CONF 900451--2

CONF-900451--2

DE90 008905

Design and Implementation of Moment Invariants for Pattern Recognition in VLSI

G. A. Armstrong
M. L. Simpson

Oak Ridge National Laboratory*
Oak Ridge, Tennessee 37831-6005

and

D. W. Bouldin
The University of Tennessee
Knoxville, Tennessee 37996

Paper to be presented to the
Technical Symposium on Optical Engineering and Photonics in Aerospace Sensing
Orlando, Florida

April 16-20, 1990

*Operated by Martin Marietta Energy Systems, Inc., for the U. S. Department of Energy under Contract No. DE-AC05-840R21400.

DISCLAIMER

Design and implementation of moment invariants for pattern recognition in VLSI

G. A. Armstrong M. L. Simpson D. W. Bouldin

Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831-6005

The University of Tennessee
Knoxville, Tennessee 37996

1. ABSTRACT

This paper describes the design of a very large scale integration (VLSI) application specific integrated circuit (ASIC) for use in pattern recognition. The pattern recognition scheme uses Hu[1] and Maitra's[2] algorithms for moment invariants. A prototype design was generated that resolved the long delay time of the multiplier by custom designing adder cells based on the Manchester carry chain. Use of the Manchester carry chain effectively incorporated the lookahead carry function into the adder cells. The prototype ASIC is currently being fabricated in 2.0-mm compiled simulator for metal oxide semiconductor (CMOS) technology (simulated at 20 MHz). The prototype consisted of a 4x8 multiplier and a 12-bit accumulator stage. The present ASIC design consists of a 9x26 multiplier (maximum propagation time of 50 ns) and a 48-bit accumulator stage. The final ASICs will be used in parallel at the board level to achieve the 56 MegaPixels/s [230 million operations per second (MOPs)] necessary to perform the moment invariant algorithms in real time on 512x512 pixel images with 256 grey scales.

2. INTRODUCTION

Pattern recognition involves processing large amounts of data at speeds from 10 to 20 MHz to attain real-time performance. To efficiently execute the pattern-recognition process, the images must be reduced to a small subset of the original data while maintaining a unique description of the image[3]. The moment invariant algorithms, as developed by Hu and Maitra, reduce the image to six descriptive constants that are invariant to changes in translation, rotation, scaling, contrast, and illumination. The algorithms require all combinations of the zero- through third-order moments for the image to be derived.

For the 256 grey-scale, 512x512 image to be processed at real time, the pixels must be processed at a stream rate of 7.86 megabytes per second (MBps). To derive the necessary moments, the ASICs will be required to perform 230 MOPs. Implementation of the moment derivation by discrete hardware would require a large amount of power and would put constraints on the possible architectures that the designer can select. The plethora of tools available for custom ASIC design, the capability to simulate the final designs, and the 120 MHz internal switching speeds of the CMOS devices make VLSI design extremely competitive with the development of discrete hardware.

3. MOMENT INVARIANTS

Moments are used to describe the distribution of a physical system with finite mass relative to a reference point in the system. The relationship is described by summing the products of each element in the system with its euclidean distance from the reference point. In physical systems, moments are used to describe the moment of inertia of a physical mass relative to one of the axes in the system. Inertia is the resistance of the rotational system to movement in the same way that the mass is the resistance of an object in a linear system to movement.

Moments also can be used to describe or represent digitized images. The mass of the elements in the physical system is replaced by the light intensity of a point in the image captured by the camera and digitization equipment. In the digitized image, the distance to the reference point must now be measured in two- rather than three-dimensional space. The use of moments to represent images allows a representation of the image to be stored in several bytes rather than the 256 Kbytes required to store a 256 grey-scale, 512x512 pixel image. In addition to the computer resources saved by data compression, the moment of the image allows the representation to be compared quickly with other images for pattern recognition applications.

Obvious problems develop when the moments are used to describe images uniquely. The physical system exists in three dimensions while the image is contained in two dimensions. The distance to the center of gravity of the physical system remains the same even under changes in orientation. This is not true for images. The image of the physical system changes because of changes in the orientation of the physical system with respect to the camera. The resultant image has distance

measurements reduced to one plane, which reduces the ability to accurately include depth in the distance measurements. Changes in orientation and proximity to the camera system hide the true moment information about the physical system. In addition, the mass of each element in the physical system has been replaced by the intensity of a pixel. The intensity of the pixel is not an inherent property of the image, as is the mass of the physical system. The intensity varies with the background lighting and the location of light sources used to illuminate the object. As a result, changes in orientation, scaling, and lighting have direct effects on the digitized image and can prevent moments from being used to generate unique representations for use in pattern recognition.

4. MOMENT PROPERTIES OF IMAGES

This section introduces the equations used to develop the moment invariant algorithms. First, a definition about moments must be established and givens about moments must be stated. The definition of a second-order, ordinary moment is given in equation (1).

$$m_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p y^q f(x,y)\, dxdy\,, \tag{1}$$

where f(x,y) is the density distribution (or intensity) that must be piecewise continuous and has nonzero values in a finite part of the xy plane. The uniqueness theorem states that if f(x,y) is piecewise continuous and has nonzero values in a finite part of the xy plane, then moments of all orders exist and the moment sequence (m_{pq}) is uniquely determined by f(x,y).

Uniqueness theorem: The double moment sequence (m_{pq}) is uniquely determined by f(x,y); conversely, f(x,y) is uniquely determined by (m_{pq}).

The digitized image that defines f(x,y) is composed of a finite number of elements or pixels that would be zero only if the image was captured in total darkness. As a result, orders of all the moments of the digital images exist. Once the moments have been determined to exist, they must be made invariant to changes.

5. MOMENT INVARIANT ALGORITHMS

Moment Invariants is a technique that attempts to make moments invariant to changes in translation, rotation, scaling, illumination, and contrast. Two moment invariant algorithms are described in this chapter. The two algorithms were developed by Hu and Maitra. Hu developed an algorithm that takes all second- and third-order moments of the image and produces seven constants that are invariant to translation, rotation, and scaling. Maitra's algorithm offers an improvement to Hu's algorithm by taking the seven constants and, in putting them in to six more equations, generates six constants that are also invariant to changes in contrast and illumination.

Moments can be made invariant to translation by considering the moment with respect to the centroid of the image. Central moments are invariant to translation changes by definition. The central moment of order (p+q) for a piecewise continuous function f(x,y) is defined in equation (2).

$$\mu_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (x-\bar{x})^p (y-\bar{y})^q f(x,y)\, dxdy \tag{2}$$

The centroids $\bar{x}$ and $\bar{y}$ are defined as

$$\bar{x} = m_{10}/m_{00} \quad \bar{y} = m_{01}/m_{00} \tag{3}$$

For a digital image, the double integration is expressed as double summation since the pixels are discrete points instead of piecewise continuous surfaces. The discrete central moment of order (p+q) is defined as

$$\mu_{pq} = \sum_{x} \sum_{y} (x-\bar{x})^p (y-\bar{y})^q f(x,y)\, dxdy \tag{4}$$

However, all seven moments of orders 2 and 3 are needed for Hu's algorithm.

Even though Hu's algorithm does not require directly the central moments of orders 0 and 1, the determination of the centroid requires the ordinary moments of orders 0 and 1. The 0 order ordinary moment also is used to normalize the centralized moments. As seen in equations (5) through (14) below, the central moments of order 1 are zero and of little use to the algorithm. The next seven moments and the moment of order 0 are used by Hu's algorithm. Hu takes advantage of the properties of moments to produce his results. The first property is that moments are linear operators; therefore, by the superposition theorem, the moment of the image is the sum of the image's pixel moments. Secondly, a moment in one coordinate system can be mapped to another coordinate system by other linear operators. These are the two properties on which moment invariants are built.

Ordinary moments are determined with respect to the origin of the coordinate system. Ordinary moments can be mapped by linear operators into a new coordinate system where the moments in the new coordinate system describe centralized moments. The central moments can be expressed in terms of ordinary moments as shown in equations (5) through (14).

$$\mu_{00} = m_{00} , \tag{5}$$

$$\mu_{10} = 0 , \tag{6}$$

$$\mu_{01} = 0 , \tag{7}$$

$$\mu_{20} = m_{20} - \bar{x} m_{10} , \tag{8}$$

$$\mu_{02} = m_{02} - \bar{y} m_{01} , \tag{9}$$

$$\mu_{11} = m_{11} - \bar{y} m_{10} , \tag{10}$$

$$\mu_{30} = m_{30} - 3\bar{x}\, m_{20} + 2\bar{x}^2 m_{10} , \tag{11}$$

$$\mu_{03} = m_{03} - 3\bar{y}\, m_{02} + 2\bar{y}^2 m_{01} , \tag{12}$$

$$\mu_{12} = m_{12} - 2\bar{y}\, m_{11} - \bar{x} m_{02} + 2\bar{y}^2 m_{10} , \tag{13}$$

$$\mu_{21} = m_{21} - 2\bar{x}\, m_{11} - \bar{y} m_{20} + 2\bar{x}^2 m_{01} . \tag{14}$$

The central moments, as previously mentioned, are invariant to changes in translation. This demonstrates how a set of moments can be mapped into a new coordinate system by linear operators causing the resulting moments to be invariant to changes in translation and orientation in the plane of the image. Another linear operator can be used to normalize the moments, thereby, making the resultant moments also invariant to changes in parallel projection or scaling. The normalized central moments, denoted by npq, are defined as:

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}} ; \tag{15}$$

$$\gamma = \frac{p+q}{2} + 1 ; \tag{16}$$

for

$$p + q = 2,3, \ldots . \tag{17}$$

As a result, the moment invariants are made invariant to scale changes by the normalization of the central moments.

5.1 Hu Algorithm

Hu's moment invariant algorithm uses the normalized central moments of orders 2 and 3 as input to seven algebraic equations [(18) through (24)] as defined:

$$\phi_1 = \eta_{20} + \eta_{02}; \tag{18}$$

$$\phi_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2; \tag{19}$$

$$\phi_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2; \tag{20}$$

$$\phi_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2; \tag{21}$$

$$\phi_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})\left[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2\right] + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})\left[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right]; \tag{22}$$

$$\phi_6 = (\eta_{20} - \eta_{02})\left[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}); \tag{23}$$

and

$$\phi_7 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})\left[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2\right] - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})\left[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right]. \tag{24}$$

This method produces seven constants for use in pattern recognition that are invariant to changes in orientation, rotation, position, size, and parallel projection (focus).

5.2 Maitra Algorithm

Equations (25) and (26) represent the changes produced when the image is translated, rotated, scaled, and subjected to a change in contrast. With the moment invariants described thus far, the results for the two images $f_1(x,y)$ and $f_2(x,y)$ should be the same under the following conditions:

$$f_1(x, y) = k f_2(x', y'), \tag{25}$$

and

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = c \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a \\ b \end{bmatrix}. \tag{26}$$

The rotation is described by the angle ϕ, (a,b) describes the translation, c produces a change in scale, and k produces the change in illumination and contrast. The second equation is used to map points in the first coordinate system into a new coordinate system in which the origin has been translated by (a,b), rotated by theta, and scaled such that the points are farther apart for c >1 and closer together for c <1.

Without the normalization procedure shown previously, the relationship between the invariant moments for the two images can be shown by equations (27) through (33)

$$\phi_1(1) = \frac{k}{c^4}\phi_2(1); \tag{27}$$

$$\phi_1(2) = \frac{k^2}{c^8}\phi_2(2); \tag{28}$$

$$\phi_1(3) = \frac{k^2}{c^{10}}\phi_2(3); \tag{29}$$

$$\phi_1(4) = \frac{k^2}{c^{10}}\phi_2(4); \tag{30}$$

$$\phi_1(5) = \frac{k^4}{c20}\phi_2(5); \tag{31}$$

$$\phi_1(6) = \frac{k^3}{c14}\phi_2(6); \tag{32}$$

$$\phi_1(7) = \frac{k^4}{c20}\phi_2(7). \tag{33}$$

If the moment invariants were figured using the normalized central moments, the variations in the results caused by the scaling term c would be eliminated. This leaves the variation because of the contrast and illumination changes made by the factor k. Maitra proposes using the following six equations, (34) through (39), that have eliminated the k term and therefore make the resulting six equations invariant to changes in illumination and contrast:

$$\beta(1) = \frac{\sqrt{\phi(2)}}{\phi(1)}; \tag{34}$$

$$\beta(2) = \frac{\phi(3)\mu_{00}}{\phi(2)\phi(1)}; \tag{35}$$

$$\beta(3) = \frac{\phi(4)}{\phi(3)}; \tag{36}$$

$$\beta(4) = \frac{\sqrt{\phi(5)}}{\phi(4)}; \tag{37}$$

$$\beta(5) = \frac{\phi(6)}{\phi(4)\phi(1)}; \tag{38}$$

and

$$\beta(6) = \frac{\phi(7)}{\phi(5)}. \tag{39}$$

Moment invariant algorithms can be combined with pattern recognition algorithms, in digital-image processing applications to compress and represent images. Moments that are normally used to uniquely describe a physical,three-dimensional system based on inherent properties, such as the mass of the elements in the system, can be used to describe digital images that represent the physical system captured in a two-dimensional image. Moment invariant algorithms developed by Hu and Maitra can make the resultant-image descriptors invariant to changes in orientation, scaling , contrast, and illumination. The numerous computations required to produce the descriptors is due to the requirement of nine moments calculations, ranging from order 0 to order 3. The time required to determine the invariant coefficients can be reduced by off-loading the computation of the moments from the computer to custom VLSI ASICs. With the use of the VLSI design, the moment invariant algorithm can potentially be determined in real time.

6. Image-Processing Architectures

VLSI designs are incorporated into a system when the solution by conventional methods is time-consuming and when the algorithm used to solve the problem can be broken down into simple modules that are implemented with a large number of repetitions. As a result, most VLSI architectures use parallelism through arrays and pipelining to optimize the performance.[4] The typical architectures used in image processing are the one- and two-dimensional arrays. The one-dimensional linear array and the two-dimensional square and hexagonal arrays are mesh-connected processor arrays. The information paths between these arrays are simple and regular and, therefore, do not require long development times.

Another commonly used architecture is the tree structure. The tree architecture is more complex than the array architectures. For a tree array to be efficient for a given application, the problem to be implemented must have a growth pattern that increases exponentially. The node processors in the tree must be able to break down the problem into two or more parts. The communications bandwidth of this type of architecture is concentrated at the lowest levels. If a given application requires

more communication at the top level than is provided by the one communication path to the root node of the tree, the process will not be able to take advantage of the capabilities of this architecture. The tree structure is particularly useful in applications requiring a search process. The Wallace[5] tree adder is an example of a tree architecture used in multiplier circuits to reduce the propagation time of a long adder bank.

Systolic (or semi-systolic) arrays are a popular architecture used in image processing. Systolic arrays attempt to use data in such a manner to maximize computational efficiency while minimizing throughput. This is achieved by ordering the data so that they are used effectively by each cell that they pass through. In a typical application, a systolic array will be able to produce one piece of output data for every piece of input data clocked into the array. This also stipulates that a systolic-based design must be synchronous rather than transparent or asynchronous. For the multiply function, this means that for every pair of operands entered into the systolic-based two-dimensional multiplier array, one product would be produced. The partial products for a series of given multiplications would ripple through the systolic array in the same manner that a series of waves would ripple through a pond. This method heavily uses the principle of pipelining to allow several multiplications to occur in parallel in the array. As a result, a high level of computational efficiency can be attained with a low I/O bandwidth. To allow the cells to be utilized in this way, the data have to be preprocessed before they can be input into the systolic array and post-processed on the way out. The internal pipelining, preprocessing, and postprocessing will require a large number of register cells. In addition, a pipelined multiplication array will require that considerable attention be given to the design of a clock distribution network to ensure that clock skew does not impair the operation.

7. Multiplication Architectures

High-speed multiplication is required for all digital-signal processing applications. Typical image-processing applications require multiplication propagation times to be on the order of 50 ns to process the data in real time. As a result, the multiplication function is the critical design function for many digital signal processing applications.

At present, several methods can be utilized to perform the multiplication function. These methods include the use of the modified Booth's algorithm, Wallace trees, semi-systolic arrays, and two-dimensional arrays. The methods can be further classified by whether they incorporate carry-save adders (CSAs) or carry-propagate adders (CPAs) or adder-accumulator subtractors (ACS). In addition, multipliers typically use a carry-lookahead adder (CLA) stage on the last bank of adders to reduce the effect of the carry-propagation time in the last stage.

Tables 1-4 illustrate some of the parameters that can be used to compare some of the multiplication techniques documented in the literature.

Table 1. 4x4, 2.0 mm NMOS Modified Booth Multiplier[6]

Technology	2.0 mm NMOS
Chip size	1300x1050 mm^2
Transistor count	250
Multiplication time	16 ns (62.5 MHz)
Power dissipation	31.5 mw
Power supply	5.0 V
CLA	None
Adder	ACS
Clock scheme	N/A
Clock phase	N/A
Multiplication	Signed magnitude

Table 2. 16x16, 0.6 mm CMOS Modified Booth Multiplier[7]

Technology	Twin tub 0.6 mm CMOS
Chip size	2400x3400 mm^2
Transistor count	11000
Multiplication time	7.4 ns (135 MHz)
Power dissipation	400 mw (includes pads)
Power supply	3.3 V
CLA	Redundant CLAs
Adder	CSA
Clock scheme	N/A
Clock phase	Three-phase
Multiplication	Signed magnitude

Table 3. 8x8, 1.0 mm NMOS Semi-systolic Multiplier[8]

Technology	1.0 mm NMOS
Chip size	2100x2100 mm^2
Transistor count	5480
Multiplication time	3.0 ns (330 MHz) (600 MHz liquid nitrogen)
Power dissipation	1500 mw
Power supply	3.0 V
CLA	Diagonal array of half adders
Adder	CSA
Clock scheme	Super buffer drivers
Clock phase	Two-phase
Multiplication	Magnitude

Table 4. 8x8, 2.5 mm CMOS Semi-systolic Multiplier[9]

Technology	2.5 mm CMOS
Chip size	N/A
Transistor count	N/A
Multiplication time	14.3 ns (70 MHz)
Power dissipation	250 mw (includes pads)
Power supply	N/A
CLA	Diagonal array of half adders
Adder	CSA
Clock scheme	Balanced distributed network
Clock phase	Single-phase
Multiplication	Magnitude

The most straightforward approach to the multiplier design is to use a two-dimensional array. Each element in the array is composed of a full adder cell, the exact composition of which does not have to be defined at this point. This array has two input data streams and attempts to make full use of each data item as it passes through the cells. The connections implement a CPA approach. The effect of the long propagate time for the carry can be reduced by the addition of CLA logic that can be added to each row of adders at the expense of an increase in area (typically 60%) and the loss of regularity because of the design and integration of the CLA function.[10]

One typical approach to reducing the long propagation times associated with multiplier designs based on simple arrays makes extensive use of pipelining to create a systolic-like architecture for multiplication.[8,9] The multiplier arrays typically use CSAs and CLA functions to reduce the time required to compute each partial product of a heavily pipelined array. Pipelined multiplier arrays can use N additional stages (making a NxN multiplier require 2N stages), with each stage having a maximum N nodes of half adders to eliminate the use of a CPA bank (therefore, the need for a CLA function) for the last stage of the multiplier. This approach reduces the maximum delay to that of a full adder cell and is a very regular design at the cost of approximately two times the increase in area over a similar approach using a CPA for the last bank of adders. Examples of this approach are illustrated in Tables 1-4. Note that this technique is critically dependent upon the clocking scheme used to synchronously clock all the pipes.

Another popular multiplier architecture approach is the modified Booth's algorithm.[6] This technique is typically implemented with multiple pipeline stages and can reduce the number of adders to approximately half those required for a simple multiplier array approach. A NxN-bit multiplier requires (N+2)/2 stages (with one adder/subtractor per stage). This approach lacks the regularity of the simple multiplier approach and, as a result, requires a long time to design and develop.

The moment generator ASIC presented in this paper is based on an asynchronous simple multiplier array that requires no clocking scheme. The TTL I/O pads of the MOSIS Tiny Chip limits the speed of parallel data transfer into the ASIC to 10 to 20 MHz. This translates to 100 to 50 ns per pipelined function in the ASIC. By implementing the multiplier as an asynchronous design, the clock timing and skew problems of synchronous pipelined designs are bypassed. The disadvantage becomes the lengthy time required to implement a large multiplication operation. This requires that the adder cells used in the multiplier array minimize the propagation time of data through the cell and particular attention be given to minimize the carry propagation effects through the adder banks.

The data path design of the multiplier and summation units at the ASIC level allow the ASICs to be cascaded at the board level to derive the higher-order moments. The ASIC is designed such that a first-order moment calculation or a multiplication with no summation can be performed on each ASIC. This would allow the ASICs to be cascaded together to perform the second- and third-order moment calculations necessary for the moment invariant algorithms. The dual function capability of the ASIC would allow the ASICs to be arrayed where each row in the array would generate a different order moment of the image.

The CPA implementation, based on the two-dimensional hexagonal array, was selected. The CPA was chosen over the CSA because of its capability to read the actual intermediate result in each intermediate pipe. This greatly simplifies simulation and testing of the chip because the data going into and coming out of the scan path will not have to be encoded and decoded.

8. Circuit Design and Layout

Figure 2 depicts the ASIC layout of the moment generator for the final ASIC design. More effort was given to reducing the area and propagation times for the final design than was given to the custom cells during the initial prototype design. The results are summarized in the table below. The decrease in propagation time was due to the removal of the buffers used to restore signals coming into the cell and leaving the cell. This was originally done to prevent crosstalk on outgoing signals from entering the cell and potentially causing intermittent errors and to restore signals coming into the cell that may have traveled a relatively long distance. After determining that none of the signals traveled more than 100 lambda (lambda = 2mm) and that the routes between cells crosses few transitioning lines the buffers were dropped from the custom cells. The selection of a different exclusive-or (EXOR) design contributed to the reduction of more than 3 ns from the critical path. The new exclusive-or design is shown in Figure 3. The exclusive-or gate also allowed the size of the custom chip as well as its internal complexity to be reduced. The custom cell sizes were further reduced by using polysilicon as a third layer of interconnect and routing. Even though the delay due to the use of polysilicon is approximately a factor of 100 greater than that due to metal connections the delay is still negligible until the length goes over 200 lambda.

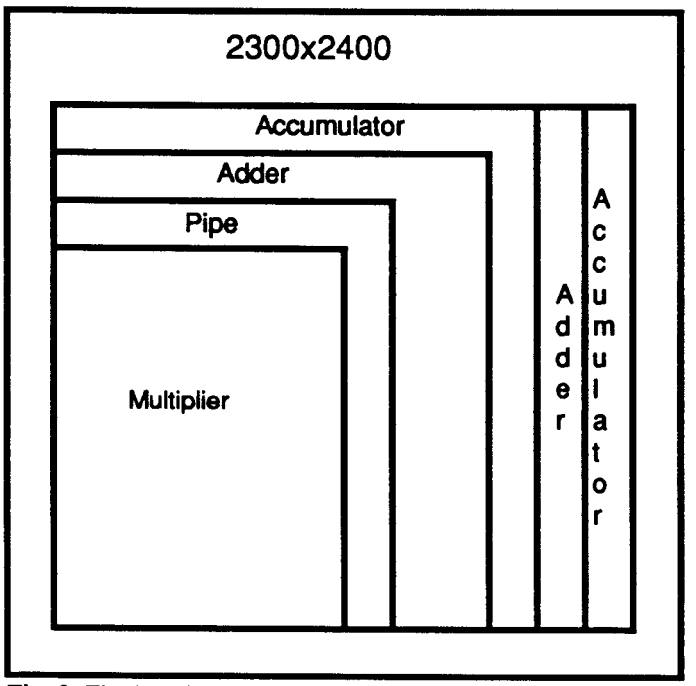

Fig. 2. Final design for ASIC layout.

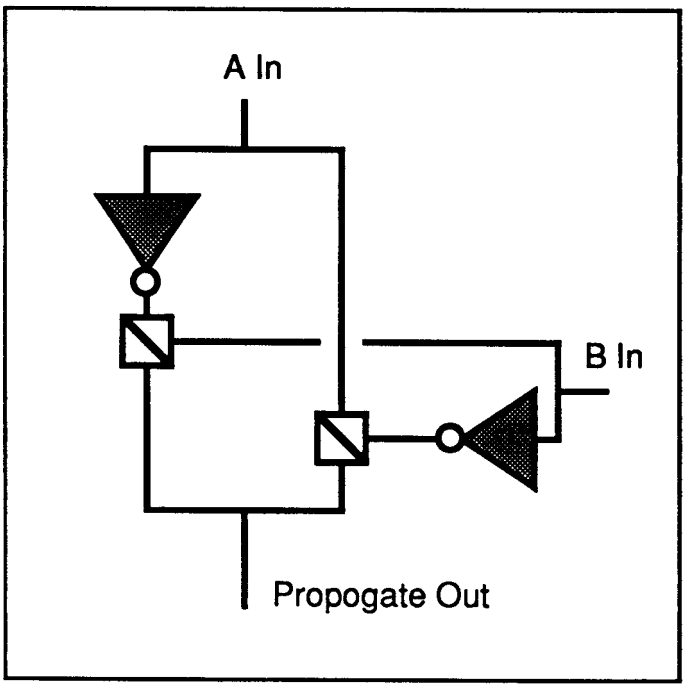

Fig. 3. Exclusive-OR Gate used in the adder cell

In the prototype design, the clocking scheme was single-phase complimentary. This was chosen because of the absence of sequential state machines that are prone to "race conditions" when using a single-phase clock. The final design incorporated a nonoverlapping two-phase clock to prevent the possibility of long, rise-and-fall times on the clock phases from producing erroneous results.

Because of the number of nodes in the final design, Pspice was not able to simulate the design. Pspice was used to simulate the individual cells and test rows and columns of the adders and accumulators. The results from these simulations were used to determine the critical paths based on the worst case propagation times. A new version of esim has been released that allows the design to be verified logically. The previous version of esim could not simulate the (EXOR) gate so Pspice was used to do all of the simulation for the prototype design. The simulation of the prototype moment generator with stray capacitances took approximately one week for each run. Running Pspice on the 9x26 multiplier would have taken weeks even without the stray capacitances.

Table 5. Speed and Size Comparisons

Prototype adder cell size	132x212 (47 transistors)
Final adder cell size	68x175 (33 transistors)
Prototype storage cell size	139x200 (34 transistors)
Final storage cell size	68x68 (12 transistors)
Prototype adder cell tp	9.5 ns
Final adder tp	4.0 ns
Prototype storage cell tp	2.0 ns
Final Storage Cell tp	1.0 ns

Table 6. Power Comparisons @ 20 MHz

Prototype adder cell mw	2.4
Final adder cell mw	1.6
Prototype storage cell mw	1.7
Final storage cell mw	0.6
Prototype ASIC core mw	140 (43 adders ,22 regs)
Final ASIC core mw	508 (282 adders, 96 regs)

Note that the power estimates do not include the power drawn by the input and output pads.

CLAs can be used to decrease the time required for the carry signal to propagate through the adder. This scheme involves computing carries for several sets of bits rather than just one pair at a time. The size of the grouping can be optimized for the application but usually is considered optimum for groups of four, especially if buffers are not included in the carry path. This approach has the disadvantage of adding a significant amount of circuitry to the design. The adder cells can be designed with a method similar to that used to implement the lookahead adder but on a two-bit basis rather than four or more. This is the basis for the Manchester carry chain conditions.[11]

9. Manchester Carry Conditions

Manchester carry conditions concentrate on minimizing the propagation time through the adder cells by studying the adder function solely from the carry signal viewpoint. Given two words, A and B, that must be added together to produce a sum, any two-bit adder cell within the adder bank can have a carry-in signal or a carry-out signal. The conditions of the input

operand bits A_i and B_i can have effects on the carry-out signal that are completely independent of the carry-in signal. Equations (40) through (42) summarize these conditions.

$$K_i = \overline{A_i B_i} \text{ (Kill Condition) ;} \tag{40}$$

$$G_i = A_i B_i \text{ (Generate Condition) ;} \tag{41}$$

and

$$P_i = A_i \oplus B_i \text{ (Propagate Condition) .} \tag{42}$$

The summation is expressed as the result of the (EXOR) of the two operand bits A_i and B_i along with the carry-in signal in an analogous manner to the way summation is performed in the full adder.

$$S_i = P_i \oplus C_{i-1} \text{ (Summation) .} \tag{43}$$

10. Performance and Test Results

The worst case propagation delay through the multiplier can be calculated by multiplying the carry propagation time delay of 0.1 ns by 26 (for 26 columns), and adding that to the product of the SumIn to SumOutBuf propagation delay of 5 ns multiplied by 8 (for 8 rows). This makes the worst case propagation delay through the multiplier 42.6 ns. As a result, the moment generator with the asynchronous multiplier can process data through its inputs and internal pipes at a maximum clock rate of 20 MHz. With the TTL I/O pads of the ASIC, the input data rates will be limited to 10 to 20 MHz. If pipes were placed between each bank of adders in the multiplier array the clock speed of the moment generator would reach 200 MHz. This would significantly increase the size of the ASIC core not only because of the extra 206 register cells used to create the pipe but also the extra registers to allow the correct multiplication bit to be applied to each bank of adders during the multiplication. There would also need to be a set of registers to hold the data bits until the complete word rippled through the entire multiplier. A significant design effort would have to be placed on designing a balanced clock network to ensure that the multiplier worked consistently without interference from clock skew.

11. Conclusions

Table 7. Key Features

Clock phase	Two-phase
Clock scheme	Superbuffers
CLA	None
Adder	Custom CPA
Transistors	10,485
Area	2300x2400-mm^2
I/O pins	132 I/O pins mm CMOS
Speed	Simulated 200 MHz
Power	Simulated 508 mw @ 20 MHz
(fully pipelined)	Simulated 6480 mw @ 200 MHz

By the time 132 I/O pads are added to the layout area, the design will be significantly beyond the size of the 2300x3400 mm MOSIS platform. This design requires the use of the 4600x6800 mm platform, which will leave a significant amount of room left over. The extra room can be used to add two additional multipliers, which would reduce each moment calculation from three ASICs to one ASIC. This would reduce the chip count and the total number of I/O pads required to do the nine moments for Hu and Maitra's algorithms and therefore reduce the total power consumed by the board. By combining the multiplications and summations necessary to handle the 56 Megapixel/s stream into one ASIC for each moment only, nine ASICs would be required to perform the necessary 230 MOPs of computations.

By combining the performance of custom VLSI special-purpose processors with pattern recognition algorithms, such as moment invariants used in image processing, high-performance computational engines that can process digitized images in real time become possible. This paper describes the design of a prototype ASIC used to implement ordinary and centralized moment calculations used for moment invariant algorithms for pattern recognition applications. The determination of a suitable architecture and the design and simulation of the cells used to build the basic functions through the simulation of the final moment generator ASIC have been presented.

12. Future Work

The moment generator ASIC could also contain integer division and subtraction functions that would allow the entire moment calculation to be done in VLSI. At this point the moment generator would be approaching more of a DSP capable of handling many different tasks. The moment generator function could also be combined by sorting to construct a database of compressed images to allow previously identified images to be quickly recognized.

13. References

1. M. K. Hu, "Visual Pattern Recognition by Moment Invariants," *IRE Transactions on Information Theory*, pp. 179-187, February 1962.

2. S. Maitra, "Moment Invariants," *Proceedings of the IEEE*, Vol. 67, No. 4, April 1979.

3. M. L. Simpson, *Moment Invariants for Automated Inspection of Printed Material*, submitted for publication, Instrumentation and Controls Division, Oak Ridge National Laboratory, 1989.

4. King-sun Fu, *VLSI for Pattern Recognition and Image Processing*, Spring Series in Information Sciences, New York, Tokyo, Springer-Verlag Berlin Heidelberg, 1984.

5. M. J. M. Pelgrom, H. E. J. Wulms, P. Stokker Van Der, and R. A. Bergamaschi, "FEBRIS: A Chip for Pattern Recognition," *IEEE Journal of Solid-State Circuits*, VOL. SC-22, No. 3, June 1987.

6. N. R. Shanbhag, and P. Juneja, "Parallel Implementation of a 4x4-bit Multiplier Using Modified Booth's Algorithm," *IEEE Journal of Solid-State Circuits*, vol. 23, no. 4, August 1988.

7. Yukihio Oowaki, and Kenji Numata, "A Sub-10-ns 16x16 Multiplier Using 0.6-mm CMOS Technology," *IEEE Journal of Solid-State Circuits*, vol. 5, no. 4, October 1987.

8. T. G. Noll, et al, "A Pipelined 330-MHz Multiplier," *IEEE Journal of Solid-State Circuits*, vol. SC-21, no. 3, June 1986.

9. M. Hatamian and G. L. Cash, "A 70-MHz 8-bit X 8-bit Parallel Pipelined Multiplier in 2.5-mm CMOS," *IEEE Journal of Solid-State Circuits*, vol. SC-21, no. 4, August 1986.

10. O. L. MacSorley, "High-Speed Arithmetic in Binary Computers," *Proceedings of the IRE*, pp. 67-91, January 1961.

11. L. A. Glasser, and D. W. Dobberpuhl, *The Design and Analysis of VLSI Circuits*, Addison-Wesley, Mass., 1985.