ACE: Automatic Centroid Extractor for Real Time Target Tracking

K. Cameron, S. Whitaker and J. Canaris
NASA Space Engineering Research Center for VLSI System Design
University of Idaho
Moscow, Idaho 83843

Abstract - **A high performance video image processor has been implemented which is capable of grouping contiguous pixels from a raster scan image into groups and then calculating centroid information for each object in a frame. The algorithm employed to group pixels is very efficient and is guaranteed to work properly for all convex shapes as well as most concave shapes. Processing speeds are adequate for real time processing of video images having a pixel rate of up to 20 million pixels per second. Pixels may be up to 8 bits wide. The processor is designed to interface directly to a transputer serial link communications channel with no additional hardware. The full custom VLSI processor was implemented in a 1.6μm CMOS process and measures 7200μm on a side.**

1 Introduction

ACE (Auto Centroid Extractor) groups contiguous pixels whose intensities are equal to or exceed a given threshold into separate objects and calculates their centroids. Proper assignment of contiguous pixels to their respective objects is guaranteed for all convex shapes and most concave shapes. Data for a $N \times M$ pixel image is processed in raster scan format with a maximum value of 1024 for N and M. ACE can process in excess of 500 objects per frame. It serially outputs the following information for each object:

$$\sum X_i I_i \quad \sum Y_i I_i \quad \sum I_i \quad \#\ of\ Pixels \tag{1}$$

where X_i and Y_i are the $x-$ and $y-$coordinates of the i_{th} pixel in an object respectively, and I_i is the intensity of the i_{th} pixel.

Threshold intensity levels are set as instructed by a Transputer via the serial link interface. Pixels whose intensities fall below the specified level are not recognized as object pixels. Masking of pixels is also performed according to an optional external RAM, which is controlled by the ACE chip. The ACE chip updates the masking pattern as instructed by a Transputer.

ACE also provides status information for each frame processed. This information includes a frame identification count, internal memory overflow flags, and an invalid shape detection flag.

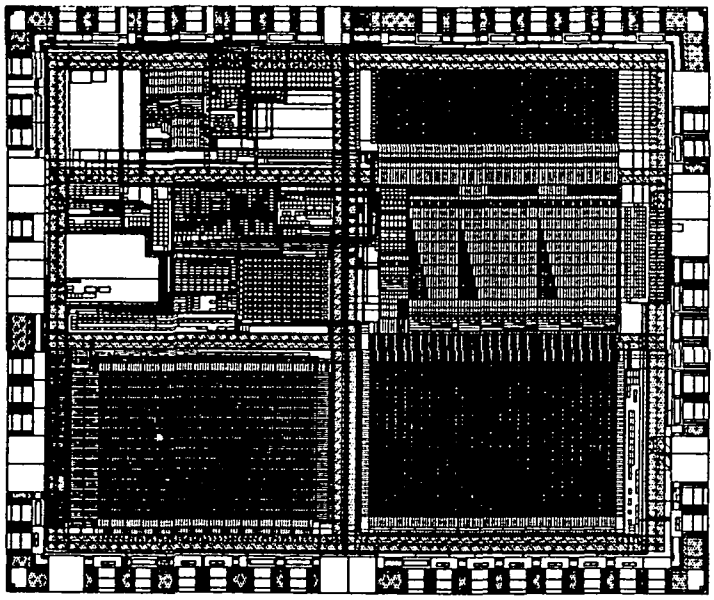


Figure 1: Auto Centroid Extractor

2 Architecture

2.1 Overview

The ACE processor may be divided into a number of subsections, each of which performs one or more of the basic operations necessary for the operation of the circuit. Figure 1 shows a plot of the ACE processor. Each of the major functional blocks in the processor are identified according to their function in Figure 2. Starting at the upper right hand corner, partial results of ongoing centroid calculations are stored in the *scratch ram*. The *calculator* section performs all of the actual arithmetic operations required. Upon the completion of an object, the results of the relevant centroid calculations are queued in the *output fifo*. These data sent out as requested by the host transputer via the *serial link*, after begin formatted by the *output processor*. The group numbers associated with the individual pixels of the previous scan line are supplied to the *image process controller* by the *group fifo*. *The image process controller* uses these group numbers in conjunction with the stream of incoming pixels from the current scan line to group the pixels into contiguous groups, and controls the *calculator* and the *scratch ram* accordingly. The *circle generator/ram interface* performs a number of related support functions. It writes circular masking regions to the optional external mask RAM in response to commands issued by the host transputer via the *serial link* interface. It also provides the masking and pixel level thresholding functions which are required during actual data processing.

Some of the major functional blocks comprising the processor are described below:

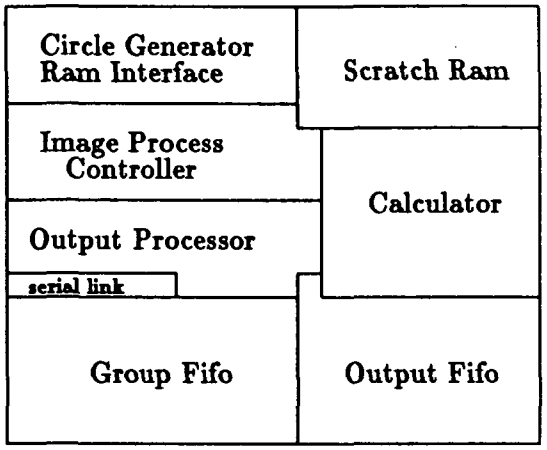


Figure 2: Image Processor Organization

2.2 Image Process Controller

The *image process controller* accepts commands from the *pixel pattern matcher* (see below) and pixel group data from the *group fifo*. Using this information, it controls the *calculator*, regulates the flow of data to and from the *scratch ram*, sends results to the *output fifo*, assigns group numbers, manages the free group storage list, detects and reports invalid objects, and arbitrates conflicting *scratch ram* read/write requests.

Scratch ram access conflicts are very simply handled. Due to the nature of the *trigger patterns* which initiate each data transfer, read requests are always separated by at least one unused access cycle. The same is true of write requests. All contentions can, therefore, be avoided by delaying the write by one cycle each time such a conflict occurs. Simultaneous read/write accesses to the same memory location are permitted, however, since very fast turn around of data through the *scratch ram* is occasionally required.

In addition to about 20 small (2-8 states), tightly interacting state machines, the controller also contains a small register stack, which is used primarily to track group numbers, and manage the *scratch ram* memory. Free (unused) group numbers are supplied by a free storage list that is maintained in the *scratch ram* itself. Group numbers are returned to the free list upon the termination of a pixel group. The controller was designed as a group of small state machines, as opposed to a monolithic controller, because of the quasi- independent, yet overlapping nature of the individual commands issued by the pixel pattern matcher (see below). The formulation of a single controller would have been very difficult and would probably have resulted in a combinatoric explosion of states.

2.3 Calculator

The *calculator* section consists of four computational units which transfer data to and from the *scratch ram* and send results to the *output fifo*. The physical organization of this section is shown in Figure 3. The leaf cells from which the calculator is constructed contain a bit slice of all necessary bussing. All signal paths are formed by abutting these

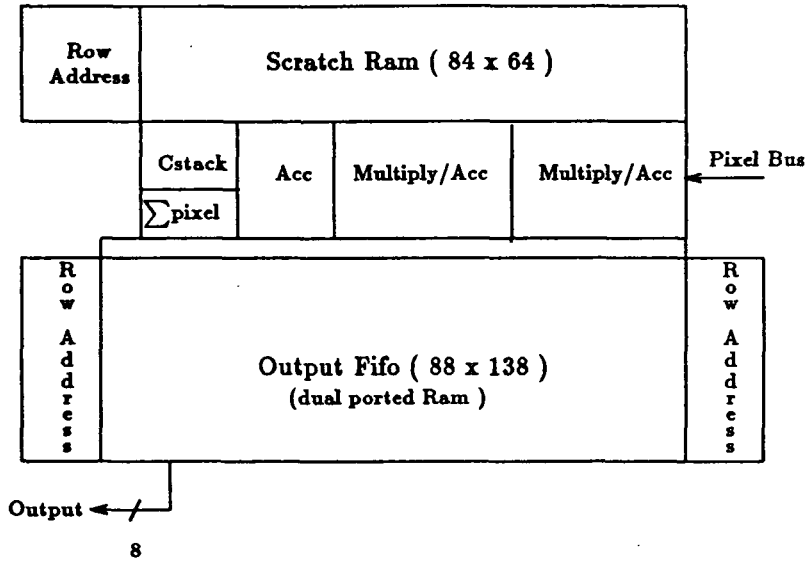


Figure 3: Computation Unit Architecture

cells. In addition, all leaf cells used to construct the calculator are pitch matched to a pair of the ram cells used to construct the *scratch ram* and the *output fifo*. This permitted the calculator section to be implemented as an extremely compact and regular structure.

The *calculator* consists two multiply/accumulators (mac), and two accumulators (acc). Each mac can performs an 8 by 10 bit multiply and two 28 bit accumulations every clock cycle. The pixel intensity acc can accumulate an 8 bit intensity and an 18 bit partial result from the *scratch ram* every clock cycle, and the pixel count acc accumulates a 1 bit pixel flag and 10 bit partial result each clock cycle. Each unit has a latency of two clock cycles.

The structure of the mac units is shown in Figure 4. The sum in and carry in inputs of the full adders at the top of the multiplier array are utilized to perform the required additions. The multiplier array is arranged in a standard configuration [5], with the sum and carry out signals feeding into separate sum and carry registers. The final addition of the carry and sum registers is performed by a Transmission-Gate Conditional-Sum Adder [4], because these adders are very fast and compact [3] when compared to other carry lookahead configurations [2]. The accumulators were created by re-arranging the cells constructed for the mac sections.

2.4 Circle Generator

The *circle generator* draws circular masking regions in the external mask RAM under the direction of the transputer. Each circle is specified in terms of its center point and radius squared. The *circle generator* uses finite difference equations (2) to calculate which pixels fall within the circles so specified, and either clears or sets the pixels accordingly.

$$R^2_{i+1} = R^2_i + \overline{2x} + 1 \qquad or, \qquad R^2_{i+1} = R^2_i + \overline{2y} + 1 \tag{2}$$

The hardware used to solve these equations is shown in Figure 5. It consists of a single

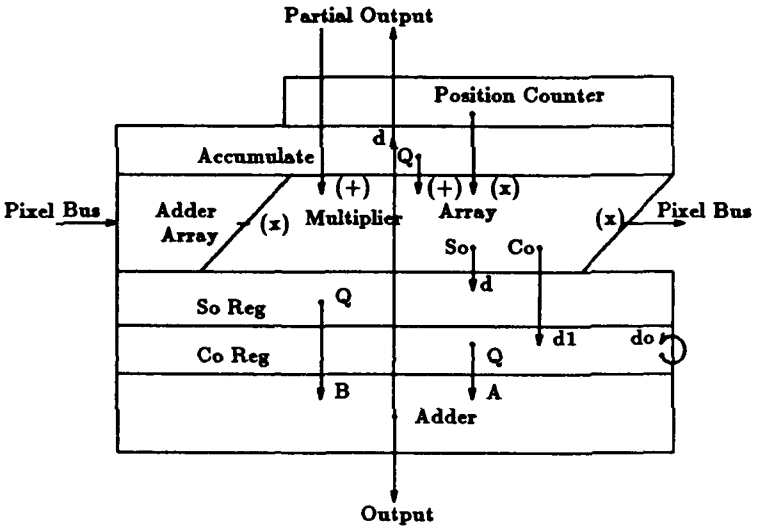
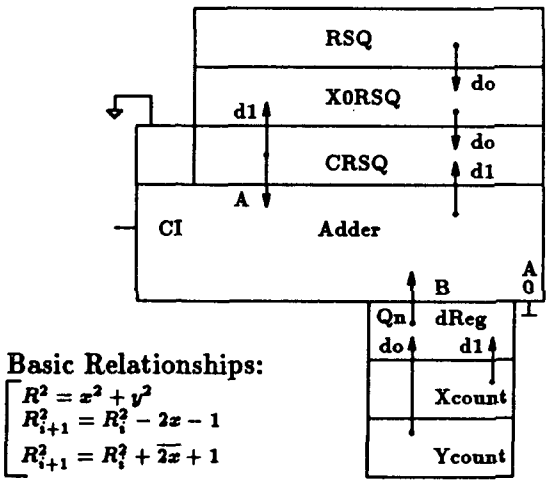


Figure 4: Multiply/Accumulator Architecture



Basic Relationships:

$$R^2 = x^2 + y^2$$
$$R^2_{i+1} = R^2_i - 2x - 1$$
$$R^2_{i+1} = R^2_i + \overline{2x} + 1$$

Figure 5: Circle Generator

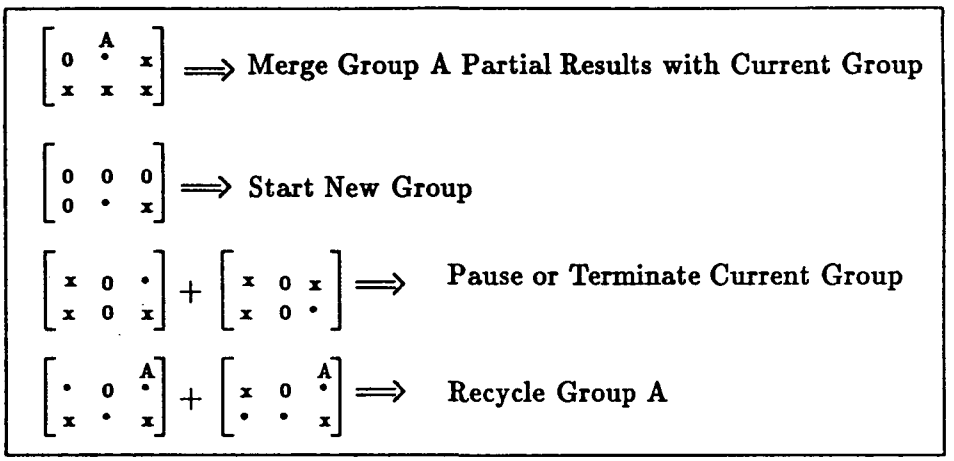


Figure 6: Primary Trigger Patterns

20 bit adder, two 20 bit registers and two 10 bit position counters. Cells designed for the *calculator* were used to implement this machine.

3 Pixel Grouper

3.1 Overview

Pixels entering ACE are first masked according to the information contained in the external mask RAM and then compared to a threshold value programmed from the host transputer. Pixels falling below the threshold are set to zero, the rest are processed as constituents of some group of pixels. Critical in this process is an efficient, readily implementable algorithm for assigning pixels to separate groups. Such an algorithm was developed for this processor. Though the algorithm used for ACE does not properly group pixels into arbitrarily complicated concave shapes, it does function correctly for all convex shapes, as well as most concave shapes. Complete generality was sacrificed in favor of an algorithm which can be easily implemented at high data rates. Subsequent implementations of ACE may utilize more general algorithms. The algorithm actually used is outlined in the below.

3.2 Algorithm

The algorithm used to group pixels utilizes information about the previous scan line, as well as the pattern of incoming pixels on the current scan line to form pixel groups. In particular, the controller scans the incoming pixels, together with the pixel patterns of the previous scan line for one of six distinct *trigger patterns*. Each *trigger pattern* contains six pixels, three from the previous scan line and three from the current scan line. Each time the incoming pixel pattern matches one of these *trigger patterns*, the controller initiates one of four basic operations. Figure 6 summarizes each of these *trigger patterns* and describes the operation initiated by each pattern.

The first pattern essentially indicates that, at every leading (leftmost) edge of a group in the previous scan line, the partial results calculated for that group are to be merged into the calculator as part of the current group (if one exists). Some preprocessing of the previous scan line information is performed to ensure that shapes consisting of multiple downward "fingers" will be merged into the current group only once. The second pattern causes a new group to be started if a pixel on the current line is not adjacent to any pixel on the previous scan line. It should be noted that it is entirely possible for subsequent occurrences of the first pattern to cause groups existing in the previous scan line to be merged into this new group.

Patterns three and four indicate that the current group should be either terminated (sent to the *output fifo*) or paused (sent to the *scratch ram*). These patterns are the ones that imply another group is about to be (re)started. If no pixels have been added to the current group on this scan line (or if the last scan line in the frame is being processed), it is assumed that the current shape does not extend into subsequent scan lines and it is terminated; otherwise, it is sent back to the *scratch ram* for subsequent processing.

The last set of patterns indicate that a group which existed in the previous scan line has been merged into another group. Its presents signals the controller to recycle the merged group into the free group list, so that it may be re-used.

This set of *trigger patterns* or rules is highly successful in grouping pixels into contiguous groups. Failures occur in two situations, however. First, it is possible to fool the machine into merging a group from the previous scan line into the current group more than once. This occurs with shapes having multiple downward "fingers" which are separated by different, non-contiguous groups. The second failure mode occurs when the machine is fooled into prematurely terminating a group. This also occurs only with shapes having multiple downward "fingers" separated by different, non-contiguous groups. If the first downward finger has no new pixels added to it in the present scan line, *trigger patterns* three and four will cause the group to be terminated even though pixels may be added from subsequent fingers.

Both defects in the basic grouping algorithm may be remedied. Zeroing out the contents of the *scratch ram* associated with a group each time it is read prevents the first failure mode. Waiting until the end of a scan line before terminating groups by sending them to the *output fifo* remedies the second failure mode. In the present implementation, however, this would also require the addition of a machine which keeps track of whether or not a given group had pixels added to it in the present scan line. It also complicates the *scratch ram* timing considerably.

4 Summary

Highly efficient pixel grouping algorithms were developed for ACE. The required calculations are performed in a specially designed architecture which minimizes signal line interconnect, and maximizes data throughput and computational efficiency. All control, pattern matching, memory management, data flow, input/output, and computational operations

are performed in parallel. These factors, combined with a custom designed layout, result in a very high performance centroid processor.

5 Future Development

A number of extensions are under consideration. Among these are a version with more bits per pixel, and an enhanced pixel grouper. A version which calculates higher moments, and allows the threshold level to be a function of position within the frame is also being considered.

6 Acknowledgments

The work reported herein was performed under the auspices of the U.S. Department of Energy for the Lawrence Livermore National Laboratory by the NASA VLSI Hardware Acceleration Center for Space Research.

References

[1] T. Axelrod, T. Tassinari, G. Barnes, K. Cameron, "A VLSI Centroid Extractor For Real-Time Target Tracking Applications," SPIE Conference, August 1989.

[2] R. Brent, H. Kung, "A Regular Layout for Parallel Adders," IEEE Transactions on Computers, Vol. C-31, March 1982, pp. 260-264.

[3] J. Canaris, K. Cameron, "A Comparison of Two Fast Binary Adder Configurations," NASA SERC 1990 Symposium on VLSI Design, University of Idaho, Moscow, Id, Jan. 1990, pp. 78-86.

[4] A. Rothermel, et al., "Realization of Transmission-Gate Conditional-Sum (TGCS) Adders with Low Latency Time," IEEE JSSC, Vol. 24, June 1989, pp. 558-561.

[5] N. Weste, K. Eshraghian, *Principles of CMOS VLSI Design*, Reading, Mass., Addison-Wesley, 1985, pp. 344-348.