Virtually scaling-free adaptive CORDIC rotator

K. Maharatna, A. Troya, S. Banerjee and E. Grass

Abstract: The authors propose a coordinate rotation digital computer (CORDIC) rotator algorithm that eliminates the problems of scale factor compensation and limited range of convergence associated with the classical CORDIC algorithm. In the proposed scheme, depending on the target angle or the initial coordinate of the vector, a scaling by 1 or $1/\sqrt{2}$ is needed that can be realised with minimal hardware. The proposed CORDIC rotator adaptively selects the appropriate iteration steps and converges to the final result by executing on average only 50% of the number of iterations required by the classical CORDIC. Unlike for the classical CORDIC, the value of the scale factor is completely independent of the number of executed iterations. Based on the proposed algorithm, a 16-bit pipelined CORDIC rotator was implemented. The silicon area of the fabricated pipelined CORDIC rotator core is 2.73 mm^2. This is equivalent to 38 000 inverter gates in the used 0.25 μm BiCMOS technology. The average dynamic power consumption of the fabricated CORDIC rotator is 17 mW at a 2.5 V supply voltage and a 20 Ms/s throughput. Currently, this CORDIC rotator is used as a part of the baseband processor for a project that aims to design a single-chip wireless modem compliant with the IEEE 802.11a standard.

1 Introduction

The Coordinate Rotation Digital Computer (CORDIC) algorithm [1, 2] provides an elegant way of computing various transcendental and trigonometric functions [2–4] by merely using table look-up, shift and addition operations. Since its introduction, the CORDIC algorithm has been used in several arithmetic processors as well as to formulate and implement different modern digital signal processing (DSP) algorithms [5–11]. In principle, a CORDIC accepts three input variables x, y and z and generates the output x', y' and z'. It can operate in two different modes: (i) rotation; and (ii) vectoring where, the z or y variable respectively, is forced to zero through a series of iterations. Each of these modes can be utilised in circular, linear and hyperbolic coordinate systems to compute various functions as shown in Table 1. The $K*$ are the scale factors that are generated during the CORDIC computation process.

The classical CORDIC approach [1, 2] suffers from three principal drawbacks: (i) the requirement of a scale factor compensation; (ii) the magnitude restriction of the input variables; and (iii) its low speed of execution. Whereas the first drawback requires additional multiplication operations, the second drawback incurs a significant impact on the accuracy of the computed function since it depends on how closely the variables y (in vectoring mode) or z (in rotation mode) can be driven to zero. They can be driven close to zero if the initial inputs lie within a certain range called the 'range of convergence'. Table 1 also lists the range of convergence for different modes of CORDIC operation. The third drawback comes from the iterative nature of the CORDIC algorithm. These drawbacks have motivated research on the development of high-performance special purpose CORDIC processors and various implementations of CORDIC processors have been suggested [12–17].

Circuit-level speed enhancements of the CORDIC algorithm have been achieved by unfolding the iteration stages and by realising it in a pipelined fashion. Redundant arithmetic has also been used to speed-up the addition operations that are at the heart of the CORDIC algorithm. On the other hand, to achieve algorithm-level speed enhancements, it is necessary for the CORDIC algorithm to converge to the final result by executing as small a number of iterations as possible. However, the penalty associated with this approach when applied to the classical CORDIC formulation is that the scale factor becomes non-constant and non-predictable and thus, extra post-processing cycles and circuitry are needed for its compensation. The complexity of the extra post-processing circuitry and cycles is in principle comparable to that of the basic CORDIC process itself.

Two different methods have been suggested to extend the 'range of convergence' of the CORDIC. The first one is to use mathematical identities to pre-process the CORDIC input quantities [2, 18]. This process is broadly known as the 'argument reduction technique'. Such mathematical identities do indeed help ease the present limitations but they are cumbersome to use in hardware applications due to a significant processing time and large hardware overhead. The second approach requires repetition of certain iteration steps [19–21]. Depending on how these repeated iterations are chosen, a large number of such iterations may be necessary to obtain an accurate result which increases the processing time significantly [15]. Moreover, in this type of approach, the scale factor does not remain constant and once again requires extra processing hardware for its compensation.

IEE Proceedings online no. 20041107

doi: 10.1049/ip-cdt:20041107

Paper first received 14th April and in revised form 25th August 2004

K. Maharatna is with the Department of Electrical and Electronics Engineering, University of Bristol, Bristol, UK

A. Troya and E. Grass are with Wireless Communication Systems, Institute for High Performance Microelectronics (IHP), Im Technologie Park 25, 15236, Frankfurt (Oder), Germany

S. Banerjee is with the Department of Electronics and ECE, Indian Institute of Technology, Kharagpur, India

Table 1: Function of the CORDIC in different modes of operation

	Hyperbolic	Linear	Circular
$y \rightarrow 0$ (vectoring)	$x' = K_h\sqrt{(x^2 - y^2)}$ $z' = z + \tanh^{-1}(y/x)$ $\lvert\tanh^{-1}(y/x)\rvert \leq 1.1182$	$x' = x$ $z' = z + (y/x)$ $\lvert y/x\rvert \leq 1$	$x' = K_c\sqrt{(x^2 + y^2)}$ $z' = z + \tan^{-1}(y/x)$ $\lvert\tan^{-1}(y/x)\rvert \leq 1.7433\,(99.9^\circ)$
$z \rightarrow 0$ (rotation)	$x' = K_h[x\cosh(z) + y\sinh(z)]$ $y' = K_h[x\sinh(z) + y\cosh(z)]$ $\lvert z\rvert \leq 1.1182$	$x' = x$ $y' = y + xz$ $\lvert z\rvert \leq 1$	$x' = K_c[x\cos(z) - y\sin(z)]$ $y' = K_c[x\sin(z) + y\cos(z)]$ $\lvert z\rvert \leq 1.7433\,(99.9^\circ)$

The principal aim of the present work is to develop a CORDIC processor that: (i) is power efficient; (ii) free from the scale factor compensation problem; and (iii) has a convergence range over the entire coordinate space. More specifically, we concentrate on the rotation and vectoring mode of operation (i.e. $z \rightarrow 0$ or $y \rightarrow 0$ respectively) in the Cartesian coordinate system. The current work is based on a 'scaling-free' rotational CORDIC algorithm [9] which has been utilised to realise a number of DSP algorithms [9–11]. However, this CORDIC algorithm has a very small range of convergence and can neither be applied for the rotation operation with large angles nor for the vectoring mode of operation. In this work we propose a 'virtually scaling-free' CORDIC rotator algorithm which requires a constant scale factor and converges to the final result executing a minimal number of iterations. This potentially reduces the power consumption since the number of arithmetic computations is reduced. The main contributions of the present work are,

1. We develop a CORDIC algorithm where the scale factor can assume only the values of either one or $1/\sqrt{2}$ depending on the input vector. This means that the scale factor is *a-priori* predictable.
2. Unlike the classical CORDIC, the value of the scale factor here is independent of the number of executed iterations. Thus, algorithm-level speed enhancement and power saving are possible in the proposed scheme (by skipping some not actually needed iterations) without worrying about the scale factor compensation.
3. We show that a basic convergence range of $[0^\circ,\ 22.5^\circ]$ is absolutely sufficient to compute the result of a CORDIC rotation with the target angle (rotation mode) lying anywhere in the coordinate space. This is also valid for the vectoring operation. Utilising this property, the range of convergence of the proposed CORDIC is extended over the entire coordinate space.
4. In the proposed scheme on an average, a 50% reduction in the number of iterations as compared to the classical CORDIC is achieved.
5. A 16-bit pipelined CORDIC rotator is successfully designed; fabricated and tested to validate the efficiency of the proposed scheme.

2 The conventional CORDIC algorithm

The rotation of a vector $[x \quad y]^T$ in the Cartesian coordinate system can be described as:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{1}$$

where, $[x' \quad y']^T$ is the final vector and θ is the angle of rotation $(-99.9^\circ \leq \theta \leq 99.9^\circ)$. In the above equation it was assumed that the rotation takes place in a clockwise direction. For the conventional CORDIC method, the target angle θ is expressed as a summation of a number of elementary rotational angles α_i so that:

$$\theta = \sum_{i=0}^{b-1} \sigma_i \alpha_i \tag{2}$$

where, b is the bit precision of the machine in which the operation is to be implemented and $\sigma_i \in \{1, -1\}$, and is known as the direction of rotation. In the conventional circular CORDIC operation α_i is expressed as:

$$\alpha_i = \tan^{-1}(2^{-i}) \tag{3}$$

Substituting (2) into (1) and using (3) one may write:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \prod_{i=0}^{b-1} \cos\alpha_i \begin{bmatrix} 1 & \sigma_i 2^{-i} \\ -\sigma_i 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{4}$$

and

$$\sigma_i = \text{sign}\left[\theta - \sum_{r=0}^{i-1} \alpha_r\right] \tag{5}$$

$$z_{i+1} = z_i + \sigma_i 2^{-i} \tag{6}$$

Equations (4)–(6) are the basic working equations of the CORDIC unit operating in the circular mode. The physical meaning of these equations is that the target angle inputted as variable z, is driven to zero by to-and-fro motion of the vector. At each iteration step the sign of the residual angle σ_i is calculated and accordingly, the vector is rotated in the clockwise or counter-clockwise direction. For the vectoring mode of operation, the value of y is forced to zero in the same iterative manner and the angles are accumulated in the z variable. In essence, the mathematical operations involved here are a sequence of multiply-and-add operations. From the hardware point of view, this function can be implemented using shift-and-add operations only. However, the result obtained in (4) requires scaling by a factor $\prod_{i=0}^{b-1} \cos\alpha_i$. The scale factor remains a machine constant as long as i runs through all the steps from zero to $b-1$. However, if i changes in a different manner, i.e. if some of the allowed iterations (i.e. elementary angle rotations) are dropped or repeated, the scale factor will not remain constant or predictable. For its compensation one may require complicated hardware structures or comparable post-processing cycles.

3 Architecture of a scaling-free CORDIC rotator

Unlike the classical CORDIC method, in the scaling-free CORDIC [9] the target angle is achieved by rotating the vector in one direction only (either clockwise or counter-clockwise) through sufficiently small elementary angles

α_i so that the norm of the vector is always preserved at each of the iteration steps. These elementary rotational angles can be expressed by the following equation:

$$\sin \alpha_i \cong \alpha_i = 2^{-i} \tag{7}$$

The 'smallness' of α_i can be worked out by considering the series expansion of sine and cosine of α_i which are given by:

$$\sin \alpha_i = \alpha_i - \alpha_i^3/3! + \alpha_i^5/5! - \ldots. \tag{8}$$

$$\cos \alpha_i = 1 - \alpha_i^2/2! + \alpha_i^4/4! - \ldots. \tag{9}$$

Now, using the approximation $\alpha_i = 2^{-i}$, (8) and (9) can be written as follows:

$$\sin \alpha_i = 2^{-i} - 2^{-3i}/6 + 2^{-5i}/120 - \ldots \tag{10}$$

$$\cos \alpha_i = 1 - 2^{-2i}/2 + 2^{-4i}/24 - \ldots \tag{11}$$

The largest term that is neglected during the approximation of $\sin \alpha_i \cong \alpha_i = 2^{-i}$, is the second term in the sine series, which is:

$$2^{-3i}/6 = 2^{-(3i+\log_2 6)} = 2^{-(3i+2.585)} \tag{12}$$

Now, let the machine in which the operation is supposed to be implemented have a wordlength of b-bits. Then, if $(3i + 2.585)$ equals or exceeds b, multiplication of any quantity by $2^{-(3i+2.585)}$ will essentially become machine zero since this operation physically means that the multiplicand gets a right-shift equal to or greater than b-bits. Therefore, the condition for preserving accuracy during such approximation becomes:

$$3i + 2.585 \geq b \text{ or } i \geq \lceil (b - 2.585)/3 \rceil$$

$$= p \text{ (since } i \text{ can adopt only integer values)} \tag{13a}$$

However, for practical purposes, the lower bound of i can be relaxed slightly and can be expressed as:

$$\lfloor (b - 2.585)/3 \rfloor = p \tag{13b}$$

The upper limit of i is $b - 1$ since a right-shift of any b-bit number by b-bits will result in machine zero.

Thus, under the above circumstances, the values of $\sin \alpha_i$ and $\cos \alpha_i$ can be expressed as:

$$\sin \alpha_i = 2^{-i} \tag{14}$$

$$\cos \alpha_i = 1 - 2^{-(2i+1)} \tag{15}$$

Considering the above approximation, $\sigma_i = +1$ (since the sign of the residual angle is always the same for all iterations) and clockwise rotation of the vector, therefore (1) may be rewritten as:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \prod_{i=p}^{b-1} \begin{bmatrix} 1 - 2^{-(2i+1)} & 2^{-i} \\ -2^{-i} & 1 - 2^{-(2i+1)} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{16}$$

It can be noted that like (4), expression (16) can be realised in practice using only shift-and-add operations. On the other hand, contrary to (4), no scaling term appears in (16), which implies a 'scaling-free' CORDIC operation. The elimination of the scaling term is a significant step, since it not only rules out the requirement of extra post-processing circuitry but also saves extra processing time. The elementary datapath component for implementing the above stated operation is shown in Fig. 1 which can be viewed as an elementary rotational stage rendering a rotation of α_i to its input vector. It requires four shifters and four adder/subtractors as the datapath components. Thus, compared to the datapath of elementary rotational stages of an unscaled CORDIC, it requires two additional shifters and two more adder/subtractors. However, for a pipelined implementation, the shifters are reduced to direct hard-wired connections and the effective overhead becomes two adder/subtractors per elementary rotational stage. On the other hand, for the elementary rotational stages corresponding to $i \geq b/2$, the extra shifters and the adder/subtractors can be omitted as the right-shift of the input quantity by $(2i + 1)$ becomes machine zero and thus no longer affects the accuracy. Thus, for the elementary rotational stages corresponding to $i \geq b/2$, the hardware cost is the same as that of the elementary rotational stages of the conventional CORDIC.

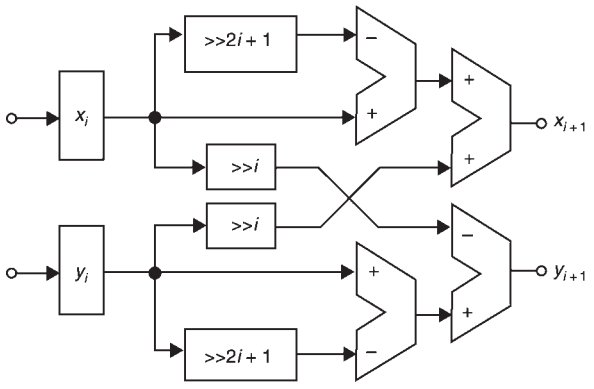

Fig. 1 *Block diagram of the elementary CORDIC rotor stage producing a rotation of by an angle* α_i

4 Expansion of the angular convergence range in the rotation mode

Since the CORDIC unit presented in the preceeding Section is scaling free it saves post-processing hardware as well as the processing time. However, the principal drawback of this CORDIC algorithm is that only computations with very small target angles can be carried out using this principle. As an example, let us consider the implementation of a CORDIC unit having a wordlength of 16-bits. Then, according to the restriction imposed by (13b), the iteration index i can assume the values $4, 5, \ldots, 15$.

Since in our case the target angle $\theta = \sum_{i=p}^{b-1} \alpha_i$, and according to (7), $\sin \alpha_i \cong \alpha_i = 2^{-i}$, the largest angle which can be computed is only $\pm 7.16^\circ$. This is even less than the range of convergence of the conventional CORDIC (99.9°) and is clearly insufficient for general-purpose usage. Thus, methods should be invented to expand the angle computation range while still preserving the 'scaling-free property' of the proposed CORDIC unit.

To expand the angular range of convergence for the scaling-free CORDIC algorithm both, the 'argument reduction' and the 'repetition of certain iteration steps' techniques are exploited with modifications. Firstly, an argument reduction technique is used to reduce the total angular range to be computed. Secondly, the elementary rotational operations are carried out in an adaptive manner to enhance the rate of convergence and to force the final angle approximation error below a certain prespecified limit. In the following discussion this scheme is explained in detail.

The main objective of the argument reduction technique is to uniquely map the results of CORDIC rotations with

large target angles θ to the results of CORDIC rotations with relatively small target angles ϕ. To do this, we divide the coordinate space into 16 equal domains (i.e. four domains per quadrant) each having a uniform angular span of $\pi/8$. Any target angle must lie in one of these 16 domains. We first examine the CORDIC rotation of an input vector with target angle θ lying in the first quadrant. This essentially means that θ lies in one of the four domains A ($[0,\ \pi/8)$), B ($[\pi/8,\ \pi/4)$), C ($[\pi/4,\ 3\pi/8)$) or D ($[3\pi/8,\ \pi/2]$). In each domain, θ can be redefined in terms of another angle ϕ as described in the following equations:

$$\theta = \phi \quad \text{in domain A} \tag{17}$$

$$\theta = \pi/4 - \phi \text{ in domain B} \tag{18}$$

$$\theta = \pi/4 + \phi \text{ in domain C} \tag{19}$$

$$\theta = \pi/2 - \phi \text{ in domain D} \tag{20}$$

It is to be noted that the angle ϕ is always bounded in the interval $[0,\ \pi/8]$. Substituting (17)–(20) into (1), the CORDIC operation on an input vector $[x \quad y]^T$ can be expressed in different domains as:

$$\begin{bmatrix} x_{\mathrm{fA}} \\ y_{\mathrm{fA}} \end{bmatrix} = \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \text{ in domain A} \tag{21}$$

$$\begin{bmatrix} x_{\mathrm{fB}} \\ y_{\mathrm{fB}} \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} (\cos\phi + \sin\phi) & (\cos\phi - \sin\phi) \\ -(\cos\phi - \sin\phi) & (\cos\phi + \sin\phi) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \text{ in domain B} \tag{22}$$

$$\begin{bmatrix} x_{\mathrm{fC}} \\ y_{\mathrm{fC}} \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} (\cos\phi - \sin\phi) & (\cos\phi + \sin\phi) \\ -(\cos\phi + \sin\phi) & (\cos\phi - \sin\phi) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \text{ in domain C} \tag{23}$$

$$\begin{bmatrix} x_{\mathrm{fD}} \\ y_{\mathrm{fD}} \end{bmatrix} = \begin{bmatrix} \sin\phi & \cos\phi \\ -\cos\phi & \sin\phi \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \text{ in domain D} \tag{24}$$

where, $x_{\mathrm{f}*}$ denotes the final vector resulting from CORDIC operations with target angles lying in different domains.

Now recapitulating that the CORDIC rotation for ϕ and $-\phi$ can be expressed as:

$$\begin{bmatrix} x'_+ \\ y'_+ \end{bmatrix} = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{25}$$

$$\begin{bmatrix} x'_- \\ y'_- \end{bmatrix} = \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \sin\phi \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{26}$$

where, the $+$ and $-$ sign at the suffix of x' and y' denote positive and negative CORDIC rotations respectively. Thus, (21)–(24) can be written as:

$$x_{fA} = x'_- \quad y_{fA} = y'_- \quad (\phi = \theta) \tag{27}$$

$$x_{\mathrm{fB}} = \frac{1}{\sqrt{2}}\left[x'_+ + y'_+\right] \quad y_{\mathrm{fB}} = \frac{1}{\sqrt{2}}\left[-x'_+ + y'_+\right] \quad (\phi = \pi/4 - \theta) \tag{28}$$

$$x_{\mathrm{fC}} = \frac{1}{\sqrt{2}}\left[x'_- + y'_-\right] \quad y_{\mathrm{fC}} = \frac{1}{\sqrt{2}}\left[-x'_- + y'_-\right] \quad (\phi = \theta - \pi/4) \tag{29}$$

$$x_{fD} = y'_+ \quad y_{fD} = -x'_+ \quad (\phi = \pi/2 - \theta) \tag{30}$$

Equations (27)–(30) show that the CORDIC operation with target angles lying in different domains in the first quadrant can be computed from the results of CORDIC rotations with target angle ϕ by simple addition and subtraction operations. This essentially means that the domains B, C and D are effectively 'folded back' to domain A. Hence, we call this technique 'domain folding'. A consequence of the domain folding operation is the generation of a constant scale factor $1/\sqrt{2}$ for the target angles lying either in domain B or domain C. This constant scale factor can be realised with minimal hardware using only shift-and-add operations within a prespecified error margin.

Up to now, only the target angle that lies in the first quadrant has been considered. By exploiting the symmetry of the coordinate axes, the domain folding technique can be easily employed to carry out CORDIC operations with target angles lying in other quadrants as well. It is easy to show that depending on the quadrant in which the target angle lies, only the sign of the final output vectors becomes changed. Thus, a range of convergence spanning the entire coordinate space is achieved. In summary, for the computation of vector rotations with an arbitrary target angle, the first step is to detect the domain in which it lies. Once the domain is detected, the computation can be carried out applying an appropriate equation from those derived in equations (27)–(30).

Although, in using the domain folding technique, it is sufficient to consider a modified range of convergence of $[0^\circ,\ \pi/8]$ only, it is still beyond the range of convergence of the scaling-free CORDIC unit presented in Section 3. To eliminate this discrepancy, one approach is to repeat some of the iteration steps more than once. The iteration index i at each stage can be chosen adaptively, in accordance with the residual angle still to be computed. The process of adaptive selection of i is described in the flowchart shown in Fig. 2. In this process at the beginning of every iteration step i, the residual angle z_i is compared with the value of 2^{-i} (the ith elementary rotation angle). If z_i is less than 2^{-i}, the ith iteration step is skipped since the meaning of this is that the residual angle to be computed is smaller than the ith elementary rotational angle. In such a case the iteration index i is updated to $i+1$ and once again the condition of equality is checked. However, if z_i is equal to or greater than 2^{-i}, x_{i+1} and y_{i+1} are computed according to (16) and the residual angle is updated accordingly. The process can be repeated until a user-defined accuracy R_{ref} is achieved. Since in our algorithm we consider only one-sided rotation, this process of selection of iteration steps ensures that only the actually needed iteration steps are performed. This reduces the number of computations significantly.

Lemma 1: Only the iteration steps corresponding to the smallest allowable value of i, is repeated more than once with the other values of i being non-repetitive.

Proof: Let at the start of jth iteration the residual angle be θ_{r} and $2^{-q} < \theta_{\mathrm{r}} < 2^{-(q-1)} < 2^{-j}$, where, $p < j < q$, p being the smallest allowable value of i. Then, according to the flowchart shown in Fig. 2, the iteration steps (or the elementary rotation operations) corresponding to $i = j$ to $(q-1)$ will be dropped and an elementary rotation

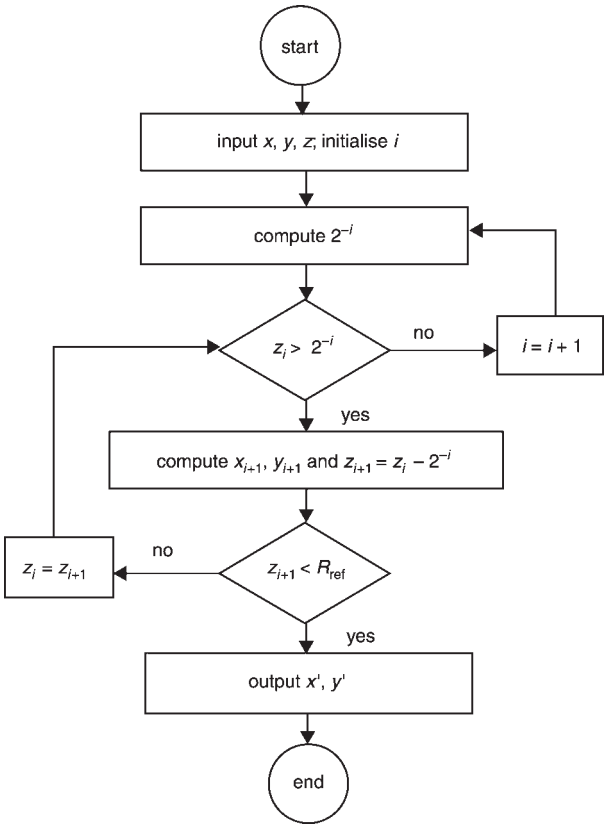

Fig. 2 *Operation principle of the adaptive CORDIC*

corresponding to $i = q$ will be performed. After this elementary rotation operation, let the new residual angle θ_{rq} still be greater than 2^{-q}. This implies that another elementary rotation operation corresponding to $i = q$ (repetition) is required. Moreover, according to the proposed scheme, application of rotation corresponding to $i = q$ twice (considering at least one repetition is required to bring down the value of the residual angle to $<2^{-q}$) essentially means that θ_r must be greater than or equal to $2^{-(q-1)}$ which is in contradiction to our initial assumption. Thus, it can be concluded that no elementary rotation steps corresponding to $i > p$ can be repeated. However, if the value of target angle θ at the start of the first iteration is such that $2^{-(p-n)} \leq \theta$ and $\theta - 2^{-(p-n)} < 2^{-p}$, where $n < p$, the elementary rotation operation corresponding to $i = p$ is to be repeated n times to cover that range. □

To examine the required number of iterations for different target angles θ within the modified convergence range $[0°, \pi/8]$ a pseudo-random sequence of angle θ has been generated and a Matlab simulation has been performed

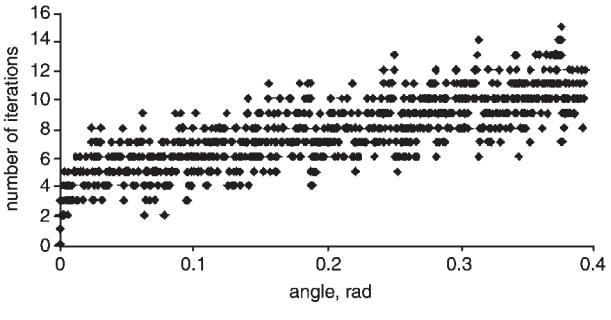

Fig. 3 *Number of required iterations with respect to the target angle lying in the range [0°, 22.5°] using the proposed algorithm*

using those angles. The result is shown in Fig. 3 considering a 16-bit wordlength and an angle approximation error of $O(2^{-16})$. It is apparent from the flowchart shown in Fig. 2 and the results of Fig. 3 that for any angle lying within this range, the required number of iterations is always smaller than the number of iterations required for the conventional CORDIC when employed for the same operation. On average, the proposed scheme needs 50% fewer iterations compared to the conventional CORDIC method. It can be easily shown analytically that according to the proposed scheme, the number of required iterations in the worst case is 15 which occurs when a CORDIC operation corresponding to the angle 21.484° is to be carried out. Thus, using the proposed scheme, a faster convergence rate compared to the classical CORDIC approach can be ensured while keeping the scale factor virtually constant.

4.1 Error analysis for rotation mode

The potential sources of error for the hardware realisation of an algorithm are two-fold: (i) the error due to the quantisation of the input word; and (ii) the finite wordlength of the arithmetic units. In the case of the CORDIC, another potential error source is the error due to the approximation of the target angles. In our scheme, the errors from the first two sources are the same as for the conventional implementation of the circular CORDIC. Thus, a detailed analysis of the third error source is presented here with the consideration that the implementation has been done with a 16-bit wordlength. However, the result can be generalised for any arbitrary implementation.

Let θ_c be the computed angle whereas θ is the ideal target angle. Then, from (1):

$$x' = x\cos\theta_c + y\sin\theta_c \tag{31}$$

$$y' = -x\sin\theta_c + y\cos\theta_c \tag{32}$$

where $\theta_c = \theta - \varepsilon$ and $\varepsilon = O(2^{-16})$ in the present case. After simplification, (31) and (32) yield:

$$x'_{error} = (x\cos\theta + y\sin\theta)(1 - \cos\varepsilon) + (-x\sin\theta + y\cos\theta)\sin\varepsilon \tag{33}$$

$$y'_{error} = (-x\sin\theta + y\cos\theta)(1 - \cos\varepsilon) - (x\cos\theta + y\sin\theta)\sin\varepsilon \tag{34}$$

Since $\varepsilon = O(2^{-16})$, $(1 - \cos\varepsilon)$ and $\sin\varepsilon$ will be $O(2^{-33})$ and $O(2^{-16})$ respectively. Thus, the maximum error that can occur due to the angle approximation according to the proposed scheme is $O(2^{-16})$ which is similar to that of the classical CORDIC [22, 23]. Equations (33) and (34) also show that the final error due to angle approximation is dependent on the initial values of the vector components x and y as well as on the ideal target angle θ. It can also be easily shown from the above mentioned equations that like the case of the classical CORDIC [22, 23], in this proposed method the final datapath error also becomes unacceptably high when non-normalised values of x and y are used at the input.

5 Expansion of the domain of convergence in the vectoring mode

The concept of domain folding can be utilised once again to achieve a convergence range over the entire coordinate space for the vectoring mode of operation of the proposed

CORDIC. To explain the complete scheme, let us first concentrate on the case where the input vector lies in the first quadrant of the coordinate space i.e. the sign of x and y are positive and they satisfy the condition $|\tan^{-1}(y/x)| \leq 90^\circ$. The first step in this scheme is to find out the appropriate domain of the input vector. Using $\tan(22.5^\circ) = (\sqrt{2}-1)$ and $\tan(67.5^\circ) = (\sqrt{2}+1)$ it can be said that the ratios $(y/(\sqrt{2}-1)x)$ and $(y/(\sqrt{2}+1)x)$ define the boundaries of the domains A-B and C-D respectively. On the other hand the boundary of domain B-C is defined by $x = y$ since $\tan(45^\circ) = 1$. Thus, by checking the inequalities $(\sqrt{2}-1)x > y$, $(\sqrt{2}+1)x > y$ and $x > y$, it is possible to determine the domain in which the initial vector lies. Once the domain is determined, the input vector can be preprocessed accordingly. Since, the range of convergence of the basic CORDIC algorithm proposed here is only 22.5°; the coordinate axes are first prerotated by an angle of 45° in a counter-clockwise direction, when the initial vectors are in domain B or C. This prerotation, in essence, is simply an addition and subtraction of x and y followed by a scaling of $\sqrt{2}$. This ensures that the final angle to be accumulated always lies within $[0^\circ, 22.5^\circ]$. For the vector lying in domain D, the approach for preprocessing is to swap x and y at the input. This swapping means that in effect the x component has to be forced to zero instead of y. This preprocessing operation is shown in the flow chart of Fig. 4 where (x_{in}, y_{in}) denotes the output of the preprocessing step.

After preprocessing, (x_{in}, y_{in}) are considered as inputs to the basic CORDIC having a convergence range of $[0^\circ, 22.5^\circ]$, which operates in the angle accumulation (vectoring) mode (shown as vectoring CORDIC in Fig. 4). The actual angle can be computed at the output of the basic CORDIC by adding/subtracting the accumulated angle to/from 45° for domain C and domain B respectively. For the vectors lying in domain A the accumulated angle during the vectoring process is the desired angle. The result of the absolute magnitude of the input vector (the square rooted result) will be available at the x' output in all these three cases. For the vector lying in domain D, the final angle can be computed by subtracting the accumulated angle from 90°. The square rooted result will be available at output y' in this case.

By exploiting the symmetry of the coordinate axes, this procedure can be directly extended to calculate the arctangent of any angle lying in the coordinate space and thus ensuring a convergence range over the entire coordinate space.

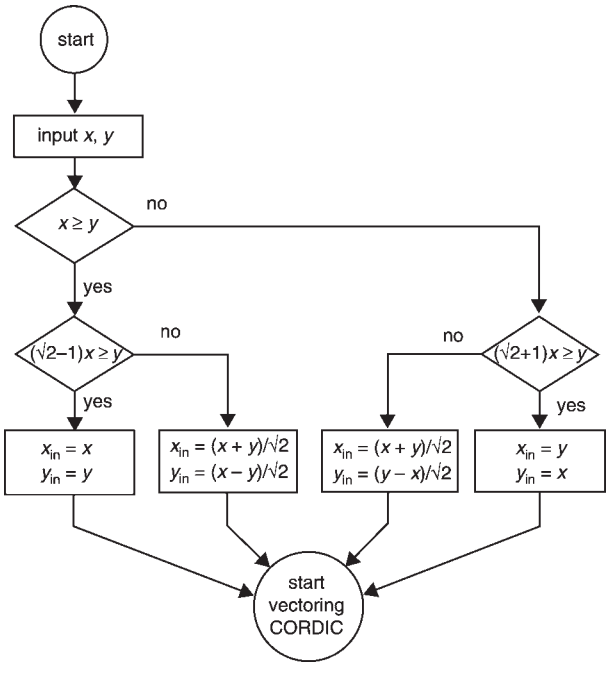

Fig. 4 *Flow diagram for the preprocessing steps for expansion of range of convergence for the proposed CORDIC in its vectoring mode of operation*

5.1 Error analysis for the vectoring mode

Let us consider that in our formulation the actually computed angle is θ_c and the ideal angle that should be computed is θ. These two angles are related by the following equation:

$$\theta_c = \theta \pm \mu \tag{35}$$

where μ is the actual error that has to be determined. We further consider that the final value of y is approximated within the precision ε. Ideally, the final value of y (i.e. y') should be zero. Thus, in an ideal situation, the following condition holds:

$$-x\sin\theta + y\cos\theta = 0 \tag{36}$$

However, since y' is approximated with accuracy ε and the angle computed is θ_c, (36) can be written as:

$$-x\sin\theta_c + y\cos\theta_c = \varepsilon \tag{37}$$

Substituting the value of θ_c from (35) and simplifying one gets:

$$\cos\mu[-x\sin\theta + y\cos\theta] \mp \sin\mu[x\cos\theta + y\sin\theta] = \varepsilon \tag{38}$$

Using (36), the first term of (38) becomes zero and the angle approximation error can be expressed as:

$$\mu = \mp\sin^{-1}\frac{\varepsilon}{x\cos\theta + y\sin\theta} \tag{39}$$

In an ideal case, $x\cos\theta + y\sin\theta = r$, where r is the magnitude of the vector. Thus, (39) can be rewritten as:

$$\mu = \mp\sin^{-1}\frac{\varepsilon}{r} \tag{40}$$

Thus, the angle approximation error is a function of the magnitude of the vector as well as the precision by which the y component of the vector is approximated to zero.

The error in the computation of the magnitude can be derived by considering the following equation of the x datapath:

$$x' = x\cos(\theta \pm \mu) + y\sin(\theta \pm \mu) \tag{41}$$

Simplifying the above equation, one gets:

$$x' = (x\cos\theta + y\sin\theta)\cos\mu \pm (y\cos\theta - x\sin\theta)\sin\mu \tag{42}$$

Using (36) and simplifying (42) yields:

$$x' = r\cos\mu = r\sqrt{1-(\varepsilon/r)^2} \tag{43}$$

where $x\cos\theta + y\sin\theta = r$. Thus, the error for the magnitude of the vector can be written as:

$$x'_{error} = r\left(1 - \sqrt{1-(\varepsilon/r)^2}\right) \tag{44}$$

As in the case of the error in angle estimation, in this case also, the final error depends on the initial magnitude of the

vector as well as on how closely the y input is driven to zero. From (40) and (44) it can be shown that for very small values of the input vector, the errors in angle approximation and magnitude calculation are too large to be acceptable. The same can be said if the value of the input vector is not normalised. This behaviour is identical to that of the classical CORDIC vectoring operation as investigated in [23].

6 Design of the 16-bit CORDIC rotator

The effectiveness of the proposed algorithm is verified through a design and implementation of a 16-bit pipelined CORDIC rotator test chip. Conventional two's complement arithmetic has been used throughout the design. In this Section, we provide the details of the design as well as the fabrication results.

6.1 Architecture

The complete CORDIC rotator capable of operating in both the rotation and vectoring modes over the entire coordinate space has three parts: (i) sign and domain detection circuitry; (ii) basic CORDIC section with a convergence range of $[0^\circ,\ 22.5^\circ]$; and (iii) the output circuitry. The block diagram of such a CORDIC rotator is shown in Fig. 5. It has three principal inputs x (16-bit), y (16-bit) and z (18-bit). An 18-bit wordlength is selected for z in order to cover the entire coordinate space ($[0,\ 2\pi]$). Similarly, it has three principal outputs x' (16-bit), y' (16-bit) and z' (18-bit). The sign detection circuitry (designated as '`sign_detect`' in Fig. 5) detects the sign of input variables x and y to find out the appropriate coordinate and domain in which the vector lies. Accordingly, the input variables are processed as has been described in Sections 4 and 5 and these processed quantities are regarded as the modified inputs to the basic CORDIC rotator. The detection of sign and domain, preprocessing and appropriate inputting of data to the basic CORDIC rotator require one clock cycle in total. The basic CORDIC rotator (designated as '`cord_pipe`' in Fig. 5) is a bit-parallel pipelined implementation with an

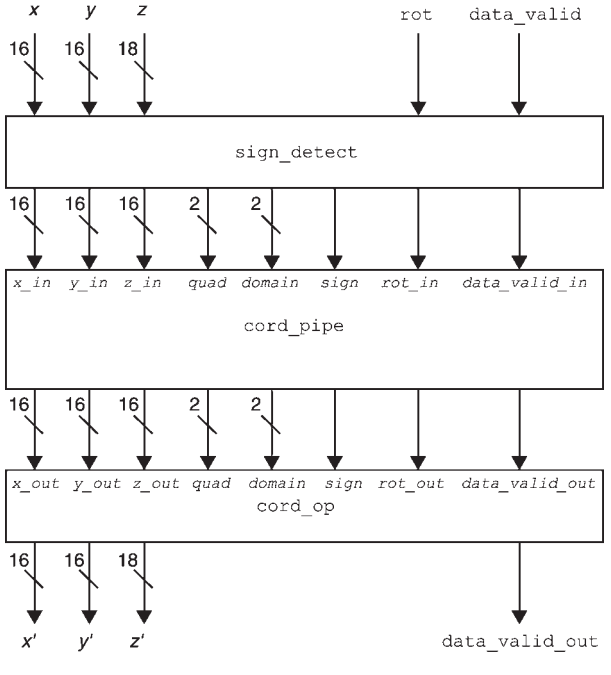

Fig. 5 *Block diagram of the pipelined CORDIC (without I/O pin multiplexing)*

internal wordlength of 16-bits. The basic pipeline is 16 stages long where the elementary rotational stage corresponding to 2^{-4} is repeated six times. Our aim is to approximate z and y for the rotation and the vectoring mode respectively with an accuracy of $O(2^{-16})$. The quadrant and domain of the initial vector detected in the `sign_detect` module are represented by two 2-bit signals namely `quad` and `domain` respectively. An additional single bit signal `sign` is also used to indicate the sign of the initial vector. These signals are transferred between two consecutive sections of the pipeline along with the data in a local register transfer manner. Thus, the data in different sections of the pipeline has a token attributed to it that carries its initial quadrant and domain information. The output circuitry (designated as `cord_op` in Fig. 5), consists of a fixed scaling unit of $1/\sqrt{2}$, a demultiplexer and a couple of adder/subtractors. This circuit post-processes the output vector emerging from the `cord_pipe` section depending on these attributes and generates the final output. All the operations at the output are completed in one clock cycle.

A single bit signal `rot` is also provided with the rotator whose logic 'high' state makes the rotator operate in rotation mode whereas; its logic 'low' state selects the vectoring mode of operation. Apart from the `rot` signal, the rotator is also equipped with a set of input and output data valid signals that indicate the presence of valid data at the input and output.

To reduce the input/output (I/O) pin count, we employed pin multiplexing at the input and output of the core CORDIC rotator. Two 8-bit datapaths are employed for the input variables x and y respectively and a 9-bit datapath is employed for input variable z. Thus, full length data (16-bit for x and y respectively and 18-bit for variable z) are available to the core CORDIC after every two clock cycles. A similar arrangement is provided at the output. Here one 8-bit datapath is provided for the variable x' and a 9-bit datapath is provided for the variable y' (for rotation)/z' (for vectoring). The I/O multiplexing circuitry uses a separate clock, which is twice as fast as the core clock. In our design we have a target core clock frequency of 20 MHz. Thus, the I/O clock frequency is of 40 MHz. This multiplexing reduces the number of required I/O pins from 100 to 56. Of those 19 pins are output pins, four pins are power pins and the remaining ones are input pins.

6.2 Hardware complexity and design flow

In our implementation, the basic CORDIC module having a convergence range of $[0^\circ,\ 22.5^\circ]$ requires 66 16-bit 2-input adders altogether. In the `sign_detect` and `cord_op` modules the terms such as $(\sqrt{2}-1)x$ and $(\sqrt{2}+1)x$ and scaling by $1/\sqrt{2}$ are realised using shift-and-add operations. Although these operations can be realised using multiple-input adders, in our implementation we have used two-input adders only. The total complexity of the `sign_detect` and `cord_op` modules are 18 and 12 16-bit two-input adders respectively. Thus, the complexity of the complete processors is 96 16-bit two-input adders. The total hardware complexity of the proposed processor in its present form is slightly higher than the classical CORDIC processor that requires 80 16-bit two-input adders considering the scale factor compensation circuitry. Despite a slightly higher hardware complexity, the proposed circuit benefits from the advantage of potentially lower power consumption compared to the classical CORDIC. This is since on average, the computational burden is reduced by 50%. However, our more recent work reveals that it is possible to reduce the total hardware cost associated with the proposed processor

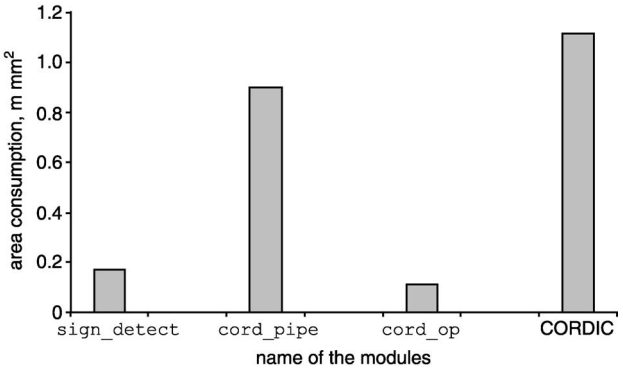

Fig. 6 *Cell area of the different modules for the pipelined CORDIC*

further and it can be made more economic than the classical CORDIC [24, 25].

For the design of the rotator, the IHP in-house design kit was used. The CORDIC rotator is first modelled in VHDL and simulated using Mentor Graphics' Modelsim simulator. After functional verification, Synopsys' Design Analyzer is used to synthesise the circuit for IHP in-house 2.5 V 0.25 µm BiCMOS technology with a target core clock frequency of 20 MHz. The synthesised cell areas of different modules are shown in Fig. 6. The Figure shows that about 15% at the cell area of the complete rotator is consumed by the sign and domain detection circuitry whereas 9.3% is consumed by the output circuitry. The cell area required by the entire rotator is equivalent to 38 000 inverter gates (approximately) in this technology.

After synthesis, the layout of the processor was performed using Cadence's Silicon Ensemble. The standard cell approach with a row utilisation of 85% is deployed. The area of the processor core after layout is 2.73 mm^2. Including 56 I/O pins, the complete chip area was measured as 6.6 mm^2.

6.3 Test results

The fabricated CORDIC rotator was packaged in a QFP 100 package. Testing was carried out in two phases. First, a pseudo-random sequence of angles and input vectors was generated. For the rotation mode of operation, the value of z was varied over the range $[0°,\ 22.5°]$ while keeping x and y constant for a particular set of z. Alternatively, for testing the vectoring mode of operation we kept $z = 0$ and the values of x and y were varied over a wide range. These vectors were used in Matlab simulations to generate our reference output results. The same set of vectors had been applied to the processor and the outputs were cross-checked with the results of Matlab. In all cases the processor exhibited a correct behaviour. The latency of the complete processor was 900 ns and the throughput was 1 set of results every 50 ns at a 20 MHz core clock frequency.

In the second phase, the current consumption of the chip was measured with a current meter when the chip operated in continuous mode with a long vector set. The average dynamic power consumption of the 26 fabricated and measured chips is 17 mW at a supply voltage of 2.5 V. The die photograph of the CORDIC processor is shown in Fig. 7.

After verification of the functional behaviour, the chips were subjected to a maximum operating frequency test. We found that all the chips showed correct functional behaviour for an operating frequency of 50 MHz (I/O clock) i.e. a 25 MHz core clock which is the limit of this tester.

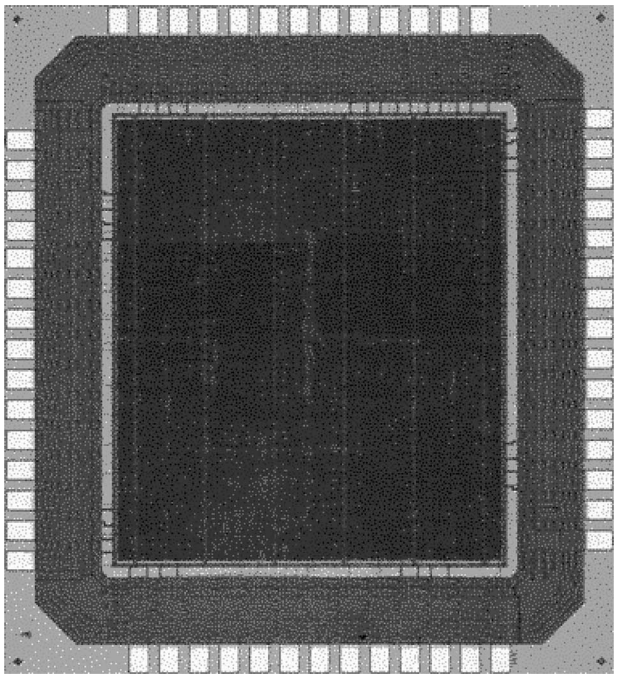

Fig. 7 *Die photograph of the pipelined CORDIC processor*

However, the synopsys' design analyzer predicts that the circuit will operate correctly up to a core clock frequency of 45 MHz.

7 Conclusions

We have described a CORDIC rotator algorithm that is virtually scaling free and has a convergence range over the entire coordinate space. The algorithm converges to the final result by adaptively selecting only needed iteration steps and hence, requires 50% fewer computations on average as compared to known CORDIC implementations. An original property of our algorithm is that the value of the scale factor (1 or $1/\sqrt{2}$) is independent of the adaptive selection of iteration steps and thus, an algorithmic-level speed-up is possible without affecting the final value of the scale factor. The computational precision of the proposed processor is similar to that of the classical CORDIC processor.

The hardware complexity of the proposed processor is slightly higher than the classical CORDIC processor. However, our more recent work reveals that the hardware cost of the proposed processor can be reduced significantly. Despite a slightly higher hardware cost, the proposed processor consumes less power compared to the classical CORDIC since the number of actually required arithmetic operations is significantly reduced.

Based on this algorithm, a 16-bit pipelined CORDIC rotator was implemented using the IHP in-house 0.25 µm BiCMOS technology. The fabrication results confirm the predicted performance, power and area parameters. Currently, this CORDIC rotator is used as a part of our baseband processor in a project that aims to design a single-chip wireless modem compliant with the IEEE 802.11a standard [26].

Without any loss of generality, redundant arithmetic can be applied in the proposed scheme for circuit-level speed enhancements. However, our main aim was to improve, on an algorithmic-level, performance and power consumption of the CORDIC processor. The proposed scheme has been shown to be efficient for that purpose.

8 Acknowledgments

The authors sincerely thank A.S. Dhar and U. Jagdhold for their valuable suggestions on the theoretical and architectural development of the CORDIC processor and C. Wolf, J. Lehman and M. Krstic for their help in test and measurement of the fabricated chip. Finally, the authors would like to thank the technology team of the IHP for fabricating the chip.

9 References

1 Volder, J.E.: 'The CORDIC trigonometric computing technique', *IRE Trans. Electron. Comput.*, 1959, **8**, (3), pp. 330–334
2 Walther, J.S.: 'A unified algorithm for elementary functions'. Proc. Joint Spring Computer. Conf., July 1971, vol. 38, pp. 379–385
3 Abruzzo, J.: 'Applicability of CORDIC algorithm to arithmetic processing'. Proc. 18th Asimolar Conf. on Circuits, Systems and Computers, 1985, pp. 79–86
4 Andrews, M., and Eggerding, D.A.: 'A pipelined computer architecture for unified elementary function evaluation', *Comput. Electr. Eng.*, 1978, **5**, (2), pp. 189–202
5 Cochran, D.S.: 'Algorithms and accuracy in the HP–35', *Hewlett-Packard J.*, 1972, pp. 10–11
6 Despain, A.M.: 'Very fast Fourier transform algorithms for implementation', *IEEE Trans. Comput.*, 1979, **28**, (5), pp. 333–341
7 Lee, D.T., and Morf, M.: 'Generalized CORDIC for digital signal processing'. Proc. Int. Conf. on Acoustics Speech and Signal Processing (ICASSP), May 1982, vol. 3, pp. 1748–1751
8 Mandal, M.C., Dhar, A.S., and Banerjee, S.: 'Multiplierless array architecture for computing discrete cosine transform', *Comput. Electr. Eng.*, 1995, **21**, (1), pp. 13–19
9 Dhar, A.S., and Banerjee, S.: 'An array architecture for fast computation of discrete Hartley transform', *IEEE Trans. Circuits Syst.*, 1991, **38**, (9), pp. 1095–1098
10 Maharatna, K., and Banerjee, S.: 'A VLSI array architecture for Hough transform', *Pattern Recognit.*, 2001, **34**, pp. 1503–1512
11 Maharatna, K., Dhar, A.S., and Banerjee, S.: 'A VLSI array architecture for realization of DFT, DHT, DCT and DST', *Signal Process.*, 2001, **81**, pp. 1813–1822
12 Deprettere, E., and Udo, R.: 'The pipelined CORDIC'. Internal Report Network Theory Section, Delft University of Technology, 1983
13 Ercegovac, M.D., and Lang, T.: 'Redundant and on-line CORDIC: application to matrix triangularization and SVD', *IEEE Trans. Comput.*, 1990, **38**, (6), pp. 725–740
14 Takagi, N., Asada, T., and Yajima, S.: 'Redundant CORDIC methods with a constant scale factor for sine and cosine computation', *IEEE Trans. Comput.*, 1991, **40**, (9), pp. 989–995
15 Hu, X., Harber, R.G., and Bass, S.C.: 'Expanding the range of convergence of the CORDIC algorithm', *IEEE Trans. Comput.*, 1991, **40**, (1), pp. 13–21
16 Wang, S., Puiri, V., and Swartzlander, E.E., Jr.: 'Hybrid CORDIC algorithms', *IEEE Trans. Comput.*, 1997, **46**, (11), pp. 1202–1207
17 Phatak, D.S.: 'Double step branching CORDIC: A new algorithm for fast sine and cosine generation', *IEEE Trans. Comput.*, 1998, **47**, (5), pp. 587–602
18 Hitotumatu, S.: 'Complex arithmetic through CORDIC', *Kodai Math. Seminar Rep.*, 1974, (26), pp. 176–186
19 Delosme, J.M.: 'VLSI implementation of rotations in pseudo-Euclidean spaces'. Proc. Int. Conf. on Acoustics Speech and Signal Processing (ICASSP), 1983, vol. 2, pp. 927–930
20 Muller, J.M.: 'Discrete basis and computation of elementary functions', *IEEE Trans. Comput.*, 1985, **34**, (9), pp. 857–862
21 Sung, T., Parng, T., Hu, Y., and Chou, P.: 'Design and implementation of a VLSI CORDIC processor'. Proc. IEEE Int. Symp. on Circuits and Systems (ISCAS), 1986, vol. 3, pp. 934–935
22 Hu, Y.H.: 'The quantization effects of the CORDIC algorithm', *IEEE Trans. Signal Process.*, 1992, pp. 834–844
23 Kota, K., and Cavallaro, J.R.: 'Numerical accuracy and hardware tradeoff for CORDIC arithmetic for special-purpose processors', *IEEE Trans. Comput.*, 1993, **42**, (7), pp. 769–779
24 Maharatna, K., Troya, A., Krstic, M., Grass, E., and Jagdhold, U.: 'A CORDIC like processor for computation of arctangent and absolute magnitude of a vector'. Proc. IEEE Int. Symp. on Circuits and Systems (ISCAS), 2004, pp. II -713–II-716
25 Maharatna, K., Troya, A., Banerjee, S., Grass, E., and Krstic, M.: 'A 16-bit CORDIC rotator for high-performance wireless LAN'. Proc. IEEE Personal Indoor and Mobile Radio Communication (PIMRC), Barcelona, Spain, 5–8 September 2004
26 Grass, E., Tittelbach, K., Jagdhold, U., Troya, A., Lippert, G., Krueger, O., *et al.*: 'On the single chip implementation of a Hiperlan/2 and IEEE802.11a capable modem', *IEEE Pers. Commun.*, 2001, **8**, (6), pp. 48–57