Original scientific paper

Informacije MIDEM
Journal of Microelectronics,
Electronic Components and Materials
Vol. 46, No. 4(2016), 197 – 208

Area and Energy efficient CORDIC Accelerator for Embedded Processor Datapaths

Abdul Rehman Buzdar[1], Liguo Sun[1], Shoab Ahmed Khan[2], Abdullah Buzdar[1]

[1]Department of Electronic Engineering and Information Science, University of Science and Technology of China (USTC), Hefei, China
[2]Department of Computer Engineering, National University of Sciences and Technology (NUST), Islamabad, Pakistan

Abstract: A proven approach to enhance the performance of an embedded processor is to add specialized hardware accelerator blocks. We present two novel CORDIC accelerator units based on mixed hardware/software approach. These CORDIC accelerators are integrated with an embedded processor datapath to enhance the processor performance in terms of execution time and energy efficiency. The first accelerator design is based on the Standard CORDIC algorithm. The Standard CORDIC based accelerated embedded processor datapath is 35% more cycle efficient than a datapath lacking Standard CORDIC accelerator. This design also leads to 34% energy reduction. The mixed hardware/software implementation of Standard CORDIC algorithm is area efficient as it saves two 16-bit adders. The second accelerator design is based on a Modified CORDIC algorithm. Our evaluation shows that a Modified CORDIC accelerated embedded processor datapath is 14.5 times more cycle efficient than a datapath lacking Modified CORDIC accelerator. This design leads to 14 times energy reduction with a very small area overhead. The mixed hardware/software Modified CORDIC accelerator is area efficient as it saves four multipliers and two adders. The Modified CORDIC hardware accelerator block has 4.3 times less latency and takes 4 times less area as compared to Standard CORDIC Time Shared implementation. The novelty of the design in the use of Modified CORDIC accelerator is that it takes a single iteration to compute the values of sine and cosine as compared to the Standard CORDIC algorithm, which requires N iterations. This provides effective use of the accelerator in programming systems where a series of values of sine and cosine are required to be computed.

Keywords: CORDIC; Accelerator; Codesign; FPGA; MicroBlaze Processor

Prostorsko in energijsko učinkovit CORDIC pospeševalnik za podatkovne poti vgrajenega procesorja

Izvleček: Dodajanje specializiranih pospeševalnih blokov v vgrajen procesor je uveljaljena metoda povečevanja njegove učinkovitosti. Predstavljamo dve novi pospeševalni enoti za CORDIC na osnovi mešane programsko strojne rešitve. Ti pospeševalniki so integrirani v podatkovne poti procesorja za zagotavljanje krajšega izvajalnega časa in energijske učinkovitosti. Prvi pospeševalnik temelji na standardnem CORDIC algoritmu in omogoča 35 % višjo učinkovitost cikla kot brez njegove uporabe. Poraba energije je 34 % nižja. Programsko/strojno mešana implementacija je prostovno učinkovita in prihrani dva 16-bitna seštevalnika. Drugi pospeševalnik temelji na modificiranem CORDIC algoritmu. Vrednotenje modificiranega algoritma je pokazalo 14.5 kratno izboljšanje učinkovitosti cikla. Istočasno se je za 14 krat zmanjšala poraba energije. Programsko/strojno mešana rešitev prihrani štiri množilnik ein dva seštevalnika. Strojno izveden modificiran CORDIC pospeševalnik ima 4.3 krat manjšo latenco in potrebuje 4 krat manj prostora kot standardna CORDIC rešitev s delitvijo časa. Prednost modificiranega CORDIC pospeševalnika je, da potrebuje le eno iteracijo za izračun sinusa in kosinusa v primerjavi s standardnim CORDIC algoritmom, ki potrebuje N iteracij. To omogoča njegovo učinkovito uporabo v programskih sistemih s potrebo po računanju velikega števila izračunavanja funkcij sinus in kosinus.

Ključne besede: CORDIC; pospeševalnik; Codesign; FPGA; MicroBlaze procesor

** Corresponding Author's e-mail: liguos@ustc.edu.cn ; abdul.buzdar@alumni.chalmers.se*

1 Introduction

The CORDIC (Coordinate Rotation Digital Computer) algorithm first introduced by Jack E.Volder [1], [2] in 1959 is used for the computation of trigonometric functions, multiplication, and division. It was extended further by John Walther [3], [4] in 1971 for the computation of a wide range of elementary functions such as logarithms, exponentials, and square roots. During the same period, Cochran [5] showed that the CORDIC algorithm is a suitable technique for scientific calculator implementation. CORDIC algorithm is used in a broad range of areas including signal processing, communication systems, robotics and computer graphics. During the past 50 years, a lot of research has been carried out on CORDIC in the area of algorithm and architecture design to achieve high performance and area efficient hardware solutions [6-8]. Angle recording CORDIC [9] solves the repetitive rotation issue of Standard CORDIC by recoding the latest inserted item into the angle set. This technique is helpful in the implementation of Discrete Fourier Transform and Discrete Cosine Transform but has a drawback of unpredictable scale factor [10]. Extended Elementary Angle Set (EEAS) CORDIC uses searching techniques such as Greedy Searching and Trellis-based Searching Algorithm (TBS) to find the required angle from an angle set [11], [12]. Pipelined CORDIC architectures [13-15] are widely implemented in digital signal processing for sine wave generation, orthogonal discrete transform, and adaptive filtering. Radix-4 [16], [17] and BCD [18], [19] CORDIC architectures are applied in situations where high precision is required. Vachhani et al. [20] implemented CORDIC design by eliminating ROM and barrel shifters, resulting in huge resource reduction. CORDIC architecture with reduced ROM has also been reported in [21]. Aggarwal et al. [22] implemented Scale-free hyperbolic CORDIC processor for waveform generation. Caro et al. [23] implemented digital synthesizer/mixer with hybrid CORDIC multiplier architecture.

CORDIC algorithm is used in many real-time applications including direct digital frequency synthesis (DDFS) having critical latency issue. The Standard CORDIC algorithm has fixed latency in which the number of iterations is directly proportional to bit precision. A lot of research has been carried out in past to improve the latency of CORDIC for the calculation of sine and cosine of an angle. Rodrigues and Swartzlander [24] proposed a 50% reduced iterative CORDIC algorithm, by using dynamic angle selection which recodes the angle. Aytore and Alkar [25] used additional logic and control circuitry to reduce the number of iterations, by involving diversified iterations. Hu and Naganathan [26] also proposed a 50% reduced iterative CORDIC algorithm, but it works only for fixed number of angles which should be known in advance. Higher Radix CORDIC algorithms have also been used for the reduction in iterations. Antelo et al. [27] proposed a radix-4 representation for σ_i in which the rotations are chosen from a set {+2,+1,0,-1,-2} of four possible iterations, but has more area overhead. Phatak et al. [28] proposed a double rotation technique in which the values of σ_i and σ_{i+1} are set using a prediction technique. Parallel angle coding is also reported in the literature at the cost of an increase in micro-rotations compared to Standard CORDIC algorithm [29-31]. Kao et al. [32] proposed the use of encoded angle to directly compute the initial iterations using look-ahead approach at the cost of area overhead. Kamboh and Shoab [33], [34] proposed an IS-CORDIC architecture which computes the values of sine and cosine in a single cycle.

The CORDIC algorithm can be implemented in software on an embedded processor. But software solutions running on a processor takes a lot of clock cycles compared to the dedicated hardware implementations. The research on the sources of inefficiency in various applications showed that 90% of the program runtime and energy is utilized by only 10% of application code [46]. This small portion of the applications which becomes a performance bottleneck can be efficiently managed by implementing them in hardware as specialized accelerator blocks [35-38]. High data rates in modern signal processing and communication systems can only be delivered by dedicated hardware solutions. Today's embedded systems use processor core with different hardware accelerators in order to speed up certain portions of the application code. This heterogeneous approach reaps the benefits of programmability of an embedded processor and efficiency of specialized hardware accelerator blocks. Most of the embedded processors today contain various dedicated hardware blocks to perform a wide range of communication system tasks efficiently. These include MAC, TCP/IP, Ethernet, CRC, and CAN etc. Implementing CORDIC as a hardware accelerator will be effective in programming systems where a series of values of sine and cosine are required to be computed.

The rest of the paper is organized as follows: in the next section, the theoretical background of Standard CORDIC algorithm is presented. Later, we describe the implementation of hardware accelerator based on the Standard CORDIC algorithm. Subsequently, we describe a Modified CORDIC algorithm and we use this technique to implement a more efficient CORDIC accelerator. Finally, we summarize our conclusions.

2 Standard CORDIC Algorithm

In Standard CORDIC algorithm we start with a unit vector and rotate it to the desired angle θ. When the unit vector reaches the desired angle the x and y co-ordinates of the unit vector give us cos θ and sin θ, respectively. Mathematically, this can be shown with the expression below.

$$\theta = \sum_{i=0}^{N-1} \sigma_i \Delta\theta \quad where \quad \sigma_i = \begin{cases} +1 & \text{for positive rotation} \\ -1 & \text{for negative rotation} \end{cases} \tag{1}$$

First, the unit vector is rotated by an angle θ_i and then by an angle $\Delta\theta_i$ again. This brings the unit vector to angle $\Delta\theta_{i+1}$, as depicted in Fig. 1. Mathematically, this can be expressed [34] by Equations (1) and (2).

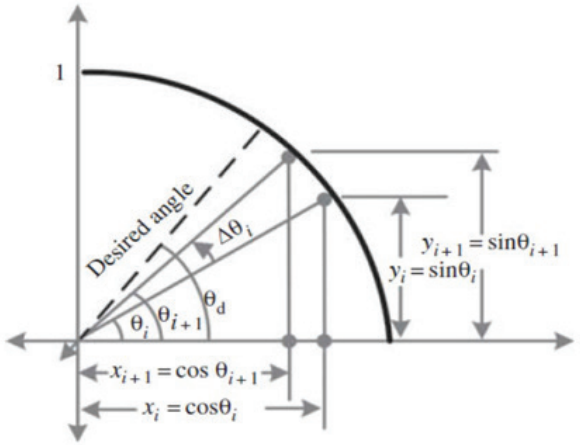

Figure 1: Standard CORDIC algorithm iterations

$$\cos\theta_{i+1} = \cos(\theta_i + \sigma_i\Delta\theta_i) = \cos\theta_i \cos\Delta\theta_i - \sigma_i \sin\theta_i \sin\Delta\theta_i \tag{2}$$

$$\sin\theta_{i+1} = \sin(\theta_i + \sigma_i\Delta\theta_i) = \sin\theta_i \cos\Delta\theta_i + \sigma_i \cos\theta_i \sin\Delta\theta_i \tag{3}$$

From Fig. 1. we can see that $x_i = \cos\theta_i$, $y_i = \sin\theta_i$ and similarly $x_{i+1} = \cos\theta_{i+1}$, $y_{i+1} = \sin\theta_{i+1}$

Substituting these in Equations (2) and (3), we get Equations (4) and (5) as given below.

$$x_{i+1} = x_i \cos\Delta\theta_i - \sigma_i y_i \sin\Delta\theta_i \tag{4}$$

$$y_{i+1} = \sigma_i x_i \sin\Delta\theta_i + y_i \cos\Delta\theta_i \tag{5}$$

Equations (4) and (5) can be expressed in matrix form as

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} \cos\Delta\theta_i & -\sigma_i \sin\Delta\theta_i \\ \sigma_i \sin\Delta\theta_i & \cos\Delta\theta_i \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{6}$$

By taking $\cos\Delta\theta_i$ common, we get Equation (7)

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \cos\Delta\theta_i \begin{bmatrix} 1 & -\sigma_i \tan\Delta\theta_i \\ \sigma_i \tan\Delta\theta_i & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{7}$$

By using the trigonometric identity

$$\cos\Delta\theta_i = \frac{1}{\sqrt{1+\tan^2\Delta\theta_i}}$$

To avoid multiplication we get Equation (8)

$$\tan\Delta\theta_i = 2^{-i} \tag{8}$$

Equation (8) can also be expressed as $\Delta\theta_i = \tan^{-1}2^{-i}$.

Substituting Equation (8) in (7) we get the following Equation.

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \frac{1}{\sqrt{1+2^{-2i}}} \begin{bmatrix} 1 & -\sigma_i 2^{-i} \\ \sigma_i 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{9}$$

Let,

$$k_i = \frac{1}{\sqrt{1+2^{-2i}}} \quad , \quad \sigma_i = \begin{bmatrix} 1 & -\sigma_i 2^{-i} \\ \sigma_i 2^{-i} & 1 \end{bmatrix}$$

Then Equation (9) can be expressed as

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = K_i \sigma_i \begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{10}$$

Initially, the index i=0, thus the Equation (10) can be given for i=0 as

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = K_0 R_0 \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{11}$$

and for index i=1, we have

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = K_1 R_1 \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \tag{12}$$

Substituting the value of $\begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$ from Equation (11) into (12) we get

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = K_0 K_1 R_0 R_1 \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{13}$$

Thus Equation (10) for indices i=N-1 becomes

$$\begin{bmatrix} x_N \\ y_N \end{bmatrix} = K_0 K_1 K_2 ... K_{N-1} R_0 R_1 R_2 ... R_{N-1} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \quad (14)$$

All the K_i are constants and their product can be computed as a constant k, so we get

$$K = K_0 K_1 K_2 ... K_{N-1} = \prod_{i=0}^{N-1} \frac{1}{\sqrt{1+2^{-2i}}} \quad (15)$$

Finally we get Equation (16) as given below

$$\begin{bmatrix} x_N = \cos\theta \\ y_N = \sin\theta \end{bmatrix} = R_0 R_1 R_2 ... R_{N-1} \begin{bmatrix} K \\ 0 \end{bmatrix} \quad (16)$$

2.1 Hardware Mapping of Standard CORDIC

For efficient hardware implementation the Standard CORDIC algorithm is listed as follows:

- To simplify the hardware θ_0 is set to the desired angle θ_d and θ_1 is computed as given below

$$\theta_1 = \theta_0 - \sigma_0 \tan^{-1} 2^0$$

Where, σ_0 is the sign of θ_0 and initialize as $x_0 = k$ and $y_0 = 0$.

- The algorithm then performs N iterations for i = 1,2,...,N-1 and computes the following set of Equations

$$if\,(\theta_i > 0) \quad \sigma_i = 1 \quad else \quad \sigma_i = -1$$

$$x_{i+1} = x_i - \sigma_i 2^{-i} y_i \quad (17)$$

$$y_{i+1} = y_i + \sigma_i 2^{-i} x_i \quad (18)$$

$$\theta_{i+1} = \theta_i - \sigma_i \tan^{-1} 2^{-i} \quad (19)$$

All the values for $\tan^{-1}2^{-i}$ are precomputed and stored in an array.

- The final iteration generates the desired results given below in the two equations

$$\cos\theta_d = x_N$$

$$\sin\theta_d = y_N$$

The Standard CORDIC algorithm is naturally suitable for hardware mapping. The *ith* iteration of the algorithm can be implemented as a CORDIC Processing Element (PE), shown in Fig. 2. The CORDIC PE implements the Equations (17), (18) and (19) of Standard CORDIC algorithm in hardware and its internal implementation

Figure 2: Standard CORDIC Processing Element (PE)

is shown in Fig. 3. These CORDIC PEs can be cascaded together for a fully parallel hardware implementation of Standard CORDIC algorithm as shown in Fig. 4. Depending on the number of cycles available for computing sine and cosine values the Standard CORDIC algorithm can also be folded and implemented as a time-shared architecture with these CORDIC PEs.

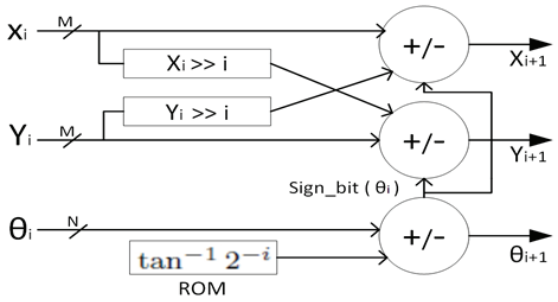

Figure 3: Internal implementation of Standard CORDIC PE

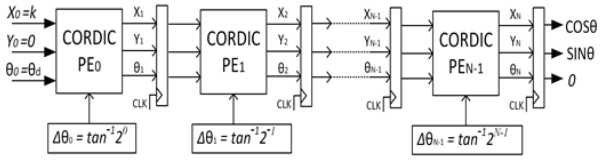

Figure 4: Pipelined fully parallel architecture of Standard CORDIC algorithm

2.2 Standard CORDIC Hardware Accelerator

We have developed a novel mixed hardware/software CORDIC accelerator unit using the Standard CORDIC algorithm. Equations (17) and (18) of Standard CORDIC algorithm are implemented in hardware using Verilog HDL hardware description language. While Equation (19) is implemented in software. We used Xilinx Spartan-6 FPGA SP605 Evaluation Kit [41] and Xilinx Embedded Development Kit (EDK) [39] for the implementation. Xilinx Microblaze soft core processor system [40] is used to execute the software part of Standard CORDIC accelerator. There are two ways to integrate a hardware accelerator core into a MicroBlaze based embedded processor system. One way is to connect the accelerator through the Processor Local Bus (PLB). The second way is to connect it using a dedicated Fast Simplex Link (FSL) bus system [42]. First, PLB was tried but

it was taking more cycles. The reason for this is the fact that it is a traditional memory-mapped transaction bus. Later, it was decided to integrate our Standard CORDIC accelerator block with the MicroBlaze processor system using a dedicated FIFO style FSL bus as shown in Fig. 5.

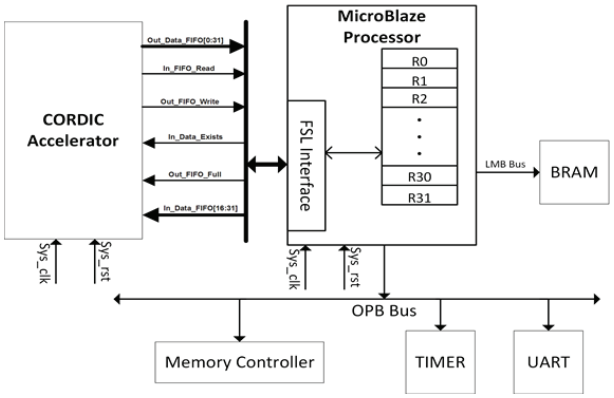

Figure 5: CORDIC Accelerator with MicroBlaze Processor System

First, the Standard CORDIC algorithm was implemented in C programming language. It was executed on MicroBlaze processor using Xilinx Software Development Kit (SDK) [39]. The cycle count for the software implementation of Standard CORDIC algorithm was measured using the XPS hardware timer block. The MicroBlaze processor takes 933 cycles to compute the values of sine and cosine. While executing the complete software implementation of Standard CORDIC algorithm. In the next step, Verilog HDL code of the hardware part of Standard CORDIC accelerator was implemented. It was verified and synthesized using Xilinx ISE design suit [39]. Table 1 shows the Synthesis results of Time Shared Standard CORDIC algorithm and Standard CORDIC hardware accelerator block. Standard CORDIC algorithm having N iterations has a latency of N times the delay of a single iteration. Here, N represents the internal word length. The Time Shared Standard CORDIC algorithm and Standard CORDIC hardware accelerator block were synthesized on 7vx485tffg1157-3 Virtex-7 FPGA device. This FPGA device uses a 28nm technology

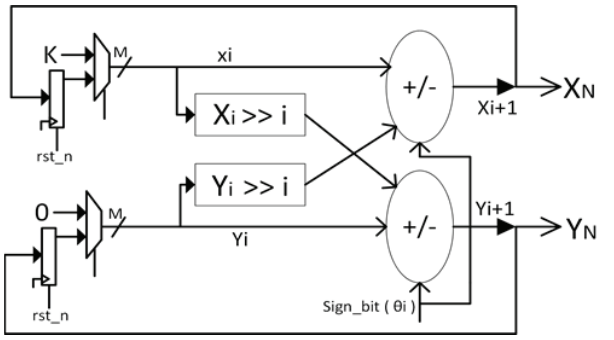

Figure 6: Standard CORDIC Hardware Accelerator Block

and gives a critical path delay of 51.52ns and 51.408ns respectively. As in the case of Standard CORDIC hardware accelerator block we have implemented θ_i Table in software. Thus no RAM is used. The mixed hardware/software implementation of Standard CORDIC algorithm is area efficient as it saves two 16-bit adders as shown in Table 1.

Table 1: Synthesis results of Time Shared Standard CORDIC algorithm and Standard CORDIC Accelerator

	Time Shared Standard CORDIC	Standard CORDIC Accelerator
Max Freq	(310/16)=19.3 MHz	(311/16)=19.4 MHz
Latency	3.220x16 = 51.52ns	3.213x16 = 51.408ns
RAMs	16x16-bit RAM	0
Adders	2x16-bit, 4x22-bit	4x22-bit
Counters	1x4-bit	1x4-bit
Multiplexers	6	4
Logic Shifters	2	2
Slice Registers	108	92
Slice LUTs	344	282
Slices	148	139

Fig. 6 shows the architecture of Standard CORDIC hardware accelerator unit. The Standard CORDIC accelerator was attached with the Microblaze processor system via FSL bus using Xilinx Platform Studio (XPS) [39]. Later the software part of Standard CORDIC accelerator was implemented in C programming using Xilinx SDK. The predefined C functions of SDK were used to communicate with hardware part of Standard CORDIC accelerator via FSL bus. In the software part of Standard

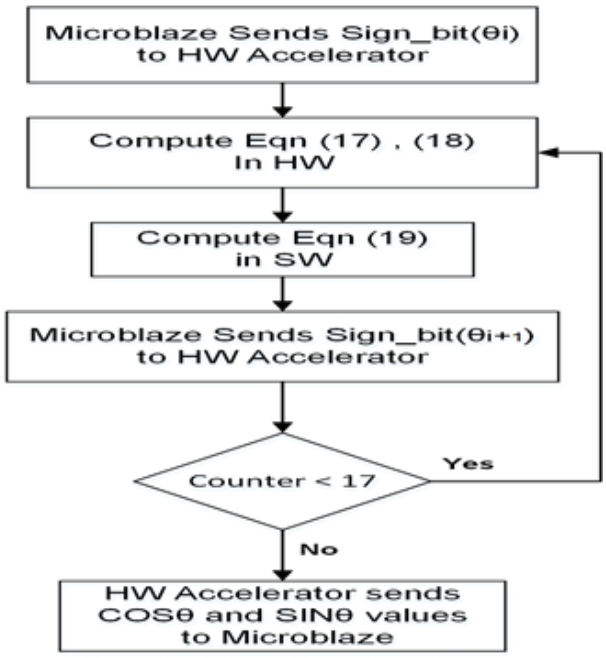

Figure 7: Flow chart for Standard CORDIC Accelerator Implementation

CORDIC accelerator the values for $\tan^{-1}2^{-i}$ are precomputed and stored in an array. Equation (19) is executed for every iteration of Standard CORDIC algorithm using C programming language. The sign bit of θ_i is sent via FSL bus to the CORDIC hardware accelerator which executes Equations (17) and (18) in hardware. For every iteration of Standard CORDIC algorithm as shown in Fig.3. After the final iteration, we get values of sine and cosine via FSL bus from the Standard CORDIC accelerator by using predefined C functions in SDK. Fig.7 shows the steps involved in the computation of sine and cosine using mixed hardware/software Standard CORDIC accelerator.

The cycle count for mixed hardware/software implementation of Standard CORDIC algorithm was measured using the XPS hardware timer block. The mixed hardware/software implementation of Standard CORDIC algorithm takes 601 cycles to compute the values of sine and cosine. The energy dissipation was calculated for both the implementations, shown in Table 3. Our evaluation shows that an accelerated embedded processor datapath is 35% more cycle efficient, than a datapath lacking Standard CORDIC accelerator. The design also leads to 34% energy reduction. This mixed hardware/software implementation is also area efficient as we implemented the Equation (19) of Standard CORDIC algorithm in software on a MicroBlaze processor, which resulted in saving two 16-bit adders. Obviously, if we implement all the three Equations (17), (18) and (19) of Standard CORDIC algorithm in hardware it will be a faster solution. Thus it's a trade-off between area and execution time.

3 Modified CORDIC Algorithm

The Standard CORDIC algorithm is dependent on σ_i for making a decision of whether to do addition or subtraction, in Equations (17), (18) and (19). This algorithmic limitation is the reason for taking more cycles for computation of desired results. To make this algorithm fast and suitable for parallel implementation we need to make some modifications in the Standard CORDIC algorithm. In Standard CORDIC algorithm we assumed that θ is the summation of N positive and negative micro-rotations of angles $\Delta\theta_i$ as shown in Equation (1). The θ can also be represented in a binary form for micro-rotations [34] as shown in Equation (20) below

$$\theta = \sum_{i=0}^{N-1} b_i 2^{-i} \quad for \quad b_i \in \{0,1\} \tag{20}$$

Here, bit b_i decides between a positive rotation of 2^{-i} or a zero rotation, for each term in the summation. To make this expression useful for hardware implementation we need to make the constant K in Equation (15) data independent by recoding the Equation (20) to only use +1 or -1. For fixed point implementation of CORDIC, the desired angle θ_d is represented as $\theta_{1.\,N-1}$. Here the most significant bit (MSB) is used for representing the sign of integer value and *N* - 1 bits are set aside for fractional part of N-bit θ. The expression (20) can be represented after recording by Equation (21) as

$$\sum_{i=0}^{N-1} b_i 2^{-i} = \sum_{i=0}^{N-1} r_i 2^{-(i+1)} + 2^{-0} - 2^{-N} r_i = 2b_i - 1, \quad where \quad r_i \in \{0,1\} \tag{21}$$

To manage the constant factor (2^{-0} -2^{-N}) in the recoding of Equation (21), an initial fixed rotation Q_{init} is given. The recoding of $b_i s$ as ±1 helps in making K a constant and its value is equal to [34].

$$K = \prod_{i=0}^{N-1} \cos(2^{-1})$$

The initial rotation is applied first offline given below by the three equations

$$Q_{init} = (2^{-0} - 2^{-N})$$

$$x_0 = k\cos(Q_{init})$$

$$y_0 = k\sin(Q_{init})$$

The following equations are computed for i = 1,2,3 .... N-1 iterations as

$$x_{i+1} = x_i - r_i \tan^{-1} 2^{-i} y_i$$

$$y_{i+1} = r_i \tan^{-1} 2^{-i} x_i + y_i$$

Here, the values of r_i are precomputed. Unlike σ_i, these iterations don't need to compute $\Delta\theta_i$ as was required in Standard CORDIC algorithm. The final iteration generates the desired results as

$$\cos\theta_d = x_N$$

$$\sin\theta_d = y_N$$

One issue in modified CORDIC algorithm which needs to be solved is the elimination of multiplication by tan 2^{-i} in every iteration. As $\tan\theta \approx \theta$ for small values of θ, this results in converting multiplication into simple shift by 2^{-i}. So we get

$$\tan 2^{-i} \approx 2^{-i} \quad for \quad i > 4 \tag{22}$$

This approximation does not affect the precision of desired output results [29], [44]. We can precompute the values for the first four iterations and store them in a ROM. In the hardware implementation of the algorithm, we can use these precomputed values for initial M iterations from a ROM. The ROM address for these

precomputed values is calculated using M most significant bits (MSBs) of θ as given below

$$index = \theta_0 2^{M-1} + \theta_1 2^{M-2} + ... + \theta_{M-1} 2^0 \quad (23)$$

x[M-1] and y[M-1] values are accessed from ROM and the remaining values of x[k] and y[k] are computed with the help of approximation of Equation (22). This results in converting multiplication by tan 2^{-i} into simple shift by 2^{-i}. This transformation helps in fully parallel hardware implementation of the algorithm for better performance. We can combine various iterations in the CORDIC algorithm to increase the performance and reduce the hardware [45]. As the iterations are not dependent on the values of $\Delta\theta_i$ in the modified CORDIC. Thus, we can substitute the values of previous iterations into the current iteration. For M=4 indexing into the tables, we get values of x_4 and y_4. Substituting these values for i=5, we get Equations (24) and (25) as given below

$$x_5 = x_4 - r_5 2^{-5} y_4 \quad (24)$$

$$y_5 = r_5 2^{-5} x_4 - y_5 \quad (25)$$

For i=6, we get Equations (26) and (27) as provided below

$$x_6 = x_5 - r_6 2^{-6} y_5 \quad (26)$$

$$y_6 = r_6 2^{-6} x_5 - y_5 \quad (27)$$

Substituting the expressions for x_5 and y_5 from Equations (24) and (25) into Equations (26) and (27), we get the following equations

$$x_7 = (1 - r_5 r_6 2^{-11} - r_5 r_7 2^{-12} + r_6 r_7 2^{-13}) x_4 - (r_5 2^{-5} - r_6 2^{-6} + r_7 2^{-7} - r_5 r_6 r_7 2^{-18}) y_4 \quad (28)$$

$$x_7 = (r_5 2^{-5} - r_6 2^{-6} + r_7 2^{-7} - r_5 r_6 r_7 2^{-18}) x_4 + (1 - r_5 r_6 2^{-11} - r_5 r_7 2^{-12} + r_6 r_7 2^{-13}) y_4 \quad (29)$$

The terms 2^{-k} with $k > P$ for a P-bit data path makes the expressions (28) and (29) outside the required precision and can be discarded. Ignoring these terms and substituting previous equations into current iteration we get the value x_N and y_N, expressed in terms of x_4 and y_4 [33, 34]. For P=16, we have

$$x_{16} = x_4 + \sum_{n=5}^{17} r_n 2^{-n} y_4 + \sum_{n=11}^{17} r_5 r_{n-4} 2^{-n} x_4 - \sum_{n=13}^{17} r_6 r_{n-4} 2^{-n} x_4 - \sum_{n=15}^{17} r_7 r_{n-4} 2^{-n} x_4 \quad (30)$$

$$y_{16} = y_4 - \sum_{n=5}^{17} r_n 2^{-n} x_4 - \sum_{n=11}^{17} r_5 r_{n-4} 2^{-n} y_4 - \sum_{n=13}^{17} r_6 r_{n-4} 2^{-n} y_4 - \sum_{n=15}^{17} r_7 r_{n-4} 2^{-n} y_4 \quad (31)$$

Expressions (30) and (31) can be reduced to the following equations

$$\cos\theta = (1 - \sum_{i=M+1}^{N-1} \sum_{j=i+1(i+j)\le P}^{N-1} r_i r_j 2^{-(i+j)}) x_M - (\sum_{i=M+1}^{N-1} r_i 2^{-i}) y_M \quad (32)$$

$$\sin\theta = (1 - \sum_{i=M+1}^{N-1} \sum_{j=i+1(i+j)\le P}^{N-1} r_i r_j 2^{-(i+j)}) y_M + (\sum_{i=M+1}^{N-1} r_i 2^{-i}) x_M \quad (33)$$

We can further optimize the modified CORDIC algorithm by using reverse encoding and mapping the expressions in r_i into two binary constants, which will require four parallel multipliers and two adders to compute the desired results in a single cycle. The expressions (32) and (33) have two constants given below

$$const_1 = (\sum_{i=M+1}^{N-1} r_i 2^{-i}) \quad (34)$$

$$const_2 = (1 - \sum_{i=M+1}^{N-1} \sum_{j=i+1(i+j)\le P}^{N-1} r_i r_j 2^{-(i+j)}) \quad (35)$$

The following Equations (36) and (37) gives the desired results in a single cycle by using these constants.

$$\cos\theta = const_2 \times x_M - const_1 \times y_m \quad (36)$$

$$\sin\theta = const_1 \times x_M + const_2 \times y_m \quad (37)$$

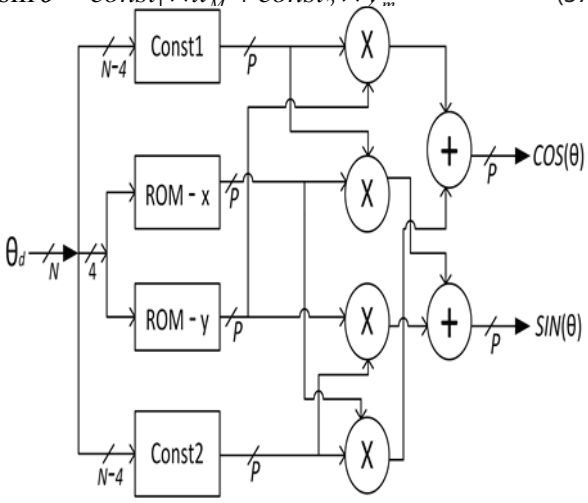

Figure 8: The optimal hardware design which computes sine and cosine in a single cycle

The single cycle modified CORDIC design [33, 34] is shown in Fig. 8. The constants in Equations (36) and (37) can be inverse coded using Equation (21). The const1 can be inverse coded as

$$\sum_{i=M+1}^{N-1} r_i 2^{-(i+1)} = \sum_{i=M+1}^{N-1} b_i 2^{-i} - 2^{-M} + 2^{-N} \quad (38)$$

The b_i s are used for computing the constant without modification. The 2^{-N} term is eliminated by appending 1 to b. MSB of b_N and the term - 2^{-M} is eliminated by flipping b_{M+1} bit and assigning negative weight to it. Thus const1 can be expressed as

$$const_1 = -b'_{M+1} 2^{-M} + \sum_{i=M+2}^{N} b_i 2^{-(i-1)} \quad (39)$$

The complement of the bit b_{M+1} results in the bit b'_{M+1}. We can implement Equation (39) in hardware by concatenating the bits b_i. The const2 can be implemented by computing $t_k s$ for $i + j = 2M + 1,\ldots,P$ as $t_k s = r_i r_j$, where $k = i+j$ and $k \leq P$. The Equation (38) can be used to inverse code the $t_k s$ and Equation (34) is used to compute its equivalent as const1. Let N=16 and P=16 holds. For this the values of $t_k s$ are computed for i=5,6,7. For i=5, t_k are inverse coded as constant value $t_k = r_5 r_j$, for j=6,7,...,11 where $k = 5+j$ and $k \leq P$.

$$\sum_{j=i+1}^{\leq P} t_5, j 2^{-(j+1)} = \sum_{k=2M+1}^{N-1} c_k 2^{-k} \quad (40)$$

Here, $c_k = b_5 \sim \wedge b_j$ and $k = 5+j$ hold. Then values of t_6, j, t_7, j are computed for every index i. The $t_k s$ are inverse coded using $b_k s$ as

$$beta_0 = \sum_{k=2M+1}^{N-1} t_k 2^{-(k+1)} = \sum_{k=2M+1}^{N-1} b_k 2^{-k} - 2^{-2M} + 2^{-N} \quad (41)$$

Equation (41) after some manipulations can be written as

$$beta_0 = -b'_{2M+1} 2^{-2M} + \sum_{k=2M+2}^{N} b_i 2^{-(i-1)} \quad (42)$$

Similarly, the values of *beta1* and *beta2* can also be computed following the same steps for i=6 and i=7, respectively. The const2 can be computed as

$$const_2 = 1 - beta_0 - beta_1 - beta_2 \quad (43)$$

The inverse coded constants const1 and const2 can be implemented in Verilog HDL for the desired angle θ_d, by the simple concatenation of bits as given below

$$Constant1 : const_1 = \{6\{\sim b[5]\}, b[10:0], 1'b1\}$$

$$Constant2 : c_0 = \{6\{b[5]\}\} \sim \wedge b[11:6]$$

$$c_1 = \{4\{b[6]\}\} \sim \wedge b[10:7]$$

$$c_2 = \{2\{b[7]\}\} \sim \wedge b[9:8]$$

$$beta_0 = \{12\{\sim c_0[5]\}, c_0[4:0], 1'b1\}$$

$$beta_1 = \{14\{\sim c_1[3]\}, c_1[2:0], 1'b1\}$$

$$beta_2 = \{16\{\sim c_2[1]\}, c_2[0], 1'b1\}$$

$$const_2 = 16'h4000 - (beta_0 + beta_1 + beta_2)$$

3.1 Modified CORDIC Hardware Accelerator

We have developed a second novel mixed hardware/software CORDIC accelerator unit using the Modified CORDIC algorithm. In Modified CORDIC algorithm based accelerator we implemented the two constants const1 and const2 in hardware using the Verilog HDL. The four multiplications and two additions in Equations (36) and (37) are implemented in software. Fig. 9 shows the block diagram of Modified CORDIC hardware accelerator unit. We used Xilinx Spartan-6 FPGA SP605 Evaluation Kit [41] and Xilinx Embedded Development Kit (EDK) [39] for the implementation. Xilinx Microblaze soft core processor system [40] is used to execute the software part of Modified CORDIC accelerator. The hardware part of Modified CORDIC accelerator is attached to the MicroBlaze processor system using Fast Simplex Link (FSL) bus system [42].

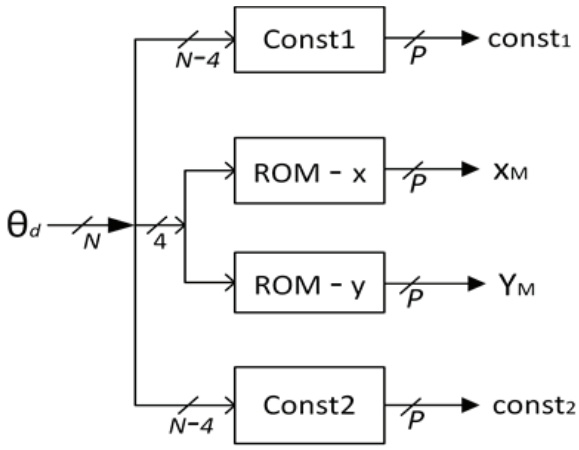

Figure 9: Modified CORDIC HW Accelerator

First, the Modified CORDIC algorithm is implemented in C programming language and it is executed on MicroBlaze processor using Xilinx Software Development Kit (SDK) [39]. The cycle count for the software implementation of Modified CORDIC algorithm was measured

using the XPS hardware timer block. The MicroBlaze processor takes 2971 cycles to compute the values of sine and cosine while executing the complete software implementation of Modified CORDIC algorithm. Most of the processor time is spent while computing the two constants of Modified CORDIC algorithm. The two constants can be implemented more efficiently in hardware using Verilog HDL by simple concatenation of bits, compared to software implementation. This is the reason for implementing the two constants in hardware and doing the four multiplications and two additions in software. In the next step, Verilog HDL code of hardware part of Modified CORDIC accelerator was implemented and verified using Xilinx ISE design suit [39]. The 16 precomputed values of x_M and y_M each for $M = 4$ in Q2.16 format was generated using MATLAB. These values are stored in a lookup table (LUT) in hardware using Verilog HDL for implementing the Equations (36) and (37) of Modified CORDIC algorithm. The last four bits of theta desired θ_d [15 : 12] form the address of this LUT. The Modified CORDIC hardware accelerator was attached with the Microblaze processor system via FSL bus, using Xilinx Platform Studio (XPS) [39].

Table 2: Synthesis results of Modified CORDIC Algorithm and Modified CORDIC Hardware Accelerator

	Modified CORDIC Algorithm	Modified CORDIC Accelerator
Max Freq	212 MHz	954 MHz
Latency	4.704ns	1.048ns
RAMs	16x36-bit RAM	16x36-bit RAM
Adders	5 x 18-bit	3 x 18-bit
Multipliers	2x(13x18-bit), 2x(18x18-bit)	0
Xors	3	3
Slice Registers	52	64
Slice LUTs	129	82
Slices	57	35

Later, the software part of the Modified CORDIC accelerator was implemented in C programming using Xilinx SDK. The predefined C functions of SDK are used to communicate with the hardware part of Modified CORDIC accelerator via FSL bus. First, the Microblaze sends theta desired θ_d through FSL bus to the hardware part of Modified CORDIC accelerator. This computes the two constants const1 and const2 in hardware and sends it along with x_M and y_M values, obtained from the LUT to the MicroBlaze processor via FSL bus. Later the four multiplications and two additions are performed in software on the MicroBlaze processor, using the Equations (36) and (37) of Modified CORDIC algorithm. Fig. 10 shows the steps involved in the computation of sine and cosine using the mixed hardware/software Modified CORDIC accelerator.

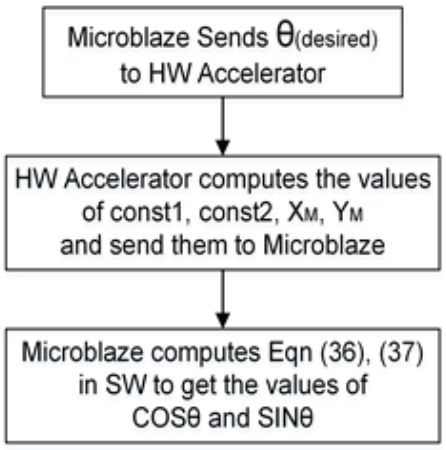

Figure 10: Flow chart for Modified CORDIC Accelerator Implementation

Table 3: Cycle count and Energy dissipation at clock period 20ns

Architecture	#Cycles	Power (mW)	Energy* (µJ)
SCORDIC SW	933	178	3.3214
SCORDIC Mixed	601	181	2.1756
MCORDIC Mixed	64	183	0.2304

*: Energy dissipation = #cycles × clock period× power.

The cycle count for mixed hardware/software implementation of Modified CORDIC algorithm was measured using the XPS hardware timer block. The mixed hardware/software implementation of Modified CORDIC algorithm takes 64 cycles to compute the values of sine and cosine. The energy dissipation was calculated for the Modified CORDIC mixed hardware/software implementation as shown in Table 3. Our evaluation shows that a Modified CORDIC accelerated embedded processor datapath is 14.5 times more cycle efficient than a datapath lacking a Modified CORDIC accelerator. This design leads to 14 times energy reduc-

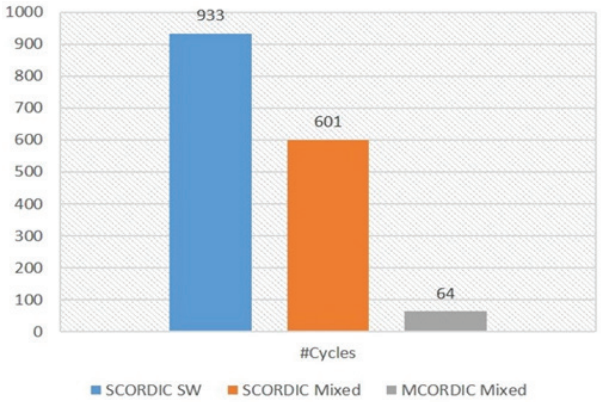

Figure 11: Cycle count of different architectures

tion with a very small area overhead. Fig.11 and 12 shows the cycle count and energy dissipation of different architectures, respectively.

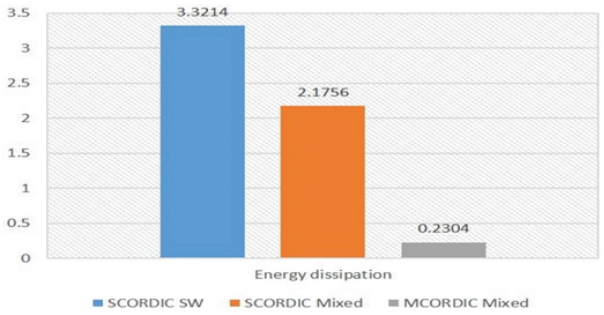

Figure 12: Energy dissipation of different architectures

The Modified CORDIC mixed hardware/software implementation is also area efficient as we performed the four multiplications and two additions of Modified CORDIC algorithm in software. This code is executed on the MicroBlaze processor system which results in saving 2x(13x18-bit), 2x(18x18-bit) Multipliers and 2x(18-bit) Adders as shown in Table 2. The Modified CORDIC algorithm and Modified CORDIC hardware accelerator block were synthesized on 7vx485tffg1157-3 Virtex-7 FPGA device. This FPGA device uses a 28nm technology and gives a critical path delay of 4.704ns and 1.048ns respectively as shown in Table 2. The Modified CORDIC hardware accelerator block has 4.5 times reduced latency compared to Modified CORDIC algorithm. Because the four multiplication and two addition operations in the critical path delay have been removed and these operations are performed in software. The Modified CORDIC hardware accelerator block has 4.3 times less latency and takes 4 times less area compared to Standard CORDIC Time Shared implementation. The novelty of the design in the use of Modified CORDIC accelerator is that it takes a single iteration to compute the values of sine and cosine as compared to the Standard CORDIC algorithm, which requires N iterations.

Table 4: Delay and Area comparison for FPGA implementations

Reference	Slices	Clock(MHz)	Latency(ns)
Volder [1]	1111	21.43	46.66
Xilinx [43]	1057	37.70	26.52
Perwaiz [47]	978	139.87	7.15
Zaidi [45]	769	151.73	6.59
Ramesh [48]	373	198.27	5.04
Aguirre [49]	276	83.99	11.90
SCORDIC TS	148	19.3	51.52
MCORDIC Proposed	35	954	1.048 + SW_{time}=11.96

Table 4 compares the area and latency of proposed Modified CORDIC mixed hardware/software implementation with other referenced CORDIC FPGA implementation designs. Our proposed technique has reduced area and latency requirements. The latency of proposed Modified CORDIC mixed hardware/software implementation is 1.048ns in addition to the time required to perform the four multiplications and two additions in software. This software code is executed on an embedded processor system using the Equations (36) and (37) of Modified CORDIC algorithm.

4 Conclusion

We have presented two novel CORDIC accelerator units using a mixed hardware/software approach. These CORDIC accelerators were integrated with an embedded processor datapath to enhance the processor performance in terms of execution time and energy efficiency. We used Xilinx Spartan-6 FPGA Evaluation Kit and Xilinx Embedded Development Kit (EDK) for the implementation. Xilinx Microblaze soft core processor system was used to execute the software part of CORDIC accelerators. These CORDIC hardware accelerators were attached with the MicroBlaze processor using FSL bus system. The first accelerator was implemented using the Standard CORDIC algorithm. Our evaluation shows that the Standard CORDIC accelerated MicroBlaze processor datapath is 35% more cycle efficient than a datapath lacking Standard CORDIC accelerator. This design also leads to 34% energy reduction. The mixed hardware/software implementation of Standard CORDIC algorithm is area efficient as it saved two 16-bit adders. The second accelerator is implemented using a Modified CORDIC algorithm. Our evaluation shows that a Modified CORDIC accelerated MicroBlaze processor datapath is 14.5 times more cycle efficient than a datapath lacking Modified CORDIC accelerator. This design leads to 14 times energy reduction with a very small area overhead. The mixed hardware/software Modified CORDIC accelerator is area efficient as it saved four multipliers and two adders.

5 Acknowledgments

This work is partially supported by the Chinese Academy of Sciences and The World Academy of Sciences CAS-TWAS President's Fellowship 2013-2017.

6 References

1. J. E. Volder, "The CORDIC trigonometric computing technique," IRE Trans. Electron. Computers, vol. EC-8, pp. 330-334, Sept. 1959.
2. J. E. Volder, "The birth of CORDIC," J. VLSI Signal Process., vol. 25, pp. 101-105, 2000.
3. J. S. Walther, "A unified algorithm for elementary functions," in Proc. 38th Spring Joint Computer Conf., Atlantic City, NJ, 1971, pp. 379-385.
4. J. S.Walther, "The story of unified CORDIC," J. VLSI Signal Process., vol. 25, no. 2, pp. 107-112, June 2000.
5. D. S. Cochran, "Algorithms and accuracy in the HP-35," Hewlett- Packard J., pp. 1-11, Jun. 1972.
6. P. K. Meher, J. Valls, T-B Juang, K. Sridharan, and K. Maharatna, "50 Years of CORDIC: Algorithms, Architectures, and Applications", IEEE Transactions on Circuits Systems-I: Regular Papers, vol.56, no. 9, pp. 1893-1907, September 2009.
7. P. K. Meher and S. Y. Park, "CORDIC Designs for Fixed Angle of Rotation," IEEE Transactions on VLSI Systems, vol.21, no.2, pp. 217–228, February 2013.
8. Y. H. Hu, "CORDIC-based VLSI architectures for digital signal processing", IEEE Signal Processing Mag., pp.16-35, 1992
9. Y.H. Hu, S. Naganathan, An angle recoding method for CORDIC algorithm implementation. IEEE Trans. Comput. 42(1), 99-102, (1993)
10. T. Srikanthan, B. Gisuthan, Optimizing scaling factor computations in flat CORDIC. J. Circuits Syst. Comput. 11(1), 17–33 (2002)
11. C.-S. Wu , A.-Y. Wu and C.-H. Lin "A high-performance/low-latency vector rotational CORDIC architecture based on extended elementary angle set and trellis-based searching schemes", IEEE Trans. Circuits Syst.II, Analog Digit. Signal Process., vol. 50, no. 9, pp.589 -601, 2003
12. C. S. Wu and A.-Y. Wu, "A new trellis-based searching scheme for EEAS-based CORDIC algorithm", Proc. IEEE Int. Conf. Acoust. Speech, Signal Processing, vol. 2, pp.1229 -1232, 2001
13. E. Deprettere, P. Dewilde, R. Udo, Pipelined CORDIC architectures for fast VLSI filtering and array processing, in IEEE International Conference on Acoustics, Speech, and Signal Processing, vol. 9 (1984), pp. 250–253
14. E. Antelo, J. Villalba and E. Zapata "A low-latency pipelined 2D and 3D CORDIC processors", IEEE Trans. Comput., vol. 57, no. 3, pp.404 -417 2008
15. M. Jun and K. K. Parhi, "Pipelined CORDIC-based state-space orthogonal recursive digital filters using matrix look-ahead," IEEE Trans. Signal Process., vol. 52, pp. 2102-2119, July 2004.
16. E. Antelo, J. Villalba, J.D. Bruguera and E.L. Zapata, "High Performance Rotation Architectures Based on the Radix-4 CORDIC Algorithm," IEEE Trans. Computers, vol. 46, no. 8, pp. 855-870, Aug. 1997.
17. E. Antelo, J. D. Bruguera and E. L. Zapata "Unified mixed radix 2-4 redundant CORDIC processor", IEEE Trans. Comput., vol. 45, no. 9, pp.1068 -1073 1996
18 J.-L. Sanchez, H. Mora, J. Mora, A. Jimeno, Architecture implementation of an improved decimal CORDIC method, in IEEE International Conference on Computer Design (ICCD), Oct. 2008, pp. 95-100
19. A.J. Morenilla, H.M. Mora, J.-L.S. Romero, F.P. Lopez, A Fast Architecture for Radix 10 Coordinates Rotation, in IEEE Southern Conference on Programmable Logic (SPL), Feb. 2007, pp. 39-44
20. L. Vachhani, K. Sridharan, P.K. Meher, Efficient CORDIC algorithms and architectures for low area and high throughput implementation. IEEE Trans. Circuits Syst. II, Express Briefs 56(1), 61-65 (2009)
21. D. De Caro, N. Petra, A.G.M. Strollo, Digital synthesizer/mixer with hybrid CORDIC-multiplier architecture: error analysis and optimization. IEEE Trans. Circuits Syst. I, Regular Papers. 56(2), 364-373 (2009)
22. S. Aggarwal, P.K. Meher, K. Khare, Scale-free hyperbolic CORDIC processor and its application to waveform generation. IEEE Trans. Circuits Syst. I, Regular Pap. 60(2), 314-326 (2012)
23. D. De Caro, N. Petra, A.G.M. Strollo, Digital synthesizer/mixer with hybrid CORDIC-multiplier architecture: error analysis and optimization. IEEE Trans. Circuits Systems I, Regular Papers. 56(2), 364-373 (2009)
24. T. K. Rodrigues, Swartzlander, E. E. (2010). Adaptive CORDIC: Using parallel angle recoding to accelerate rotations. IEEE Transactions on Computers, 59(4), 522-531.
25. Aytore, E., Alkar, A. Z. (2010). Highly accurate reduced iteration CORDIC processor algorithm. International Journal of Electronics, 97(2), 163-176.
26. Y.H. Hu and Naganathan, S. (1993). An angle recoding method for CORDIC algorithm implementation. IEEE Transactions on Computers, 42(1), 99-102.
27. Antelo, E., Villalba, J., Bruguera, J. D., Zapata, E. L. (1997). High performance rotation architectures based on the radix-4 CORDIC algorithm. IEEE Transactions on Computers, 46(8), 855-870.
28. Phatak, D. S. (1998). Double step branching CORDIC: A new algorithm for fast sine and cosine generation. IEEE Transactions on Computers, 47(5), 587-602.
29. Juang, T. B., Hsiao, S. F., Tsai, M. Y. (2004). Para-CORDIC: Parallel CORDIC rotation algorithm. IEEE Transactions on Circuits and Systems I, 51(8), 1515-1524.

30. Juang, T. B. (2006). Area/delay efficient recoding methods for parallel CORDIC rotations. Proceedings of Asia Pacific Conference on Circuits and Systems, pp. 1541-1544.
31. Juang, T. B. (2008). Low latency angle recoding methods for the higher bit-width parallel CORDIC rotator implementations. IEEE Transactions on Circuits and Systems II, 55(11), 1139-1143.
32. Kao, C. C. (2011). High-performance CORDIC rotation algorithm based on look-ahead techniques. International Journal of Electronics 98(8), 1075-1089.
33. Hamid Mehmood Allah Ditta Kamboh and Shoab Ahmed Khan (2014) IS-CORDIC: a fixed-point inverse recoded single iteration CORDIC architecture, International Journal of Electronics.
34. Shoab, A. K. (2011). "Digital design of signal processing systems: A practical approach" (1st ed). New York, NY: John Wiley.
35. V. Sklyarov, I. Skliarova, A. Rjabov, A. Sudnitson, "Zynq-based System for Extracting Sorted Subsets from Large Data Sets", Informacije MIDEM-Journal of Microelectronics, Electronic Components and Materials Vol. 45, No. 2 (2015), 142-152.
36. Abdul Rehman Buzdar, Liguo Sun, Azhar Latif and Abdullah Buzdar, "Distance and Speed Measurements using FPGA and ASIC on a high data rate system" International Journal of Advanced Computer Science and Applications(IJACSA), 6(10), 2015, 273-282.
37. Abdul Rehman Buzdar, Liguo Sun, Azhar Latif and Abdullah Buzdar, "Instruction Decompressor Design for a VLIW Processor", Informacije MIDEM-Journal of Microelectronics, Electronic Components and Materials Vol. 45, No. 4 (2015), 225-236
38. Abdul Rehman Buzdar, Azhar Latif, Liguo Sun and Abdullah Buzdar, "FPGA Prototype Implementation of Digital Hearing Aid from Software to Complete Hardware Design" International Journal of Advanced Computer Science and Applications(IJACSA), 7(1), 2016, 649-658.
39. Xilinx Inc. FPGA Design Tools. Silicon Devices. [Online]. Available: http://www.xilinx.com
40. Xilinx. MicroBlaze. [Online]. Available: http://www.xilinx.com/tools/microblaze.htm
41. Xilinx Spartan-6 FPGA SP605 Evaluation Kit. [Online]. Available: http://www.xilinx.com/products/boards-and-kits/ek-s6-sp605-g.html
42. Xilinx Fast Simplex Link (FSL). [Online]. Available: http://www.xilinx.com/products/intellectual-property/fsl.html
43. Xilinx. CORDIC v4.0. Xilinx LogiCore DS429, pp. 1–29.
44. Madisctti, A., Kwentus, A. Y., Wilson, A. N. J. (1999). A 100 MHz, 16 bits, direct digital frequency synthesizer with a 100-dBc spurious-free dynamic range. IEEE Journal of Solid-state Circuits, 34(8), 1034-1043.
45. Zaidi, T., Chaudry, Q., Khan, S. A. (2004). An area and time efficient collapsed modified CORDIC DDFS architecture for high rate digital receivers. Proceedings of INMIC 2004, pp. 677-681.
46. Rehan Hameed, Wajahat Qadeer, Megan Wachs, Omid Azizi, Alex Solomatnikov, Benjamin C. Lee, Stephen Richardson, Christos Kozyrakis, and Mark Horowitz. Understanding sources of inefficiency in general-purpose chips. SIGARCH Comput. Archit. News, 38:37-47, June 2010.
47. Perwaiz, A., Kamboh, H. M., Khan, S. A. (2010). FPGA fabric specific optimization for RTL design. Pakistan Journal of Engineering and Applied Sciences, 6, 52-57.
48. Ramesh Bhakthavatchalu, Parvathi Nair, Jismi K., and Sinith M.S., "A Comparison of Pipelined Parallel and Iterative CORDIC Design on FPGA", IEEE 5th International Conference on Industrial and Information Systems, ICIIS, Jul 29-Aug 01, 2010.
49. F. Aguirre-ramos, A. Morales-reyes, R. Cumplido, C. Feregrino-uribe, "An Area Efficient Composed CORDIC Architecture," Advances in Electrical and Computer Engineering, vol.14, no.2, pp.113-116, 2014.

Arrived: 11. 04. 2016
Accepted: 27. 10. 2016