VLSI Architectures for Geometrical Mapping Problems in High-Definition Image Processing

K. Kim
Superconducting Super Collider Lab.
2550 Beckleymeade Avenue
Dallas, TX 75237

J. Lee
Department of Electrical Engineering
University of Houston
Houston, TX 77204-4793

Abstract- **This paper exploits a VLSI architecture for geometrical mapping address computation. The geometric transformation is reviewed under the field of plane projective geometry, which evokes a set of basic transformations to be implemented for the general image processing. The homogeneous and 2-Dimensional cartesian coordinates are employed to represent the transformations, each of which is implemented via an augmented CORDIC as a processing element. A specific scheme for a processor, utilizing fully-pipelining at the macro-level, parallel constant-factor-redundant arithmetic and fully-pipelining at the micro-level, is assessed to produce a single chip VLSI for the HDTV applications under the current state-of-art MOS technology.**

1 Introduction

Geometrical transformations are widely discussed in the field of digital image processing such as high-definition television(HDTV), image recognition, interactive computer graphics and vision processing [1,2,3]. The primary interest of these transformations is to project an image in a different domain, to extract additional signal conveying the information of the image. Moreover, it affords value-added images over the conventional displaying via the high resolution, definition, and flexible framing. Consequently, a geometrical mapping processor is about to appear to support a real-time processing. In recent years, several geometrical mapping processing modules have been developed and applied successfully for an appropriate application, They are implemented either by popular graphics package or application software accompanying an acceleration box [5], or a VLSI Processor [6]. We are interested in a VLSI implementation of a processor to realize a real-time speed for TV image processing, with a sufficient set of transformations to make a value-added display.

It has been known that two barriers have existed toward the development of such a processor. The first is the lack of a sufficiently high-speed arithmetic computation technique to generate the mathematical functions required for geometrical mapping. The second is the need for an extensive library of geometrical mapping functions. To overcome these, two key techniques have been developed in [4,6]: The first is a very high speed radix-2 signed-digit adder and the second is a pipelined micro-programmable arithmetic function generator. In this paper, we study the same problem with the goal of optimizing the overall functionality and performance. We achieve this goal by improving the basic cell.

In the following section, we will review the requirement of the geometrical mapping processor by introducing its definition and applications. In Section 3, we will study various CORDIC schemes to implement a basic cell, which can be used to compose the necessary function set for the geometric transformations.

2 Geometrical Mapper

Transformation of a sub-image requires a mapping of the sub-image from one point to the transformed, pixel by pixel. To rearrange the image, it is necessary to calculate the destination address of each pixel, which is called a geometrical mapper

In the field of plane projective geometry, transformation from a point to another point is represented as a multiplication in homogeneous coordinates [10]. Let a 2-dimensional (2-D) point $p_x = (x, y)$ is represented as (ax, ay, a) in right-handed homogeneous coordinates, with a non-zero constant a. The vector p_x is referenced to an origin $(0, 0)$. The most useful transformations are translation, scaling and rotation, examples of which are respectively defined as:

$Trans(x, d)$: translating p_x to $(x + d, y)$
$Rot(x, \theta)$: rotating the vector p_x by an angle of θ about x-axis
$Scale(x, c)$: scaling the vector p_x by c along x-axis.

$$\begin{aligned}(x, y) \cdot Trans(x, d) &= (x + d, y) \\ (x, y) \cdot Rot(x, \theta) &= (xcos\theta - ysin\theta, xsin\theta + ycos\theta) \\ (x, y) \cdot Scale(x, c) &= (cx, y)\end{aligned} \qquad (1)$$

Or, the composite of 3 different transformations in 2-D is represented by

$$T = \begin{vmatrix} c \cdot cos\theta & sin\theta & 0 \\ -c \cdot sin\theta & cos\theta & 0 \\ cd & 0 & 1. \end{vmatrix}, \qquad (2)$$

which is called an affine transformation. The affine transformation is performed via a set of multiplication and trigonometric function.

Easily observed, the affine transformation is a necessary transformation to map a sub-image into another area of the image domain, with sliding, re-sizing and proper rotation. Its immediate applications include sub-image generation for the multiple picture-in-picture (PIP) TV, image template generation for the recognition and vision/graphics processing

Further sophisticate transformation useful for the general image processing is the spherical, which basically transforms between the plane and sphere surfaces. A spherical transformation from p_x to $q_x = (u, v)$ can be represented by using a set of elementary functions, such as square root, division, and squaring operations.

$$\begin{aligned} u &= \frac{rx}{\sqrt{r^2 - x^2 - y^2}} \\ v &= \frac{ry}{\sqrt{r^2 - x^2 - y^2}}, \end{aligned} \tag{3}$$

where r denotes the curvature degree of sphere surface. A conventional way to implement the transformations starts from a software package, i.e., interactive graphics package. To implement a dedicate hardware, possibly a set of modular structures in VLSI, it is necessary to figure out a basic cell of those functions, and there has been two different approach: the first based on a set of elementary function generators and the second on a programmable module. For the first approach, fast function generators are necessary and the performance is limited by the slowest function generator. Apparently, the trigonometric functions are the bottleneck while being implemented via the first idea. To optimize the trigonometric function generation, while considering the regularity of its structure, CORDIC has been suggested the recursiveness of the CORDIC iteration has been misleading a concept that the second approach is not usually better than the first one.

Recently, as VLSI technologies evolve, the effectiveness of the integration is not simply a complexity of the multiplication but also implies a communication complexity more than the multiplication complexity include regularity of the structure, simplicity of the design and localization of the interfacing. In these senses, CORDIC has been widely reviewed again, and shown to be appropriate for a couple of algorithmic processors. In brief, CORDIC is a set of recursive algorithms, which can be easily programmed to generate a set of elementary functions via a different mode and a proper zero-enforcing. It is also capable of vector-oriented processing.

3 CORDIC Techniques

In this section, we will review CORDIC functions to i) perform a vector transformation and ii) generate elementary functions. CORDIC comprises of three linear recursive equations, namely $X-$, $Y-$ and $Z-$ recurrences. Table 1 summarizes the computing mode, input and output specifications of CORDIC functions of our interest. As shown in the Table, these functions are classified into two cases, one which enforces $Z[N]$ to be zero (known as *rotating*) and the other which enforces $Y[N]$ to be zero(known as *vectoring*). We will discuss these cases in the following sections.

3.1 Rotating case

The vector rotation for $p_x = (X[0], Y[0])$ by the angle θ can be realized by an iteration algorithm called CORDIC [12] instead of computing trigonometric functions and applying matrix multiplication. CORDIC realizes a vector rotation by a partial sum of micro-angle rotations with a pre-fixed sequence of angles. When the rotation macro-angle is represented

Mode	Input	Enforcing	Output
Circular	$Z[0] = \theta, (X[0], Y[0])$	$Z[N] = 0$	Rotation by θ
Circular	$Z[0] = 0, (X[0], Y[0])$	$Y[N] = 0$	$X[N] = \sqrt{X[0]^2 + Y[0]^2}$ $Z[N] = tan^{-1}(Y[0]/X[0])$
Linear	$Z[0] = 0, (X[0], Y[0])$	$Y[N] = 0$	$Z[N] = Y[0]/X[0]$
hyperbolic	$(X[0], Y[0])$	$Y[N] = 0$	$X[N] = \sqrt{X[0]^2 - Y[0]^2}$

Table 1: Available CORDIC Processing

as a sum of decomposed micro-angles, i.e $\theta = \sum_{k=0}^{n} \theta_k$,

$$r_x^t = \prod_{k=0}^{n} k_k \begin{bmatrix} 1 & -tan\theta_k \\ tan\theta_k & 1 \end{bmatrix} p_x^t \tag{4}$$

where $k_k = cos\theta_{i,k}$ is a micro-scale composing a final scale factor, explained later. Such a specific form of the pre-fixed micro-angle sequence as $\tan^{-1} 2^{-i}$, is attractive for VLSI implementation since it is composed only of additions, shiftings, and a arctangent lookup table

Non-redundant : The micro-iterations of the conventional (hereafter, it will be called non-redundant) CORDIC use the following 3 linear recursive equations [12]:

$$\begin{aligned} X[i+1] &= X[i] + m\sigma_i 2^{-i} Y[i] \\ Y[i+1] &= Y[i] - \sigma_i 2^{-i} X[i] \\ Z[i+1] &= Z[i] - \sigma_i \tan^{-1} 2^{-i} \end{aligned} \tag{5}$$

where m will be set to one for the circular CORDIC, while $m = 0$ for the linear and -1 for the hyperbolic. With an initial value of $Z[0] = \theta$, CORDIC rotates initial values of $X[0]$ and $Y[0]$, to the last value $X[n]$ and $Y[n]$ while making $Z[i]$ close to zero in each i iteration, so that $Z[n]$ is forced to be zero. With n number of iterations, n-bit accuracy of $X[N]$ and $Y[N]$ can be achieved. For a known angle, the direction of the rotation, σ_i can be pre-computed or calculated one by one on-the-fly using the following selection function.

$$\sigma_i = \begin{cases} 1 & \text{if } Z[i] \geq 0 \\ -1 & \text{if } Z[i] < 0 \end{cases} \tag{6}$$

The CORDIC rotation does not preserve the input norm. To get a rotated vector having the same length as the input $(X[0], Y[0])$, $X[n](Y[n])$ needs to be compensated by a scaling factor K

$$K = \frac{\|[X[n], Y[n]]^t\|}{\|[X[0], Y[0]]^t\|} = \prod_{i=0}^{n-1} \sqrt{1 + \sigma_i^2 2^{-2i}}, \tag{7}$$

where $\|\cdot\|$ stands for the norm of the vector. Note that K is *constant* for the non-redundant scheme since σ_i is in {-1, 1}.

Redundant : Non-redundant CORDIC is slow inherently with delay of $O(n^2)$ due to its recursiveness and serial dependency, since a micro-rotation with delay $O(n)$ should be finished before processing the next micro-rotation. Delay performance of a macro-rotation (n micro-rotations) can be improved from $O(n^2)$ to $O(n)$ by using redundant arithmetic (carry-free addition such as carry save or signed-digit addition) to determine the direction of the rotation $\hat{\sigma}_i$, based on an estimate instead of an exact value [14]. The redundant arithmetic gives a delay of $O(1)$ instead of $O(n)$, and the estimation of direction is necessary not to erode the advantage of $O(1)$. This requires the modification of the recurrences and selection function. This redundant CORDIC scheme produces the output about 4 times faster than the non-redundant [14]. However, it introduces additional cost since the scale factor K is variable depending on a macro-angle by allowing $\hat{\sigma}_i$ to be in {-1, 0, 1}.

Constant-Factor-Redundant : To reduce implementation cost of redundant CORDIC, it would be good to have a constant scale factor by forcing $\hat{\sigma}_i$ in {-1, 1}. However, since $\hat{\sigma}_i$ is determined from an estimate, there arises a convergence assurance question. A scheme appending correcting iteration stages at proper positions was proposed for it [15]. Along to this idea, the number of extra correcting iterations is further reduced by dividing the micro-iterations (for $i = 0$ to $i = n-1$) into two groups: one group where the direction of the rotation is in {-1, 1} for $i = 0$ to $i = n/2$ and the other in {-1, 0, 1} for $i = (n+1)/2$ to $i = n-1$ correcting iterations by 50 % since correcting iteration is not needed for the second half of the micro-iterations and we still obtain a constant scale factor K since the value of K in n-bit precision does not depend on the $\hat{\sigma}$ value for $(n+1)/2 \leq i \leq (n-1)$. Z-recurrence also can be modified so that $\hat{\sigma}_i$ is determined quickly by looking at a few most significant bits. This new scheme is called Constant-Factor-Redundant-CORDIC(CFR-CORDIC). The modified recurrences and selection functions for the scheme are described below.

$$\begin{aligned} X[i+1] &= X[i] + \hat{\sigma}_i 2^{-i} Y[i] \\ Y[i+1] &= Y[i] - \hat{\sigma}_i 2^{-i} X[i] \\ U[i+1] &= 2(U[i] - \hat{\sigma}_i 2^{i} \tan^{-1} 2^{-i}) \end{aligned} \tag{8}$$

where $U[i]$ is for the implementation simplicity, which is equal to $2^i Z[i]$, and the selection function is given as follows:

$$\hat{\sigma}_i = \begin{cases} 1 & \text{if } \hat{U}[i] > 0 \\ & \text{or } \hat{U}[i] = 0 \cap i < n/2 \\ 0 & \hat{U}[i] = 0 \cap i \geq n/2 \\ -1 & \text{if } \hat{U}[i] < 0 \end{cases} \tag{9}$$

When t fractional bits are used in the estimate value, i.e., $\hat{U}[i]$ is computed using t fractional bits of redundant representation of $U[i]$, the following correcting iteration need to be included, where the interval between indexes of correcting iterations should be less than or equal to $(t-1)$ up to the last iteration index equal to $n/2$. When the correction stage is necessary at the jth step of micro-iteration,

$$U^C[j+1] = U[j+1] - 2\hat{\sigma}_j^C 2^j tan^{-j} 2^{-j} \tag{10}$$

with the direction of the rotation $\hat{\sigma}_j^C$ determined from the same selection function of eq.(9), except being decided based on $\hat{U}[j+1]$ instead of $\hat{U}[i]$.

3.2 Vectoring case

While the rotating case affords vector-wise rotation to implement a geometrical mapper, the vectoring case does elementary functions as in Table 1. Apparent difference between the vectoring and rotating mode is the zero enforcing parameter, which necessitates a different selection function. For the conventional CORDIC, the recurrence equations are given:

$$\begin{aligned} X[i+1] &= X[i] + \sigma_i 2^{-i} Y[i] \\ Y[i+1] &= Y[i] - \sigma_i 2^{-i} X[i] \\ Z[i+1] &= Z[i] + \sigma_i \tan^{-1} 2^{-i} \end{aligned} \tag{11}$$

with the following selection function,

$$\sigma_i = \begin{cases} 1 & \text{if } Y[i] \geq 0 \\ -1 & \text{if } Y[i] < 0 \end{cases} \tag{12}$$

The selection function for CFR-CORDIC in vectoring has been developed shown below: Let $W[i] = 2^i Y[i]$ in the same token as for the rotating case, then

$$\begin{aligned} X[i+1] &= X[i] + \hat{\sigma}_i 2^{-i} Y[i] \\ W[i+1] &= 2(W[i] - \hat{\sigma}_i X[i]) \\ Z[i+1] &= Z[i] + \sigma_i \tan^{-1} 2^{-i} \end{aligned} \tag{13}$$

$$\hat{\sigma}_i = \begin{cases} 1 & \text{if } \hat{W}[i] > 0 \\ & \text{or } \hat{W}[i] = 0 \cap i < n/2 \\ 0 & \hat{W}[i] = 0 \cap i \geq n/2 \\ -1 & \text{if } \hat{W}[i] < 0 \end{cases} \tag{14}$$

Here the correcting stage at the jth step is defined as follows:

$$W^C[j+1] = W[j+1] - 2\hat{\sigma}_j^C X[j+1] \tag{15}$$

So far, we discussed about recursive structures of several CORDIC schemes to implement the basic PE. The PE, augmented by a translator, necessitates scaling operation at each stage, because shuffling of the output at each stage makes continuous accumulation of the scaling factor complex to be processed at the final stage. The scaling operation has been solved either by an explicit way or an implicit. The explicit way is dividing the rotated vector by a constant, which is known for the non-redundant, to be calculated while running the micro-steps of CORDIC [12,14]. The division can be processed by another CORDIC (in a linear mode) or a divider. The implicit approach reconfigures the sequence of micro-iterations of the CORDIC, eventually to have a different norm from that without scaling micro-iterations. Scaling micro-iterations target in general at making the adjusted scaling factor in a form of 2^i or 1, which can be easily set to the unit size. Each micro-iteration can be composed of i) reduction axis-scaling [16], ii) repetition of vector-scaling, iii) expansion axis-scaling or combinations thereof. Relevant issues regarding search for the solution are to be further studied, better than the greedy method or the decomposed search [18]. In summary, the explicit scaling almost doubles the system complexity, while the implicit increases 25 % for non-redundant CORDIC and about 30 % for redundant CORDIC.

3.3 VLSI Scheme

To maximize the throughput of the geometric processor, the fully spanned architecture is selected. Affine transformer is a trivial case, which can be implemented by using a single CORDIC of which micro-iteration is expanded to include an addition. To implement a spherical transformer, 4 CORDICs are configured: i) circular square root of $\sqrt{x^2+y^2}$, ii) hyperbolic square root of $\sqrt{r^2-(\sqrt{x^2+y^2})^2}$, and two iii) linear divisions of u and v. To get first estimates of the VLSI size, a typical TV image processing application is considered: $O(10^5)$ pixel/image addressing and $O(10^{-1})sec$ screen flashing. For the case, the number of input bits $b_i \approx \sqrt{pixel\ \ number}$, for which 12 bits are sufficient. To allow possible interpolations between pixels, b_f is set to be 16. Each CORDIC module requires $(b_i + log_2 b_i)$ steps of micro-iterations, and 30% additional iterations for an implicit scaling.

For the spherical transformer, using fully spanned 4-CORDIC, the number of TRs are estimated about 30K (4*6K*1.3).

References

[1] R. Nicholl and T. Nicholl,"Performing Geometric Transformations by program Transformation," *ACM Trans. on Graphics,* Vol. 9, No 1, pp.28-40, 1990.

[2] N. Ansari and E. Delp,"Recognizing Planar Objects in 3-D Space," *Proc. of SPIE,* Vol. 1197, pp.127-138, 1989.

[3] R. Cossu, M. Ercoli and L. Moltedo,"Extension of CGI functions for Generation and Manipulation of Raster Image," *Computers & Graphics,* Vol.13, No 1, pp.39-48, 1989.

[4] T. Nakanishi and H. Yoshimura,"A High-speed Address Generator for Affine Transformation," *Nat. Conv. IECE*, 1985.

[5] ACM/SIGGRAPH Graphics Standards Planning Committee, Report of the CORE Definition Subgroup, 1977.

[6] H. Yoshimura, T. Nakanishi and H. Yamauchi, "A 50-MHz CMOS Geometrical Mapping Processor", *IEEE Transactions on circuits and systems*, Vol 36, No. 10, pp.1360-1364, October 1989

[7] K. Arbter and et.al., "Application of Affine-invariant Fourier Descriptors to Recognition of 3-D Objects," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 12, No 7, pp.640-647, July 1990.

[8] T. Wakahara, "On-line Handwritten Character Recognition Using Local Affine Transformation," *Systems and Computers in Japan*, Vol.20, No 7, pp.10-19, July 1989.

[9] K. Aono, M. Toyokura and T. Araki, "30nsec (600 Mops) Image Processor with a Reconfigurable Pipeline Architecture," *Proc. IEEE 1989 Custom Integrated Circuits*, pp.24.4.1-4, 1989.

[10] E. Maxwell, General Homogeneous Coordinates in Space of Three Dimensions, The University Press, Cambridge, England, 1961.

[11] J. Eldon, Z. Stroll and E. Swartzlander,"Image Processing Address Generator Chip," *Proceedings of IEEE Int. Conf. Acoustics, Speech, and Signal Processing 3*, pp.993-996, 1985.

[12] J. Walther, "A Unified Algorithm for Elementary Functions", *AFIPS Spring Joint Computer Conference*, pp.379-385, 1971.

[13] H. Kung,"Let's Design Algorithms for VLSI Systems," *Caltech Conf. VLSI*, pp.65-90. 1979.

[14] M. Ercegovac and T. Lang, "Redundant and On-Line CORDIC: Application to Matrix Triangularization and SVD", *IEEE Trans. on Computers*, Vol. C-39, No 6, pp.725-740, June 1990.

[15] N. Takagi, T. Asada and S. Yajima,"Redundant CORDIC Methods with a Constant Scale Factor for Sine and Cosine Computation", Submitted to IEEE Trans. on Computers, 1989.

[16] G. Haviland and A. Tuszynski,"A CORDIC Arithmetic Processor Chip," *IEEE Trans. on Computers*, Vol C-29, No 2, pp.68-79, Feb. 1980.

[17] J. Delosme, "VLSI Implementation of Rotations in Pseudo-Euclidean Spaces" , *Proceedings of IEEE Int. Conf. Acoustics, Speech, and Signal Processing 2*, pp.927-930, 1983.

[18] J. Lee and T. Lang, "Matrix Triangularization by Fixed-point Redundant CORDIC with a Constant Scale Factor" , *Proc. SPIE Conference on Advanced Signal Processing Algorithms, Architectures, and Implementations*, July 1990.

[19] S. Note et. al.,"Automated Synthesis of a High Speed CORDIC Algorithm with the CATHEDRAL-III Compilation System", *Int. Conf. Circuit and System*, pp.581-584, 1988.