# A CORDIC LIKE PROCESSOR FOR COMPUTATION OF ARCTANGENT AND ABSOLUTE MAGNITUDE OF A VECTOR

*Koushik Maharatna*, Alfonso Troya**, Miloš Krstić**, Eckhard Grass** and Ulrich Jagdhold***

* Dept. of EE, University of Bristol, UK (Koushik.Maharatna@bristol.ac.uk)
** IHP, Frankfurt (Oder), Germany {troya, krstic, grass, jagdhold}@ihp-microelectronics.com

## ABSTRACT

In this paper, we propose a CoOrdinate Rotation DIgital Computer (CORDIC) like processor for computing absolute magnitude of a vector and its corresponding phase angle. It does not require the scale factor compensation step and addition/subtraction operation along the *z* datapath, has a convergence range over the entire coordinate space and shows similar error characteristics as that of the conventional CORDIC. The synthesis result shows that the proposed processor is hardware economic and suitable for low power applications.

## 1. INTRODUCTION

CORDIC algorithm is used for elegant computation of several transcendental functions [1]. Two such functions are the absolute magnitude of a vector and the corresponding phase angle (arctangent computation). These functions can be evaluated using the CORDIC in its angle accumulation or vectoring mode. In this case, the *y* component of the vector is forced to zero using iterative vector rotation in a *to and fro* manner through a set of elementary rotation angles. At the end, the magnitude value and the accumulated angle (the phase angle) are available as the *x* and *z* component of the output respectively. However, the main drawback of the traditional CORDIC algorithm is that it generates a scale factor that needs to be compensated using extra circuitry that incurs a computation complexity of the same order as that of the CORDIC itself. On top of that, several *not actually needed* iterations are performed while forcing the *y* component to zero.

In this paper we propose a similar type of algorithm and the corresponding VLSI architecture which eliminates the requirement of scale factor compensation, simplifies the angle accumulation operation along the *z* datapath and reduces the hardware cost significantly compared to that of the conventional CORDIC. The algorithm has a convergence range over the entire coordinate space. In essence, this algorithm is based on a *scaling free CORDIC* algorithm having a limited range of convergence proposed earlier [2, 3]. However, this algorithm is not as versatile as the CORDIC and is only comparable with its vectoring operation in the circular coordinate system. This work is resulted from a larger project that targets at a single chip implementation of IEEE 802.11a compatible modem. This new algorithm has been used for the synchronizer section of the targeted modem [4]. The rest of the paper is structured as follows: Section 2 describes the theory of the proposed algorithm, and Section 3 describes the VLSI implementation of the algorithm. The performance evaluation of the proposed scheme is done in Section 4 and conclusions are drawn in Section 5.

## 2. THEORETICAL BACKGROUND

In developing the algorithm, we will proceed in two steps: First we will show that a CORDIC with a convergence range of $[0, \pi/8]$ is absolutely sufficient to cover the entire coordinate space using a novel scheme called *domain folding* and second, we will use the *scaling free CORDIC* formulation described in the reference [2] in combination with first step to formulate the new algorithm.

### 2.1. Domain folding

We start with the assumption that the CORDIC has a convergence range $[0, \pi/8]$ and the initial vector lies in the first quadrant of the coordinate space. We divide the first quadrant into four domains namely, $A \in [0, \pi/8)$, $B \in [\pi/8, \pi/4)$, $C \in [\pi/4, 3\pi/8)$ and $D \in [3\pi/8, \pi/2]$. To check the appropriate domain in which the vector lies, we also consider two signals namely $x_{AB} = \sqrt{2}+1$ and $x_{CD} = \sqrt{2}-1$.

The vector to be actually processed i.e. $[x' \; y']^T$, is obtained after applying the pseudo-code shown in Figure 1 to the original input vector $[x \; y]^T$. This is done to keep the final accumulated angle in the range $[0, \pi/8]$. For vectors belonging to domain B, the final accumulated angle is subtracted from $\pi/4$ in order to get the actual phase angle. Similarly, for domain C, the final accumulated angle is added to $\pi/4$ in order to get the actual phase angle

whereas, for domain D the accumulated angle is subtracted from $\pi/2$.

```
if (x ≥ y) then
        if (x ≥ x_AB) then              -- Domain A
                x' = x;
                y' = y;
        else                            -- Domain B
                x' = (x + y)/√2;        -- pre-rotation by π/4
                y' = − (y − x)/√2;
        end if
else
        if (x < x_CD) then              -- Domain D
                x' = y;                 -- pre-rotation by π/2
                y' = x;
        else                            -- Domain C
                x' = (x + y)/√2;        -- pre-rotation by π/4
                y' = (y − x)/√2;
        end if
end if
```

**Figure 1**. Pseudo-code used in domain folding

One thing to be noted is that in this formulation, the range of convergence needed is always [0, $\pi/8$]. Thus, in essence, all the domains are "folded back" into domain A and hence the name *domain folding*.

It is straightforward to see that the same procedure is also applicable for the vectors lying in other quadrants. In this case, the input vector is first pre-rotated in the clockwise direction by appropriate angle, *viz.*, by $\pi/2$ when in 2nd quadrant, by $\pi$ when in 3rd quadrant and by $3\pi/2$ when in 4th quadrant. Then the operation proceeds as shown in Figure 1. This pre-rotation essentially means only changing of sign and swapping of the $x$ and $y$ components. At the output, the pre-rotated angle is added to the accumulated angle to get the final result. Thus, a CORDIC having a convergence range of [0, $\pi/8$] is sufficient to cover the entire coordinate space.

### 2.2. The scaling free CORDIC

The details of the scaling free CORDIC algorithm are provided in the reference [2, 3]. The working equation of the scaling free CORDIC can be given as

$$\begin{bmatrix} x1 \\ y1 \end{bmatrix} = \prod_{i=p}^{b-1} \begin{bmatrix} 1-2^{-(2i+1)} & 2^{-i} \\ -2^{-i} & 1-2^{-(2i+1)} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix}, \quad (1)$$

$$z_{i+1} = z_i + 2^{-i}, \quad (2)$$

where $[x1 \;\; y1]^T$ is the final vector, $[x_i \;\; y_i]^T$ is the intermediate vector at the beginning of $i^{th}$ iteration step, $b$ is the wordlength, $p = \lfloor (b - 2.585) / 3 \rfloor$ and $z_{i+1}$ is the angle accumulation variable at the end of $i^{th}$ iteration step ($z_0 = 0$). The block diagram of an elementary rotational section

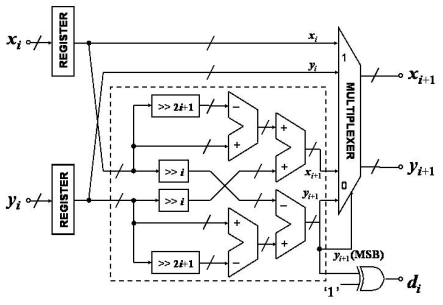


**Figure 2**. Elementary rotational stage of the scaling free CORDIC

resulting from Equation 1 is shown in Figure 2 by the dotted boundary.

Each of the elementary rotational stages of the scaling free CORDIC costs two adders and two shifters more compared to that of the conventional CORDIC. For pipeline implementation the shifters essentially reduce to wire connections only and thus the resulting overhead is just two adders. However, for the iteration index (elementary rotational section) $i \geq (b/2)-1$, the hardware cost of the rotational stages *will be the same* as that of the conventional one since a right shift by $(2i+1)$-bit results in machine zero or retention of sign bit only. Furthermore, since this formulation completely eliminates the requirement of scale factor compensation circuit, the overall hardware complexity of the scaling free CORDIC is less than the conventional one.

### 2.3. The new algorithm

The new CORDIC like algorithm for computing the absolute magnitude and phase angle of a vector can be constructed by utilizing the scaling free CORDIC algorithm in conjunction with the domain folding technique. The complete algorithm can be summarized as follows:

1. *Detect the quadrant in which the vector lies*: This can be easily detected by checking the MSB of the input parameters $x$ and $y$.
2. *Modify the input vector*: This step should be done by following the domain folding technique described in Figure 1. The aim of this operation is to bring the actual angle to be accumulated within the range [0, $\pi/8$].
3. *Use scaling free CORDIC in vectoring mode*: This step corresponds to the actual angle accumulation operation and can be carried out as in the conventional CORDIC.
4. *Output generation*: The correct output can be generated by following the rules described in subsection 2.1.

Under an implementation point of view, further optimization can be done by only considering one-sided vector rotation instead of to and fro motion of the vector. Rotating the vector in one single direction essentially means that the accumulated angle can be described as a *pure summation* of powers of two. In this process, the *not actually needed iteration steps* are to be skipped. The final accumulated angle can be described by a bit pattern that contains logic '1' corresponding to the allowed iteration steps and logic '0' corresponding to the not allowed iteration steps. In essence, this technique eliminates *all the required addition/subtraction operation* along the $z$ datapath and reduces the hardware cost drastically. This process can be summarized as follows:

1. Compute the intermediate vector at $i^{th}$ iteration step.
2. If $y_{i+1} < 0$ then assign $x_{i+1} = x_i$ and $y_{i+1} = y_i$ and enter a logic '0' ($d_i$ in Figure 2) in the appropriate position of the $z$ datapath register. This operation essentially means that the $i^{th}$ iteration is ignored.
3. If $y_{i+1} > 0$ then assign $x_{i+1} = x_{i+1}$ and $y_{i+1} = y_{i+1}$ and enter a logic '1' ($d_i$ in Figure 2) in the appropriate position of the $z$ datapath register. This operation essentially means that the $i^{th}$ iteration is accepted.
4. Take the binary value of the $z$ datapath register when $y_{i+1} = 0$ (this is the final accumulated angle) and process it to generate the final output value following the rules described in subsection 2.1.

Considering these modifications, the final structure of an elementary rotational stage is as shown in Figure 2.

## 3. ARCHITECTURE AND IMPLEMENTATION

The complete architecture of the proposed processor consists of three modules *viz.* the *Domain Detection Circuit*, *Basic CORDIC Pipeline* and *Output Unit*. For convenience, we describe a 16-bit fixed-point pipeline implementation of the proposed Processor. Two's complement arithmetic is used throughout the implementation.

The *Domain Detection Circuit* is responsible for detecting the appropriate quadrant and the corresponding domain in which the vector lays. It consists of three comparators, two adders and a scaling circuit of $\sqrt{2}$. The scaling circuit is realized using shift-and-add technique and thus, it is more economical compared to a full multiplier. It generates two 2-bit signals namely *quad* and *domain*. While the *quad* signal indicates the initial quadrant in which the vector lays, the *domain* signal indicates the domain in the first quadrant where it is folded back.

The *Basic CORDIC Pipeline* has a convergence range of $[0, \pi/8]$. For a 16-bit implementation, the value of $p$ is 4 (see subsection 2.2). Thus, the largest right shift allowed in this formulation is by 4 bits. In order to cover the convergence range of $[0, \pi/8]$, we have used the $i = 4$ stage six times and $i = 5, 6, \ldots, 14$ stages once each. The stage $i = 15$ is omitted since the right shift of a number by 15-bit position essentially results in the retention of the sign bit only.

The architecture of the basic CORDIC pipeline is shown in Figure 3. Each of the pipeline stages corresponding to $i = 4$, 5 and 6 requires four adders. On the other hand, stages $i = 7, 8, \ldots, 14$ require two adders each. In order to balance the pipeline, the stages $i = (7, 8)$, (9, 10), (11, 12) and (13, 14) have been merged as shown by the dotted boundary in Figure 3, hence reducing the total length of the pipeline to 12 stages (index $j$ in Figure 3). An array of registers is associated with different pipeline stages to keep the intermediate binary values of the accumulated angle. Depending on the decision of a particular stage, i. e., whether a rotation operation is accepted or rejected, logic '1' or '0' is entered at the LSB position of the register array and the value is passed to the next stage as shown in Figure 3. However, a simple combinatorial circuit is necessary to interpret the decisions made by the six $i = 4$ stages. The decisions made in these sections give the 3 MSB of the final representation of the accumulated angle. At the end, the basic CORDIC pipeline generates a 13-bit unsigned value for the accumulated angle φ, which can be further processed by the output unit according to the principle stated in subsection 2.1. The absolute magnitude of the vector is available at $x$ output.

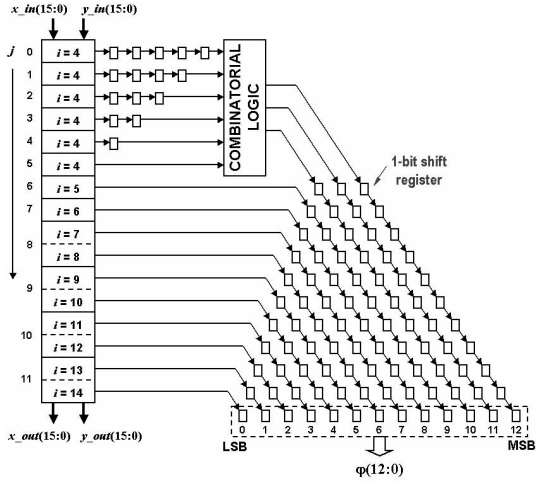


**Figure 3**. Architecture of the basic CORDIC pipeline.

The *domain* and *quad* signals generated by the *Domain Detection Circuit* flow through the pipeline along with the data (not shown in Figure 3). Thus, it can be viewed as if each of the data has a *token* attributed to it that essentially carries the information about its initial quadrant and domain which can be processed by the output unit to generate the final result.

The main hardware of the *Output Unit* consists of an adder and some registers. Depending on the *domain* and *quad* signals, it generates the final phase angle by following the procedure described in subsection 2.1.

The synthesized cell area of the complete processor in IHP 0.25 µm BiCMOS technology library is 0.5 $mm^2$ (16 k inverter gates). The *Domain Detection Circuit*, the *Basic CORDIC Pipeline* and the *Output Unit* occupy 0.088$mm^2$, 0.411$mm^2$ and 0.009$mm^2$, respectively. The power consumption of the processor is 6 mW.

## 4. PERFORMANCE EVALUATION

### 4.1. Error analysis

The error performance of the algorithm is shown in Figures 4 and 5. The algorithm is modeled in Matlab and then the value of $x$ and $y$ inputs are varied in the range [0, 1]. Figures 4 and 5 show that the proposed algorithm shows similar error characteristic than that of the conventional CORDIC algorithm. For the values of $x$ and $y > 1$ or $<< 1$, the angle computation error is too high to be acceptable.

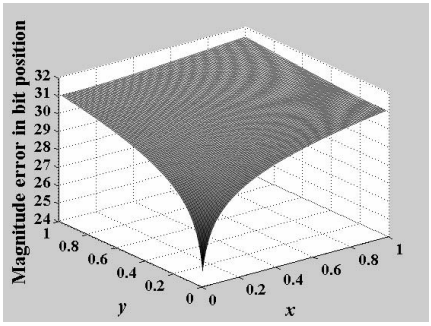


**Figure 4**. Error in magnitude calculation.

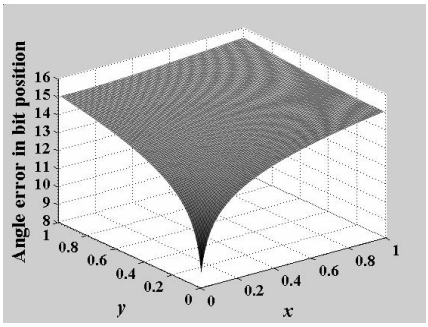


**Figure 5**. Error in angle calculation.

### 4.2. Hardware complexity

A comparison of the hardware complexity of the proposed design with some other CORDIC processors operating in the vectoring mode is provided in Table 1. It shows that when the scaling circuitry is considered, the proposed design requires less hardware compared to the other designs.

| CORDIC type | # full adders | # registers | scaling |
|---|---|---|---|
| [5] | 1,280 | 3,114 | YES |
| [6] | 512 | 1,280 | YES |
| Conventional | 768 | 768 | YES |
| Proposed | 816 | 553 | NO |

**Table 1**. Comparison of the proposed design with some other existing designs (16-bit implementation).

## 5. CONCLUSIONS

In this paper, we propose a CORDIC like algorithm for computing the magnitude and phase of a vector. A 16-bit VLSI implementation is also addressed. The proposed algorithm does not need the scale factor compensation step. Its hardware cost is less than that of the conventional CORDIC when the scale factor compensation circuitry is taken into account. The complete elimination of the arithmetic processing for the $z$ datapath makes it an attractive one from the hardware cost and low power application point of view. The algorithm proposed here shows similar error characteristic to that of the conventional CORDIC. The synthesis results show that the proposed design occupies a very small area and consumes very low power.

## 6. REFERENCES


[1] J. S. Walther, "A Unified Algorithm for Elementary Functions", *Proc. Joint Spring Comput. Conf.*, vol. 38, pp. 379 – 385, Jul. 1971.

[2] E. Grass, B. Sarker and K. Maharatna, "A Dual Mode Synchronous/Asynchronous CORDIC Processor", *Proc. 8th IEEE International Symposium on Asynchronous Circuits and Systems*, pp. 76 – 83, Manchester, U. K., April 2002.

[3] K. Maharatna, A. S. Dhar and Swapna Banerjee, "A VLSI Array Architecture for Realization of DFT, DHT, DCT and DST", *J. Signal Processing*, vol. 81, pp. 1813 – 1822, 2001.

[4] M. Krstic, A. Troya, K. Maharatna and E. Grass, "Optimized Low-Power Synchronizer Design for the IEEE 802.11a Standard", *Proc. ICASSP*03, vol. II, pp. 321 – 324.

[5] H. Dawid and H. Meyr, "The Differential CORDIC Algorithms: Constant Scale Factor Redundant Implementation without Correcting Iterations", *IEEE Trans. Comput.*, vol. 45, no. 3, pp. 307 – 318, 1996.

[6] D. Timmermann and S. Dolling, "Unfolded Redundant CORDIC VLSI Architecture with Reduced Area and Power Consumption", http://www-md.e-technik.uni-rostock.de/ma/dtim/vlsi97.pdf