FPGA-based High Performance Page Layout Segmentation

Nalini K. Ratha, Anil K. Jain
Department of Computer Science
Michigan State University
East Lansing, MI 48824
ratha@cps.msu.edu, jain@cps.msu.edu

Diane T. Rover
Department of Electrical Engineering
Michigan State University
East Lansing, MI 48824
rover@ee.msu.edu

Abstract

A page layout segmentation algorithm for locating text, background and halftone areas is presented. The algorithm has been implemented on Splash 2 – an FPGA-based array processor. The speed as determined by the Xilinx synthesis tools projects an application speed of 5 MHz. For documents of size 1,024 × 1,024 pixels, a significant speedup of two orders of magnitude compared to a SparcStation 20 has been achieved.

1. Introduction

Image segmentation is an important step in a computer vision system. The process of spatial partitioning of an image into mutually exclusive, connected and "homogeneous" image regions is known as image segmentation [3, 9]. Each region is expected to be homogeneous with respect to a defined image property such as gray level, color or texture. Typically, image segmentation is carried out in the early stages of a vision system to facilitate proper handling of different regions. An image segmentation problem is similar to the problem of multidimensional pattern clustering in the sense that we need to define the similarity criterion between pixels or pattern vectors and the number of segments or clusters [6]. Algorithms for image segmentation are based on the following approaches: (i) thresholding or clustering, (ii) boundary detection, and (iii) region growing. The similarity between pixels is based on attributes such as gray level, color, texture and optical flow.

In this paper, we describe the design and implementation of a high performance system for image segmentation. In particular, we demonstrate the design of a FPGA-based architecture for page layout segmentation based on "local" properties of the image such as mean and variance. In an automated document image understanding system, page layout segmentation plays an important role for segmenting text, graphics and background areas. Segmentation allows us to apply character recognition algorithms to only the regions of text, and to apply graphics and symbol recognition algorithms in the graphics areas. A number of algorithms have been reported for page layout segmentation [10]. Haralick et al. [2] use image morphology-based techniques. Jain and Bhattacharya [4] have used a multi-channel filtering approach based on Gabor filters. Jain and Chen [5] have used color information along with Gabor filter outputs for page layout analysis applicable to locating address labels. Recently, Jain and Karu [7] have proposed an algorithm to learn texture discriminating masks needed for segmentation. The performance of this approach for page layout segmentation has been demonstrated by Jain and Zhong [8]. A simpler method for page layout segmentation is based on computation of mean and variance in a limited spatial domain (7 × 7) around each pixel. We have implemented this algorithm (see Section 3) on Splash 2.

The architecture of a general-purpose processor is designed so that it provides a reasonable performance over a wide variety of applications. Even the newer processors having the latest architectural features such as superscalar, superpipelined units are not capable of providing support for the high computing demands of specialized applications. Many compute-intensive application areas, including image processing, face the challenge of optimizing the performance of algorithms on general-purpose architectures. A number of parallel processing approaches have been used for image processing applications. Application Specific Integrated Circuits (ASICs) have been designed for situations where parallel processing techniques are not able to provide the desired performance cost-effectively. However, the ASIC-based approach suffers from the following limitations: (i) once an ASIC is fabricated, it is difficult to make changes in the design; (ii) the design cycle is fairly long; and (iii) ASIC solution is very costly if a large number of units are not produced to amortize the fixed cost of the masks needed for fabrication. These limitations of ASICs can be overcome by adopting the Field-Programmable Gate Array (FPGA)-based custom computing approach. Designers

0-8186-7502-0/96 $5.00

can implement their algorithms on FPGAs relatively easily without using any special fabrication facility. The main architectural advantage of this approach in contrast to general-purpose processing is that the designer is not constrained by a general-purpose instruction set. The special instructions suitable for the given application can be implemented to enhance the overall performance of the system. Using the reconfigurable property of the FPGAs, several applications can be developed, or existing designs can be modified. The design time is reduced substantially, and the system cost is significantly less because off-the-shelf components can be used.

The paper is organized as follows. Section 2 contains a brief description of Splash 2, the FPGA-based attached array processor used for prototyping the algorithm. The sequential algorithm and its computational complexity is outlined in Section 3. The mapping of the algorithm onto Splash 2 is presented in Section 4. The experimental results are described in Section 5. Conclusions and future work are outlined in Section 6.

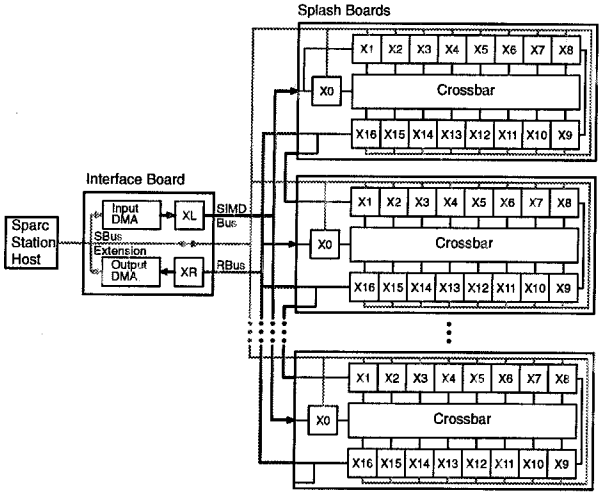


Figure 1. Architecture of Splash 2.

2. Splash 2

The Splash 2 system consists of an array of Xilinx 4010 FPGAs. Figure 1(a) shows a system-level view of the Splash 2 architecture. The Splash 2 is connected to the host through an interface board that extends the address and data buses. The Sun host can read/write to memories and memory-mapped control registers of Splash 2 via these buses. A detailed description of the system is given in [1]. Each Splash 2 processing board has 16 Xilinx 4010s as PEs ($X_1 - X_{16}$) in addition to a seventeenth Xilinx 4010 (X_0), which controls the data flow into the processor board. Each

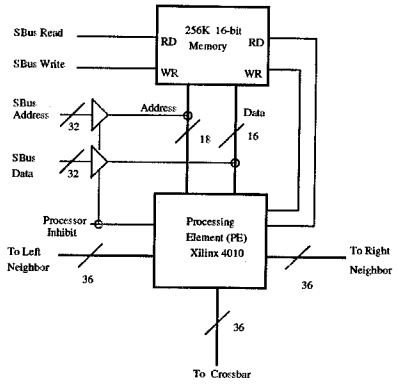


Figure 2. A processing element of Splash 2.

PE has 512 KB of memory. The Sun host can read/write this memory. The PEs are connected through a crossbar that is programmed by X_0. There is a 36-bit linear data path (SIMD Bus) running through all the PEs. The PEs can read data either from their respective memory or from any other PE. A broadcast path also exists by suitably programming X_0. The processor organization for a PE is shown in Figure 2. The Splash 2 system supports several models of computation, including PEs executing a single instruction on multiple data (SIMD mode) and PEs executing multiple instructions on multiple data (MIMD mode). It can also execute the same or different instructions on single data by receiving data through the global broadcast bus. The most common mode of operation is systolic in which the SIMD Bus is used for data transfer. Individual memory available with each PE makes it convenient to store temporary results and tables.

To program Splash 2, we need to program each of the PEs (X_1- X_{16}), the crossbar, and the host interface. The crossbar sets the communication paths between PEs. In case the crossbar is used, X_0 needs to be programmed. The host interface takes care of data transfers in and out of the Splash 2 board. A special library is available to support these features using VHDL programming as described in [1]. The synthesis process involves the following stages: (i) VHDL to XNF translation: to obtain a vendor specific netlist from the VHDL source code; (ii) Partition, Placement and Routing: to fit the logic generated onto a physical PE; (iii) Delay and Timing Analysis: to analyze the timing and delay; (iv) XNF to bit stream translation; and (v) Bit stream to raw file generation. The raw file is loaded onto each PE, and a configuration file for the crossbar is used to describe the crossbar usage. The host uses these files to control the attached processor through a set of routines callable by a C program. Using the host interface, the memory addressable by each PE can be initialized.

3. Page Layout Segmentation Algorithm

The page layout segmentation problem can be considered as assigning a class label to each pixel in an image. We are interested in assigning one of three labels to a pixel: background, text, and halftone. The page segmentation algorithm has three stages of computation, namely, (i) mean gray value in a window, (ii) variance of the gray value in a window, and (iii) final label assignment. The flowchart of the segmentation algorithm is shown in Figure 3. The

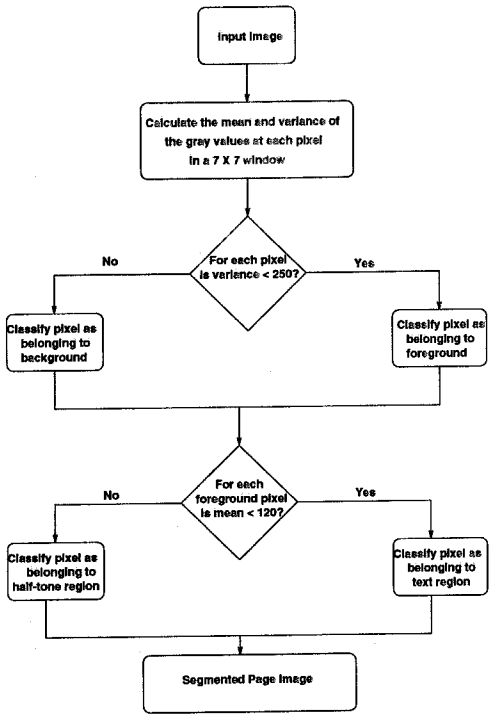


Figure 3. Flowchart of the page layout algorithm.

input to the algorithm is the gray level scanned image of the document and the output is the labeled image, where each pixel is assigned one of the three class labels. A sample input image and the segmentation result produced by this algorithm are shown in Figure 4. This page segmentation algorithm takes about 90 seconds of CPU time on a SPARCstation 20 for a 1,024 $\times$ 1,024 image. The output of the segmentation algorithm (shown in Figure 4(b)) is fed into a post processing stage [8] to place boxes around regions of interest. This "block" representation of Figure 4(a) is shown in Figure 4(c). Our interest here is to implement only the segmentation algorithm on Splash 2.

We now summarize the computational requirements of our page segmentation algorithm. The spatial neighborhood size for computing the mean and variance is chosen to be 7 $\times$ 7. For computing the mean value, we have 48 addition operations. The variance computation involves 49 subtraction operations and 49 multiplications, followed by 48 additions. This excludes the operations needed to compute the array indices. Hence, for each pixel, we need approximately 200 arithmetic operations. A 1,024 $\times$ 1,024 image, therefore, requires approximately 200 million arithmetic operations. It is now evident that the computational requirements of our segmentation algorithm are enormous; a special processor is necessary to achieve real-time performance.

4. Mapping the Page Layout Algorithm onto Splash 2

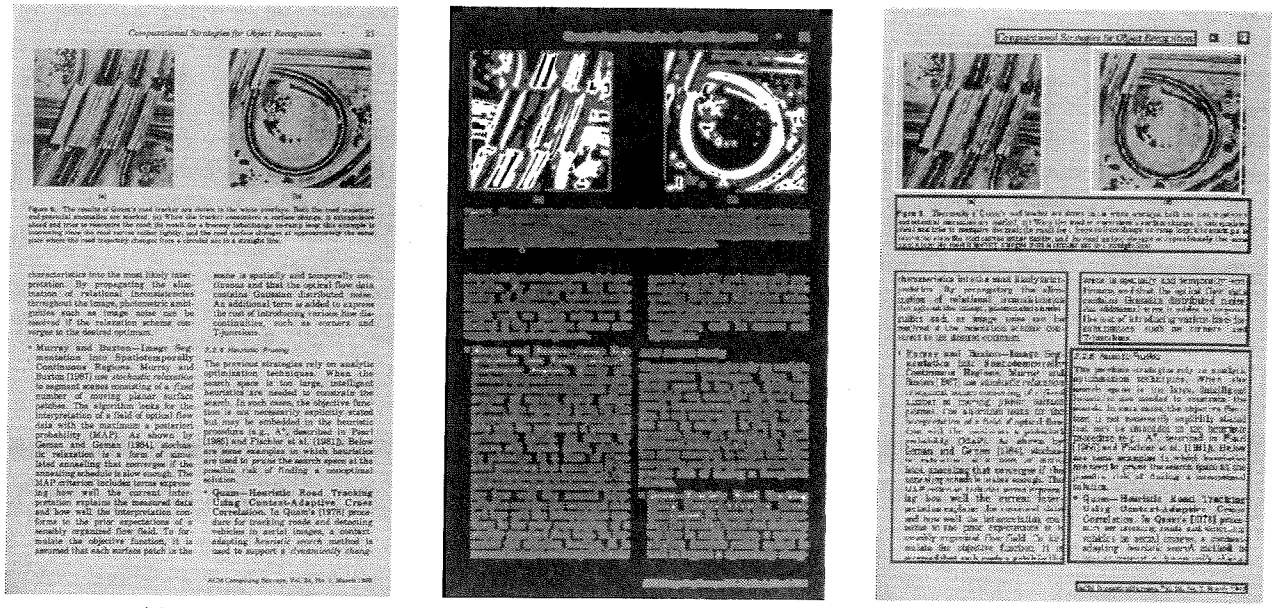
There are two main phases in the segmentation algorithm, not considering the post processing stage: computing mean and variance and classifying each pixel. The computation of mean and variance is done in a parallel fashion. The mean value can be computed by convolving the input image with a 7 $\times$ 7 mask of all 1's. For this purpose, we use our convolution algorithm which runs at a clock rate of 19 MHz on Splash 2 [11]. The virtual to physical PE mapping is the same as in our earlier approach. The schematic for convolution using a systolic algorithm is shown in Figure 5. The expression for the variance σ^2 in an $n \times n$ window can be written as follows:

$$\sigma^2 = \frac{(\sum_i \sum_j I_{ij}^2) - N\mu^2}{N}, \qquad (1)$$

where I_{ij} is the gray value at pixel (i, j), μ is the mean of the gray value in the $n \times n$ window, and $N = n^2$ is the total number of pixels in the window. From eq. (1), it can be easily seen that the variance computation can be pipelined into two stages. In the first stage of the pipeline, the sum of squares of the gray values can be computed. In the second stage, the variance can be computed, which needs the mean value, μ, as an input. The sum of squares is easily translated to a convolution operation again with a mask of all 1's as shown in Figure 6. We assume the availability of a single-board Splash system having sixteen physical PEs. Figure 6 depicts the configuration of Splash 2. The mean computation is performed on $X_1 - X_7$; and the sum of squares computation is performed on $X_8 - X_{14}$. X_{15} computes the variance and assigns the final label to a pixel.

5. Experimental Results

The sequential algorithm has been written in C and tested on several input images. On a SPARCstation 20 (33 MFLOPS, 82 MIPS), it takes about 90 seconds of CPU time to segment a 1,024 $\times$ 1,024 image. The algorithm has been mapped onto Splash 2. The application speed is projected



(a) (b) (c)

Figure 4. Page Layout Segmentation. (a) Input gray-level image; (b) Result of the segmentation algorithm; (c) Result after post-processing.

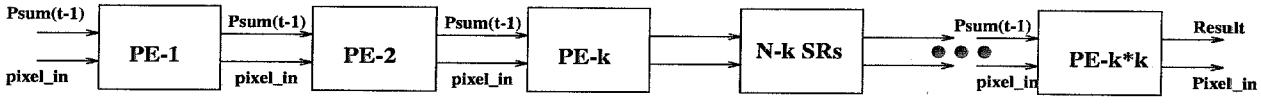


Figure 5. Schematic for filtering on Splash 2.

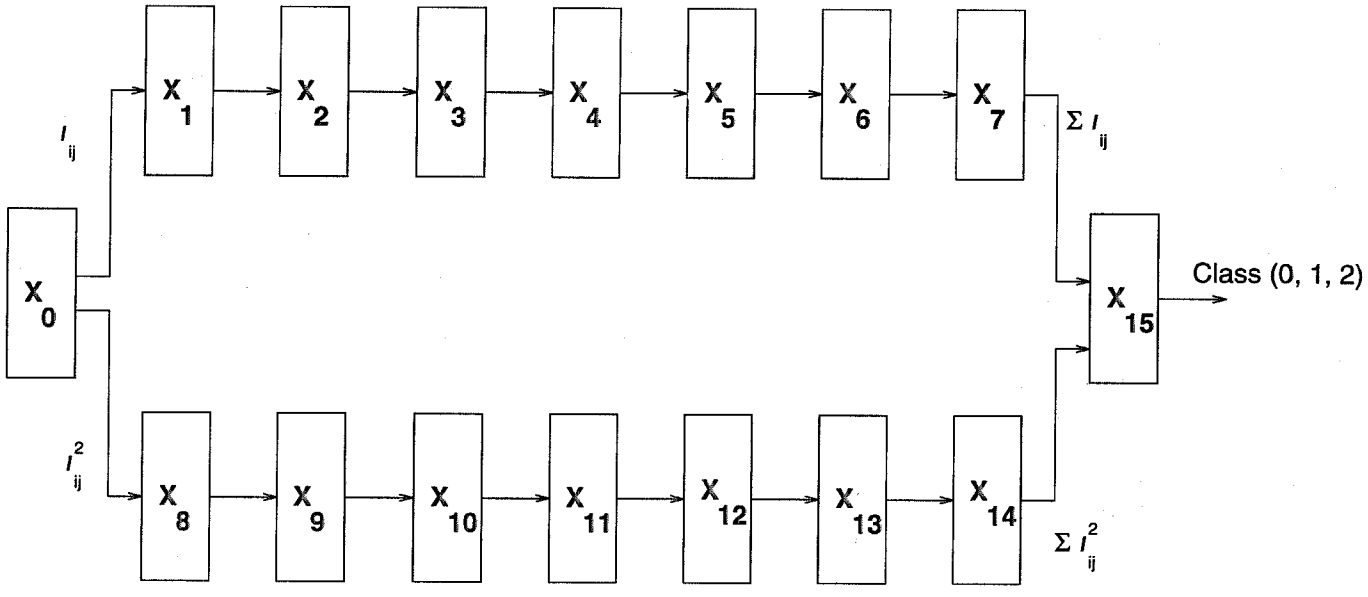


Figure 6. Schematic for page layout segmentation on Splash 2.

as 5 MHz from the Xilinx synthesis tools. At runtime, we are presently limited to a 1 MHz clock due to the setup in the interface board. Using the Splash 2 at 5 MHz, the execution time for the same page segmentation algorithm is expected to be 360 milliseconds. Therefore, a speedup of close to 250 has been achieved. The simulation results are shown for the PEs X_7, X_{14} and X_{15} in Figure 7. The output of X_7 shows the sum of the pixels and the output of X_{14} shows the sum of squares of pixels. Various signals in X_{15} show the computations of mean and variance and the final label assignment. The final labels are stored in the local memory of X_{15} which is read by the host at the end.

The host interface program written in C reads the input image, creates windows of 32×32 pixels and loads them on X_0 memory. The window size of 32×32 pixels is a restriction laid down by the convolution stage. After running the specified number of clock ticks for Splash, the result is produced and stored in X_{15} memory. The host reads back this result from the Splash board. This procedure is repeated for all the possible windows in the input image. The boundaries of the 32×32 windows are shown in Figure 8(a). Figure 8(b) shows the segmented output of the input image (Figure 4(a)) generated by the Splash system. This segmented output is identical to the output generated by the sequential algorithm (see Figure 4(b)).

6. Conclusions and Future Work

Our analysis of the computational demand of page layout segmentation shows a large disparity between the computational requirements and the available computational power at an affordable cost. This gap is bridged by using novel custom computing machines. We have described a configuration of the Splash 2 FPGA-based system for meeting the computational load of a page layout segmentation algorithm. Our implementation leads us to the conclusion that custom computing machines will play a vital role in designing certain compute-intensive applications. The Splash 2 system can contain multiple processing boards, allowing greater speedups. This will be carried out in the next phase of the project.

In the current implementation, we have observed that the overall clock speed of 5 MHz on Splash 2 is limited because of the slow integer multipliers in the Synopsys library. We will explore replacing these with faster multipliers that will improve the overall application speed by a factor of two. We are in the process of evaluating different hardware integer multipliers.

References

[1] D. A. Buell, J. M. Arnold, and W. J. Kleinfelder, editors. *Splash 2: FPGAs for Custom Computing Machines*. IEEE Computer Society Press, Los Alamitos, 1996.

[2] R. M. Haralick. Document image undestanding: geometric and logical layout. In *Proc. of IEEE Computer Vision and Pattern Recognition*, pages 385–390, 1994.

[3] R. M. Haralick and L. G. Shapiro. Image segmentation techniques. *Computer Vision, Graphics, and Image Processing*, 29(1):100–132, January 1985.

[4] A. K. Jain and S. Bhattacharjee. Text segmentation using Gabor filters for automatic document processing. *Machine Vision and Applications*, 5:169–184, 1992.

[5] A. K. Jain and Y. Chen. Address block location using color and texture analysis. *CVGIP: Image Understanding*, 60(2):179–190, September 1994.

[6] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, Englewood Cliffs, New Jersy, 1988.

[7] A. K. Jain and K. Karu. Learning texture discrimination masks. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 1996. To appear.

[8] A. K. Jain and Y. Zhong. Page layout segmentaion based on texture analysis. In *Proc. 2nd International Conf. on Image Processing*, pages 308–311, 1995.

[9] N. R. Pal and S. K. Pal. A review on image segmentation techniques. *Pattern Recognition*, 26(9):1277–1294, September 1993.

[10] T. Pavlidis and J. Zhou. Page segmentation and classification. *CVGIP: Image Understanding*, 54(6):484–486, November 1992.

[11] N. K. Ratha, A. K. Jain, and D. T. Rover. Convolution on Splash 2. In *Proc. of IEEE Symposium on FCCM*, pages 204–213, 1995.

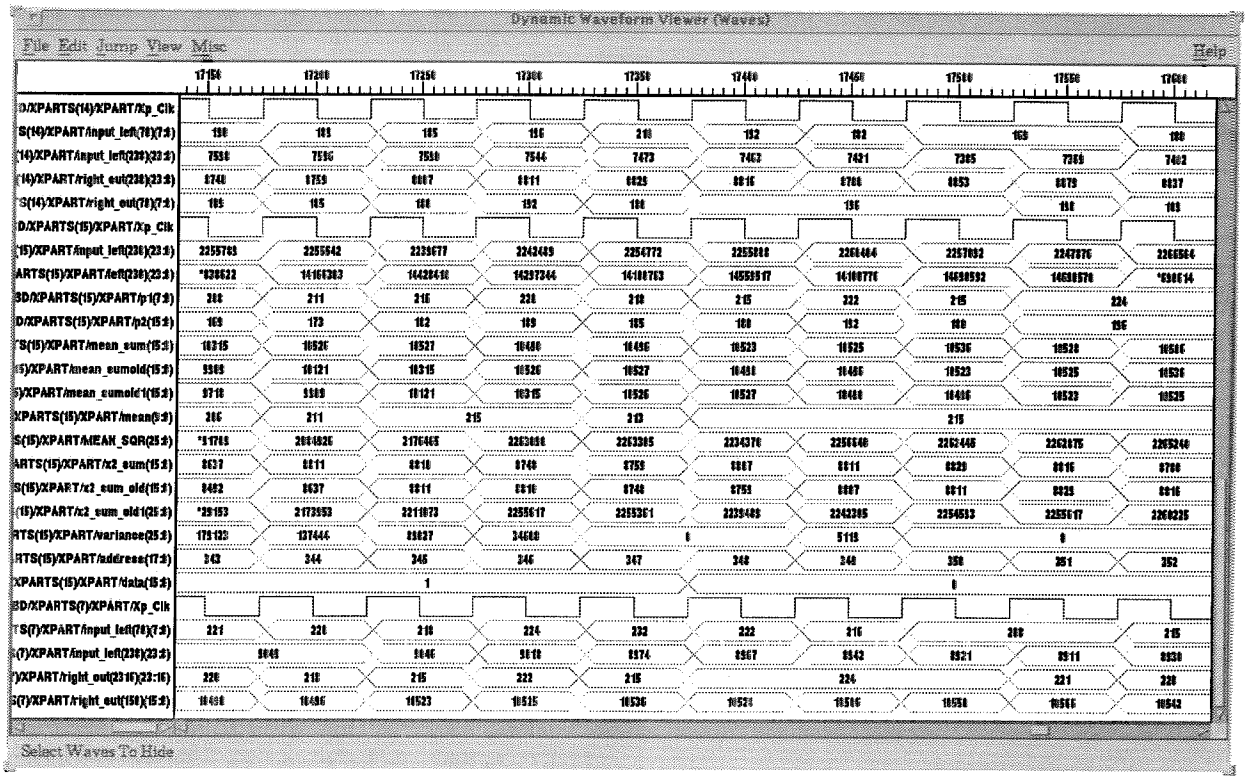
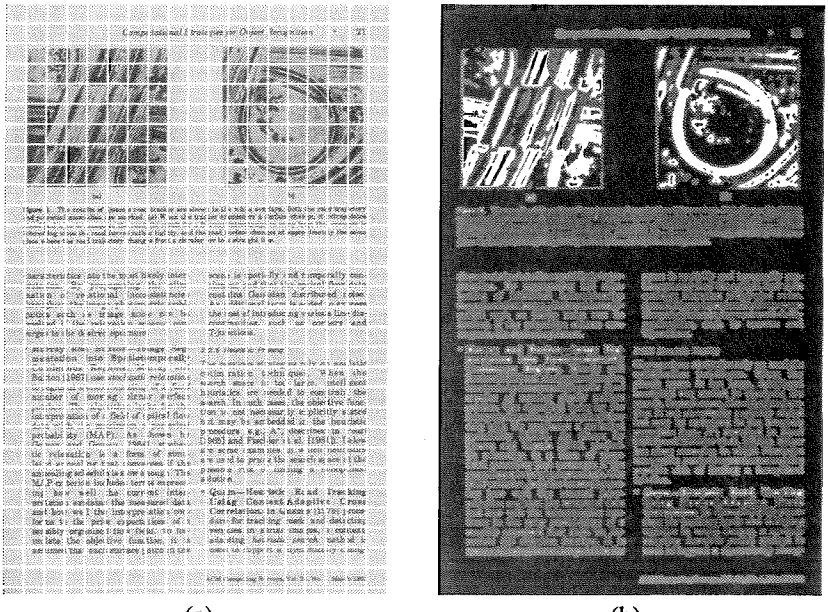


Figure 7. Simulation results of page layout segmentation on Splash 2.



(a) (b)

Figure 8. Page Layout Segmentation using Splash 2. (a) Input gray-level image with 32×32 pixel windows shown; (b) Result of the segmentation algorithm execution on Splash.