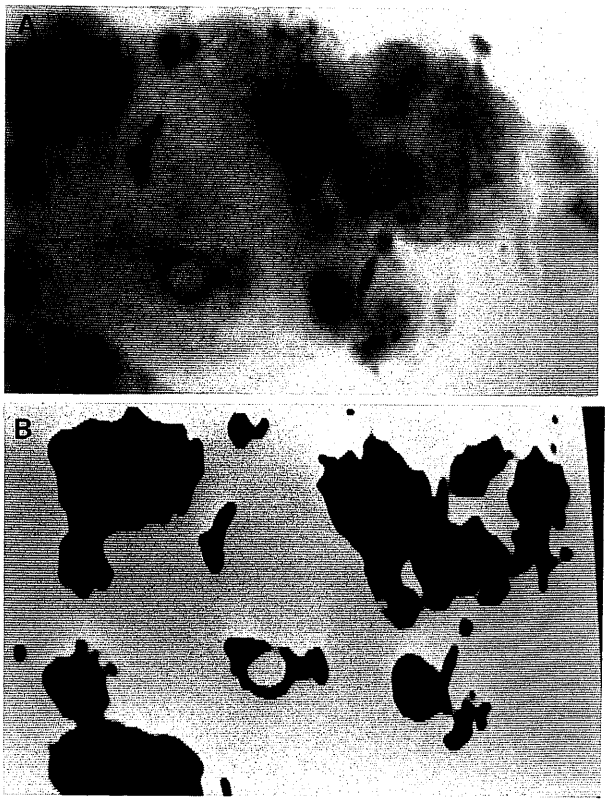

Figure 17. (A) A cancer cell image; (B) segmented cancer cell image, $FACT = 1.0$, $\alpha_1 = \alpha_2 = 0.4$.

Figure 16A shows an example of a building with a lamp post in the foreground. There are mountains as well as trees in the background and the lighting is of high contrast resulting in strong shadows that obscure the front middle section of the building. Figure 16B shows the segmented image. In particular, note that the lamp post has been segmented from the background. Also, note the large rectangular façade (located at the viewer's top left-hand side) of the building that is partially segmented into a border and center portion. This corresponds well with the unprocessed image. The background has been almost totally eliminated except for a few small regions.

Figure 17A shows the unsegmented view of a biomedical image containing a portion of a malignant cervical squamous epithelial cell. The segmented image given in Figure 17B shows various features of the cell such as the clumped-up nucleus and the vacuole-like structure (the object with a circular hole in the middle) that are irregular cell features of interest to a lab technician performing some analysis on the cell. The vacuole-like structure segments quite well in spite of its intensity being very close to the background of the cell image. All detail in the nucleus is lost; however, the segmentation process could guide further analysis of the cell by providing a mask for the region of interest of the cell image.

The segmentation process is controlled by the input parameters α_1 and α_2. As illustrated in Figure 18, the magnitude of these paramters controls the speed at which the relaxation proceeds at each iteration. Figure 18A shows the segmented image for relatively low values of α's. Figures 18B to 18D show the results when α's are increased, until Figure 18D when full segmentation is obtained. Full segmentation is obtained when only two differing intensity values are displayed on the monitor. Actual values for α's are given in the figure captions. Note that only five iterations are used for the results shown in this paper. More results on covergence rate and quality of results for varying values of α_1 and α_2 are given in the paper by Bhanu and Faugeras (1982).

The segmentation system tolerates variability in scene lighting; variations in ambient light levels have little effect on the image segmentation. Figures 19 to 21 show this aspect of the segmentation system. Each pair of images shows the actual image as well as the segmented image. The first image (Figure 19A) was recorded under optimal lighting conditions (maximum contrast). The next image (Figure 20A) was recorded with the lens on the TV camera stepped down by one stop (which reduces the available light by one-half). As seen, the segmentation (Figure 20B) varies little from the segmentation (Figure 19B) of the original image recorded using twice as much light. The following image (Figure 21A) is obtained by stepping down once again resulting in a recorded image with only one-fourth the light of that in Figure 19A. Again, the final segmentation (Figure 21B) differs little from the image recorded with four times as much light. As can be seen, the segmentation of each of these three images (Figures 19A, 20A, and 21A) varies little from one to the other even though these images vary greatly in both contrast and brightness.

Note that we use a noninterlaced camera (image size 512 × 240) and as such do not need a frame buffer. If higher image resolution (512 × 512) is desired, then no modifications are needed in the chip.

Further, it is not possible to acquire 512 × 512 images from TV camera using the RS-170 standard. Since RS-170 transmits images in interleaved fashion, the actual image dimensions are 512 × 480 pixels and a frame buffer would be required to assemble fully an image before passing it to the segmentation system for processing. However, if 512 × 512 images were available (or even 512 × 256), system accuracy would increase since the computed mean value would be more accurate. This

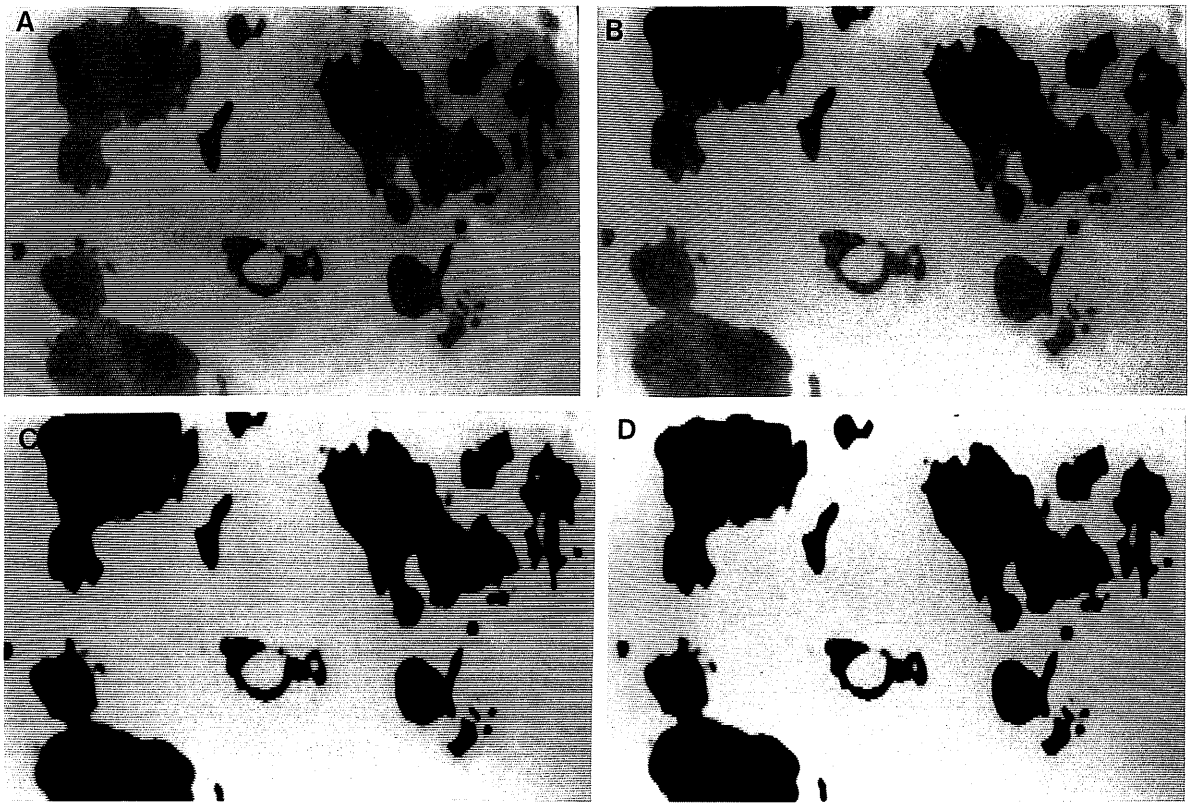

Figure 18. (A) Segmented cancer cell image, $FACT = 1.0$, $\alpha_1 = \alpha_2 = 0.02$; (B) segmented cancer cell image, $FACT = 1.0$, $\alpha_1 = \alpha_2 = 0.04$; (C) segmented cancer cell image, $FACT = 1.0$, $\alpha_1 = \alpha_2 = 0.10$; (D) segmented cancer cell image, $FACT = 1.0$, $\alpha_1 = \alpha_2 = 0.25$.

improvement occurs since only simple shifting was used to approximate division by 256. If there were 256 lines, the division would be exact since 256 is 2^8. With 512×512 images the segmentation board requires no modifications. The current digitizer board could be used if a frame buffer were available. System software need not be changed other than to support the digitizer board and frame buffer. The system was built with the intention of segmenting 512×512 images and as such, the probability assignment logic and delays can be programmed to handle 256×256, 256×512, and 512×512 images. The programming is performed by setting the correct bits in the system command register. The only image-resolution dependent sections of the segmentation board are the probability assignment logic and the shift-register delays used to line up the three-line neighborhood. The delays are also programmable and will support images with 2048 pixels per horizontal scan line.

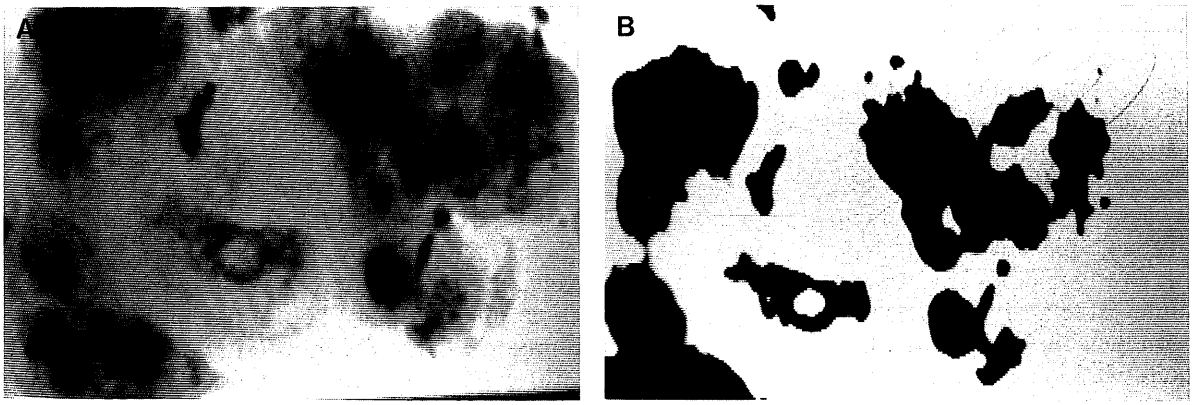

Figure 19. (A) A cancer cell image, camera F-stop = 2.8; (B) segmented cancer cell image, $FACT = 1.0$, $\alpha_1 = \alpha_2 = 0.50$.

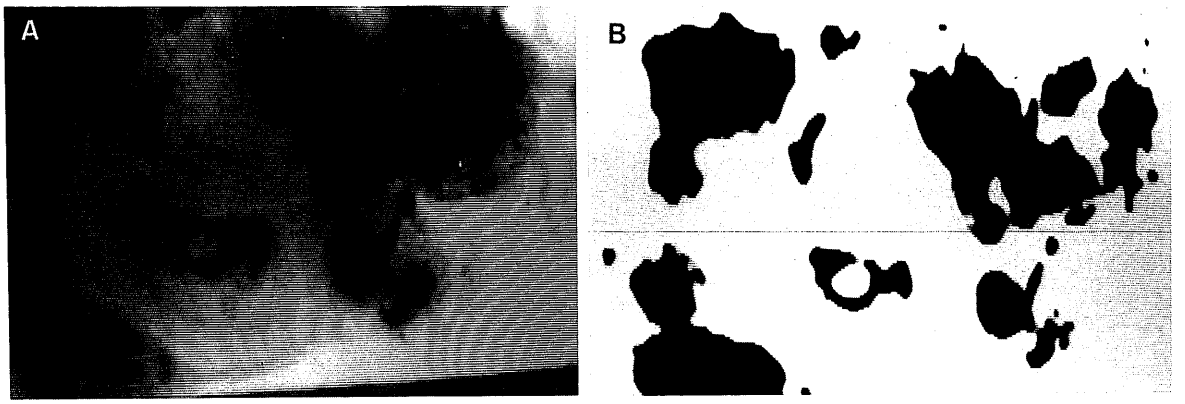

Figure 20. (A) A cancer cell image, camera F-stop = 4.0; (B) segmented cancer cell image, $FACT = 1.0$, $\alpha_1 = \alpha_2 = 0.50$.

8 Conclusions

This research illustrates the advantages of using a high-level CAD design tood such as PPL for the design of real-time images processors. These high-level tools provide design times that are an order of magnitude less than full-custom techniques. Due to the reduction in design time provided by PPL, experimental circuitry can be quickly prototyped directly into VLSI, thus avoiding the breadboard step. The design time of the real-time segmentation chips was about three months. Fabrication time was also about three to four months. A total design and fabrication time of about six to seven months was all that was required to implement fully a single chip. PPL puts the advantages of VLSI design within the reach of small research groups.

Not only does this research prove the feasibility of using PPL to design real-time image processors, it shows that the fabricated chips can meet the needs of practical systems. The segmentation board occupies just one slot in a VME card cage. When integrated with a data terminal, TV camera, digitizer, and CPU card, the entire system needs only a modest amount of space and power. This reduction in space requirements, system complexity, and power consumption is a direct benefit of VLSI technology.

This research also shows the effectiveness of using a pipelined approach to implement the relaxation algorithm. The pipelined approach and the computational requirements of the relaxation algorithm allow each of the iterations of the algorithm to be almost totally overlapped. In the final design a total of 8 clock cycles of latency was incurred directly in the real-time segmentation chip itself. Thus, the latency per iteration is the sum of the two horizontal line delays (1300 clock cycles) plus the 8 clock cycles of latency, for a total latency of 1308 clock cycles per iteration. The total latency for the segmentation system is equal to the number of iterations (five) times 1308. This amounts to 6540 clock cycles, which at 10 MHz corresponds to a delay of

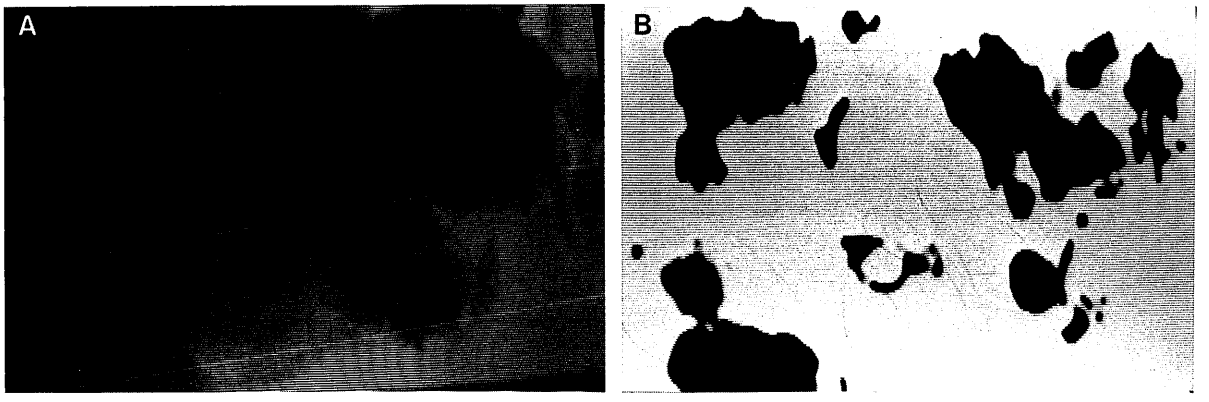

Figure 21 (A) A cancer cell image, camera F-stop = 5.6; (B) segmented cancer cell image, $FACT = 1.0$, $\alpha_1 = \alpha_2 = 0.50$.

0.654 ms or less than six horizontal scan times. This delay was imperceptible during the experiment. This research forms the basis for further work in VLSI and image processing. Possible extensions to this research include adding various enhancements to the segmentation system such as the capability to segment images into multiple classes and the incorporation of edge information (Bhanu and Holben 1990, Bhanu and Parvin 1987).

References

Anderson RL (1985) Real-time gray-scale video processing using a moment-generating chip. IEEE Journal of Robotics and Automation, RA-1(2)(June):79–85

Barbe DF (1981) VHSIC systems and technology. IEEE Computer, 14(2):13–22

Bhanu B (1986) Automatic target recognition: State of the art survey. IEEE Transactions on Aerospace and Electronic Systems, AES-22(4)(July):364–379

Bhanu B, Faugeras OD (1982) Segmentation of images having unimodal distributions. IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-4(4)(July):408–419

Bhanu B, Faugeras OD (1984) Shape matching of two-dimensional objects. IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-6(2)(March):137–156

Bhanu B, Holben RD (1990) Model based segmentation of FLIR images. IEEE Transactions on Aerospace & Electronic Systems, AES-26(1)(January)

Bhanu B, Parvin BA (1987) Segmentation of natural scenes. Pattern Recognition, 20(5):487–496

Fouse SD, Cumming AD, Wong VS, Nudd GR (1981) Development of VLSI image understanding systems. Technical Report 1050, USC Image Processing Institute, Los Angeles, California (September), 111–128

Fu KS (Ed.) (1984) VLSI for pattern recognition and image processing. Springer-Verlag

Kruger RP, Thompson WB (1981) A technical and economic assessment of computer vision for industrial inspection and robotic assembly. Proceedings of IEEE 69:1524–1538

Kung HT, Foster MJ (1980) Design of special-purpose VLSI chips: Example and opinions. IEEE Computer, 13(1)(January):26–40

Mead C, Conway L (1980) Introduction to VLSI systems. Addison-Wesley Publishing Company, Reading, MA

Nagin P, Hanson A, Riseman E (1982) Studies in global and local histogram guided relaxation algorithms. IEEE Transactions on Pattern Analysis and Machine Intelligence. PAMI-4(3)(May):263–277

Nudd GR, Nygaard PA, Fouse SD, and Nussmeier TA (1979) Development of custom designed integrated circuits for image understanding. Technical Report USCIPI 910, USC Image Processing Institute, Los Angeles, California, 120–145

Offen RJ (1985) VLSI image processing. McGraw-Hill Book Company, New York

Ohlander R, Price K, Reddy DR (1978) Picture segmentation using a recursive region splitting method. Computer Graphics and Image Processing 8:313–333

Rosenfeld A, Hummel R, Zucker S (1976) Scene labeling by relaxation operations. IEEE Transactions on Systems, Man and Cybernetics SMC-6(6)(June):420–433

Smith KF (1983) Design of regular arrays using CMOS in PPL. Proceedings of IEEE International Conference on Computer Design/VLSI in Computers, 1–4

Smith KF, Carter TM, Hunt CE (1982) Structured logic design of integrated circuits using the storage/logic array (SLA). IEEE Transactions on Electron Devices 4(April):765–776

Smith KF, Israelson P (1985) Comparison of the Path Programmable Logic design methodology with other custom and semicustom approaches. IEEE International Conference on Computer Design/VLSI in Computers, (October):73–76

Weste NHE, Eshraghian K (1985) Principles of CMOS VLSI design, a system perspective. Addison-Wesley Publishing Company, Reading, MA

Willet TJ (1978) Hardware implementation of image processing using overlays: Relaxation. Proceedings DARPA Image Understanding Workshop, 175–181