A Fast FPGA based Architecture for Determining the Sine and Cosine Value

Conference Paper · February 2014

DOI: 10.13140/2.1.5158.5922

4 authors:

Atanu Dey
Indian Institute of Technology Kharagpur

Tanima Bhattacharyya

Abul Hasnat
Government College of Engineering and Textile Technology, Berhampore, West Bengal

Santanu Halder
Kalyani Government Engineering College

Proc. of Int. Conf. on Advances in Communication, Network, and Computing, CNC

ELSEVIER

A Fast FPGA based Architecture for Determining the Sine and Cosine Value

[1]Atanu Dey, [1]Tanima Bhattacharyya, [1]Abul Hasnat and Santanu Halder [2]
[1] Department of Computer Science & Engineering
Government College of Engineering & Textile Technology, Berhampore, West Bengal, India
Email: atanud0@gmail.com, tanima.bhattacharyya0301@gmail.com, abulhasnat.007@gmail.com
[2] Department of Computer Science & Engineering
Government College of Engineering & Leather Technology, Kolkata, India
Email: sant.halder@gmail.com

***Abstract*— This paper aims to design a fast FPGA based architecture for determining the sine and cosine values using Taylor's Series. In this work, a pipelined architecture has been proposed for designing a flexible and scalable digital sine and cosine wave generator. An FPGA–Based architecture is implemented in VHDL, synthesized on Xilinx Spartan 3 xc3s200-5ft256 FPGA kit simulated on ModelSim 6.2c. The proposed architecture gives accuracy of the computed sine and cosine values up to 21st bits in all cases and up to 22 or 23 bits for some cases in six clock cycles only whereas CORDIC architecture needs 21 clocks for 21bit accuracy. Trigonometric waves is used in countless applications i.e. Software Defined Radio (SDR). The proposed system is able to operate at the frequency of 93.119 MHz.**

***Index Terms*— FPGA, SDR, CORDIC, Sine, Cosine.**

I. INTRODUCTION

There are plenty of applications which require digital wave generators. Wireless and mobile systems are among the fastest growing application areas; in particular, Software Defined Radio (SDR) is currently a focus of research and development. An SDR allows performing many functions based on a single hardware platform, thus highly reconfigurable resources for signal processing are needed, mainly for modulation and demodulation of digital signals. Fields of development are increasing every day with applications such as cell phones or military communication.

There is a widely used method, CORDIC [1-4] which is implemented in most of the application to generate digital sine and cosine waves. Although CORDIC algorithm computes trigonometric angle values in a simple and faster way but it needs n number of clock cycles to get accuracy up to nth bit [1-4]. So there is a need to develop a system which would give the better accuracy in less number of clock cycles. The proposed architecture offers an alternative, which takes 23 bits input (for single precision format) and generates Sine and Cosine values in six clock cycles only and gives accuracy up to 21 bits in all the cases (and for some cases 22 bits or 23 bits accuracy achieved) where as CORDIC needs 21 clock cycle to compute the same accuracy. In the present work, The Taylor series [5-7] is implemented in VHDL based pipelined architecture is proposed which is synthesized in Xilinx Spartan 3 XC3S50-5PQ208 FPGA, simulated on the Modelsim 6.2c from Mentor Graphics Corporation.

This paper is organized as follows: Section II describes the existing CORDIC algorithm briefly. Section III briefly explains the Taylor series [5-7]. Section IV presents the top level design of proposed hardware. Section V depicts the proposed system architecture. Section VI shows the experimental results and finally section VII concludes and remarks about some of the aspects analyzed in this paper.

II. EXISTING METHODOLOGY

Coordinate Rotation Digital Computer (CORDIC) is a well known algorithm used to approximate iteratively some transcendental function [1-4].

Let the starting vector is defined as $v_0 = (x_0, y_0)$ and after n iterations, v_0 moves to v_n in anticlockwise direction by θ^0 as shown in figure 1.

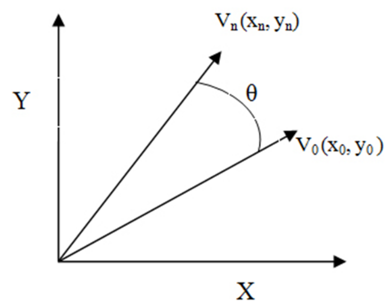


Figure 1. Anticlockwise rotation of a vector v_0 to v_n by θ^0

So, the vector $v_n = (x_n, y_n)$ can be represented by Eq. 1and Eq. 2.

$$x_n = x_0 \cos\theta - y_0 \sin\theta \qquad (1)$$

$$y_n = y_0 \cos\theta + x_0 \sin\theta \qquad (2)$$

In each iteration *i*, the vector perform a micro-rotation by θ_i, so the new vector is calculated with a similar function

$$x_{i+1} = x_i \cos\theta_{i+1} - y_i \sin\theta_{i+1} \qquad (3)$$

$$y_{i+1} = y_i \cos\theta_{i+1} + x_i \sin\theta_{i+1} \qquad (4)$$

When the term $\cos\theta_{i+1}$ is factorized, components in the vector are described by

$$x_{i+1} = \cos\theta_{i+1}(x_i - y_i \tan\theta_{i+1}) \qquad (5)$$

$$y_{i+1} = \sin\theta_{i+1}(y_i + x_i \tan\theta_{i+1}) \qquad (6)$$

$\tan\theta_{i+1}$ is restricted to $\pm 2^{-i}$, so multiplication is converted in an arithmetic right shift. It is also useful to use the identity $\cos\theta_{i+1} = \cos(\arctan 2^{-i})$ to define the next variables.

$$K_i = \cos(\arctan 2^{-i}) = 1/\sqrt{(1+2^{-2i})} \qquad (7)$$

$$d_i = \pm 1 \qquad (8)$$

Cosine is an even function, therefore $\cos(\alpha) = \cos(-\alpha)$. So (5) and (6) can be transformed into

$$x_{i+1} = K_i(x_i - y_i d_i 2^{-i}) \qquad (9)$$

$$y_{i+1} = K_i(y_i + x_i d_i 2^{-i}) \qquad (10)$$

Multiplication by K_i is avoided by considering it as a gain factor for all iterations. If n number of iterations are performed, then K is defined as the multiplication of every K_i.

$$K = \prod K_i = \prod I / \sqrt{(1+2^{-2i})} \tag{11}$$

As the vector is initialized with constant K, the vector components of each the iteration are simplified to

$$x_{i+1} = (x_i - y_i d_i 2^{-i}) \tag{12}$$

$$y_{i+1} = (y_i + x_i d_i 2^{-i}) \tag{13}$$

On each iteration it is necessary to decide whether $d_i = 1$ or $d_i = -1$. In order to make that decision, the difference between the desired angle and the current angle is used. So a new variable know as accumulator is defined as:

$$z_{i+1} = z_i - d_i \arctan 2^{-i} \tag{14}$$

The value of z_0 is the angle for which sine and cosine are to be calculated. To know whether d_i should be positive or negative, the following rule is used:

$$\begin{array}{ll} d_i = -1 & \sin z_i < 0 \\ d_i = +1 & \sin z_i \geq 0 \end{array} \tag{15}$$

The well known CORDIC by Volder [1], is a classical technique to compute both sine and cosine using such decomposition into micro-rotations. These rotations are chosen in such a way that the value can be computed using one adder or one subtractor and two shifters for each clock cycle.

In vectoring mode, coordinates (x_0, y_0) are rotated until y_0 converges to zero. In rotation mode, initial vector (x_0, y_0) starts aligned with the x-axis and is rotated by an angle of θ_i every cycle, so after n cycles, θ_i is obtained angle. Figure 2 shows the CORDIC pipeline architecture given by Esbetan O. Garcia et al[2].

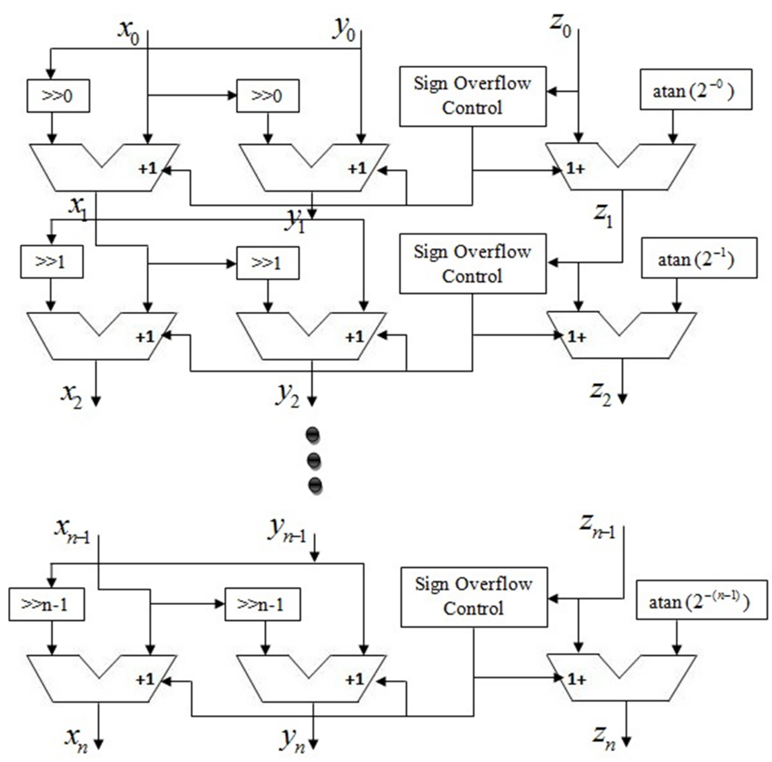


Figure 2. Pipelined CORDIC Architecture

Like CORDIC, proposed implementation does not need to have Cosine values stored in memory, instead it approximates the results, on each step of the iterative process. The proposed architecture may be extended for a specific resolution without a considerable increase in the number of components used.

III. Proposed Methodology

Taylor series [5-7] is a representation of a function as an infinite sum of terms that are calculated from the values of the function's derivatives at a single point. Sine and Cosine value is calculated using Eq. 16 and Eq. 17 respectively.

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \frac{x^{12}}{12!} + .. \qquad (16)$$

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!} + \frac{x^{13}}{13!} - .. \qquad (17)$$

Where $-\infty < x < \infty$

As the present work takes into consideration tenth and eleventh power of x for cosine and sine values respectively, the Taylor series reduces to Eq. 18 and Eq. 19.

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} \qquad (18)$$

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!} \qquad (19)$$

As $1/(2!)$, $1/(4!)$, $1/(6!)$, $1/(8!)$ and $1/(10!)$ are constant so these values are stored in a look up table. The next section describes VHDL implementation of Eq. 18 and Eq.19.

IV. Top Level Design

The top level design of proposed architecture is shown in Figure 3. The proposed architecture takes one 23-bit value as input. Then the system calculates the Sine and Cosine value using the given input. Finally, the system generates two 23-bit values Sine and Cosine respectively. Fig. 3 shows the top level design of the proposed architecture.

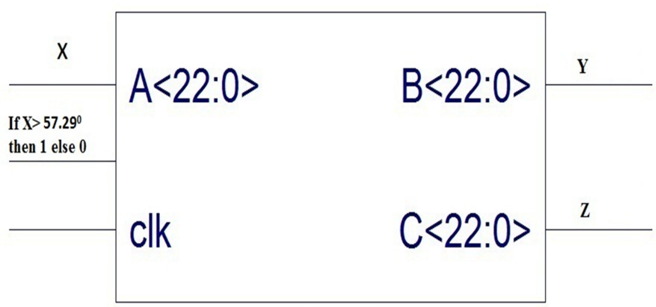


Figure 3. The top level design of Proposed architecture

In the present work, if the input angle is less than 57.29 degree results to less than 1 in radian. Thus for any angle θ which is less than 57.29 degree is directly converted into radian value and its 23 bit binary representation is used as input and as output Y, Z is considered as $\sin(x)$, $\cos(x)$ values respectively. But if the input angle x is greater than 57.29 degree, then simply $(90 - x)$ is used as input angle and its 23 bit binary representation of the angle in radian is used as input to the system. Second input is set to one to indicate to alternate the output i.e. Y, Z is considered as $\cos(x)$, $\sin(x)$ respectively because $\sin(90 - x) = \cos(x)$ or vice versa. This is done simply to avoid normalization of input and output values further i.e. if the input angle is less than 57.29 degree then its corresponding radian value lies in the range $0 \le x < 1$. Assuming decimal point on left side, corresponding binary value is given as input and output is also taken as the given binary string assuming decimal point lies in the left side. Thus no normalization is needed.

V. SYSTEM ARCHITECTURE

The proposed system architecture is shown in Figure 4 which has a pipelined architecture, consisting two adder, three subtractor and five multiplier for sine and cosine angle calculation to get 21-bit accuracy all the times and 22 to 23 accuracy in some cases.
The various blocks of this architecture are described next. All the inputs in each and every block are taken in 23 bits and also all blocks forwards output in 23 bits.

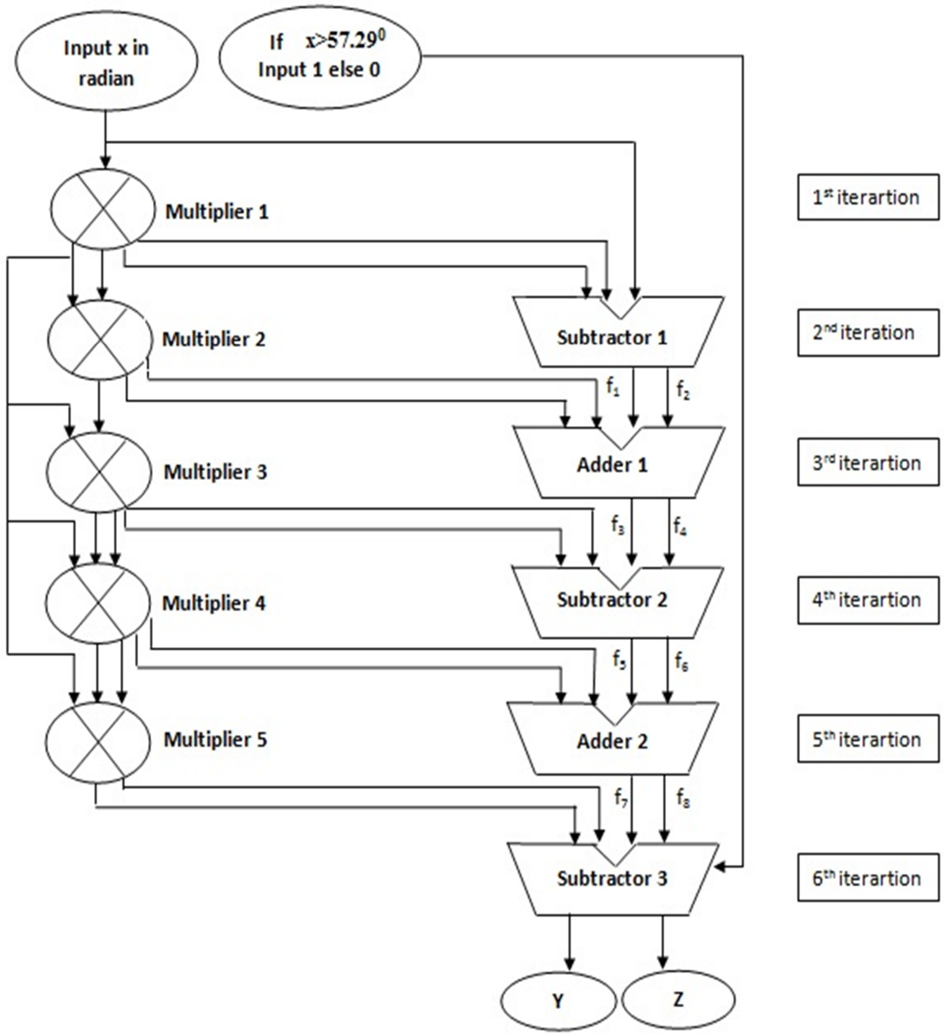


Figure 4. Proposed system architecture

Multiplier 1:

Multiplier1 unit takes angle as input (in radian) and calculates x^2, x^3, $\frac{x^2}{2!}$ and $\frac{x^3}{3!}$. Multiplier1 forwards x^2 value as input to all other multipliers. It also forwards x^3 value as input to Multiplier 2 and also forwards $\frac{x^2}{2!}$ and $\frac{x^3}{3!}$ as inputs to Subtractor 1.

Multiplier 2:

Multiplier 2 takes x^2 and x^3 as inputs and calculates $x^4, x^5, \frac{x^4}{4!}$ and $\frac{x^5}{5!}$. Multiplier2 forwards x^4 and x^5

values input to Multiplier 3 and also forwards $\frac{x^4}{4!}$ and $\frac{x^5}{5!}$ as inputs to Adder 1.

Multiplier 3:

Multiplier 3 takes x^4 and x^5 as inputs and calculates $x^6, x^7, \frac{x^6}{6!}$ and $\frac{x^7}{7!}$. Multiplier3 forwards x^6 and x^7 values input to Multiplier 4 and also forwards $\frac{x^6}{6!}$ and $\frac{x^7}{7!}$ as inputs to Subtractor 2.

Multiplier 4:

Multiplier 4 takes x^6 and x^7 as inputs and calculates $x^8, x^9, \frac{x^8}{8!}$ and $\frac{x^9}{9!}$. Multiplier4 forwards x^8 and x^9 values input to Multiplier 5 and also forwards $\frac{x^8}{8!}$ and $\frac{x^9}{9!}$ as inputs to Adder 2.

Multiplier 5:

Multiplier 5 takes x^8 and x^9 as inputs and calculates $x^{10}, x^{11}, \frac{x^{10}}{10!}$ and $\frac{x^{11}}{11!}$. Multiplier 5 forwards x^{10} and x^{11} values input to Multiplier 5 and also forwards $\frac{x^{10}}{10!}$ and $\frac{x^{11}}{11!}$ as inputs to Subtractor 3.

Subtractor 1:

It takes angle as direct input (in radian) and also two more inputs $\frac{x^2}{2!}$ and $\frac{x^3}{2!}$ and calculates f_1 and f_2 as follows

$$f_1 = x - \frac{x^3}{3!} \tag{18}$$

$$f_2 = 1 - \frac{x^2}{2!} \tag{19}$$

And forwards the outputs f_1 and f_2 as input to Adder 1.

Adder 1:

It takes two inputs f_1, f_2 and calculates f_3 and f_3 as follows

$$f_3 = f_1 + \frac{x^5}{5!} \tag{20}$$

$$f_4 = f_2 + \frac{x^4}{4!} \tag{21}$$

And forwards the outputs f_3 and f_4 as input to Subtractor 2.

Subtractor 2:

It take two inputs f_3 and f_4 and calculates f_5 and f_6 as follows

$$f_5 = f_3 - \frac{x^7}{7!} \tag{22}$$

$$f_6 = f_4 - \frac{x^6}{6!} \tag{23}$$

And forwards the outputs f_5 and f_6 as input to Adder 2.

Adder 2:

It takes two inputs f_5, f_6 and calculates f_7 and f_8 as follows

$$f_7 = f_5 + \frac{x^9}{9!} \tag{24}$$

$$f_8 = f_6 + \frac{x^8}{8!} \quad (25)$$

And forwards the outputs f_7 and f_8 as input to Subtractor 3.

Subtractor 3:

It take two inputs f_7 and f_8 and calculates f_9 and f_{10} as follows

$$f_9 = f_7 - \frac{x^{11}}{11!} \quad (26)$$

$$f_{10} = f_8 - \frac{x^{10}}{10!} \quad (27)$$

And finally the output f_9 is the calculated value of $\sin(x)$ and f_{10} as calculated value of $\cos(x)$ if input angle in degree is less than 57.29 else f_9 is the calculated value of $\cos(x)$ and f_{10} as calculated value of $\sin(x)$.

The system architecture shown in Fig. 4 is a pipeline architecture which computes sine and cosine values in only 6 clock cycle resulting faster computation.

VI. Experimental Results

The system for Sine Cosine value calculation is implemented using VHDL, synthesized for a Xilinx Spartan 3 xc3s200-5ft256 with simulation on the Modelsim 6.2c from Mentor Graphics Corporation. Experiment has been conducted on different value in the range $[0 \ , \frac{\Pi}{2}]$. The proposed architecture is capable to operate at the clock frequency of 93.119 MHz and takes only 6 clock cycles to get the result in 23 bits. The device utilization summary for the proposed architecture for sine and cosine value computation is given in Table I. For comparison study of the proposed architecture with the standard CORDIC architecture, the device utilization summary of CORDIC architecture simulated on a Xilinx Spartan 3 xc3s200-5ft256 using Xilinx ISE 7.1 is also given in Table I.

TABLE I. Comparison Device Utilization Summary Of The Proposed Architecture And CORDIC

	Proposed Method		CORDIC	
Parameter	Used	%	Used	%
Number of Slices	946	49	1075	55
Number of Flip Flops	1061	27	570	14
Number of 4 input LUTs	1411	36	1737	44
Number of bonded IOBs	69	39	43	24
Number of GCLKs	1	12	1	12
Maximum Frequency	93.119 MHz		124.67MHz	

Device utilization summary is given in Table I, it shows that the proposed architecture takes 129 (1075-946) number of Slices, 326 (1737-1411) number of 4 input LUTs less compared to that of CORDIC architecture. But the proposed architecture computes sine and cosine values up to 21st bit accuracy in only 6 clock cycle whereas CORDIC architecture will require 21 clock cycles to compute the same [1-3]. The FPGA based proposed system operates as fast as 93.119 MHz.

The proposed architecture has been tested using Modelsim 6.2c simulator and computation accuracy achieved up to 21th bits or more. Table II shows the computed sine and cosine values using the proposed architecture which clearly shows the accuracy up to 21th bits in only 6 clock cycles whereas to get accuracy up to 21th bits CORDIC architecture needs 21 clock cycles. Table II shows the computed sine and cosine values calculated using proposed method. The Figure 5(a) and 5(b) shows 23 bit, 22 bit and 21 bit accuracies given by the proposed system for sine and cosine values. For calculation of sine(x), the proposed system produces 23 bit, 22 bit and 21 bit accuracy for 53.33%, 81.66% and 100% cases respectively and for cosine(x), it produces 23 bit, 22 bit and 21 bit accuracy for 53.33%, 80%, and 100% cases respectively.

Figure 6 shows the snapshot of sample output which is simulated on ModelSim SE 6.2c where input angle (in radian) and output values are shown in red and yellow marked rectangles respectively.

TABLE II. EXPERIMENTAL RESULTS FOR BOTH SINE AND COSINE VALUES

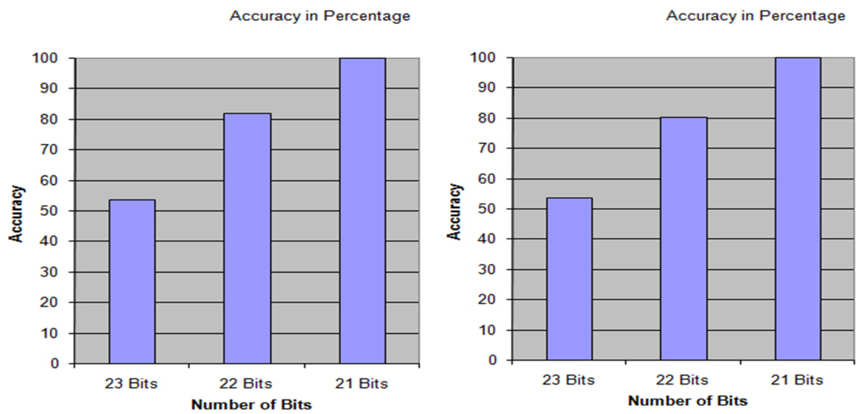
Sl	Angle (Degree)	Angle in Radian (23-bit)*	Sine(x)*	Sine(x) (Decimal)	Cosine(x)*	Cosine(x) (Decimal)
01	0.20	00000000111001001100001	0000000011100100110000 1	0.00349065	11111111111111111001101	0.99999390
02	2.60	00001011100111011110110	0000101110011100111010 0	0.04536298	11111111101111001000101	0.99897062
03	3.750	00010000110000010101001	0001000010111110010001 0	0.06540298	11111111011100111010111	0.99785892
04	5.000	00010110010101110001100	0001011001001111110110 0	0.08715581	11111111000001101001111	0.99619469
05	5.458	00011000011000101111011	0001100001011001100001 1	0.09511606	11111110110101101101111	0.99542125
06	6.743	00011110001000001100010	0001111000001110111110 1	0.11741607	11111110001110101010111	0.99308276
07	7.85	00100011000100101111101	0010001011110110111010 1	0.13658011	11111101100110011101111	0.99062902
08	8.59	00100110011000010110011	0010011000111100101000 1	0.14936277	11111101001000001101101	0.98878246
09	10.372	00101110010101111011000	0010111000010111000000 0	0.18003846	11111011110100010001111	0.98365957
10	12.895	00111001100111011000101	0011100100100001010110 0	0.22316505	11111001100010110011101	0.97478067
11	13.9475	00111110010100010110101	0011110111011010001010 0	0.24103271	11111000011100111100111	0.97051699
12	15.39	01000100110000110110000	0100001111110000011101 1	0.26538784	11110110110100011111111	0.96414173
13	16.595	01001010001001011010110	0100100100011101011001 0	0.28560473	11110101010101100100010	0.95834732
14	17.80	01001111100001111111101	0100111001000010000011 1	0.30569530	11110011101111101100000	0.95212939
15	19.03	01010101000001101110000	0101001101111000111000 1	0.32606318	11110010000000100101010	0.94534778
16	20.382	01011011000100010101000	0101100100101000101110 0	0.34827757	11101111111110001110010	0.93739128
17	22.75	01100101101001011110000	0110001011111111011111 0	0.38671096	11101100000101010101111	0.92220097
18	24.00	01101011001110111010100	0110100000011111111001 1	0.40673664	11101001110111100001111	0.91354545
19	26.67	01110111001010011010011	0111001011100111111010 1	0.44885116	11100100110000110110011	0.89360652
20	28.335	01111110100110100001110	0111100110000000100010 1	0.47462597	11100001010100111111100	0.88018758
21	29.23	10000010100110011101010	0111110100000010010101 0	0.48831653	11011111011001110001001	0.87266651
22	30.00	10000110000010101001001	1000000000000000000001 0	0.50000000	11011101101100111101011	0.86602540
23	31.57	10001101000011100101110	1000011000000110101101 1	0.52353987	11011010000111001100000	0.85200117
24	32.75	10010010010101000001001	1000101001111101010011 1	0.54097447	11010111010011100101010	0.84103901
25	35.50	10011110100111011001001	1001010010101000111101 0	0.58070283	11010000011010011110000	0.81411551
26	37.257	10100110011101110100001	1001101011111010111011 0	0.60539111	11001011110000011111001	0.79592804
27	38.82	10101101011100110000111	1010000001111010111011 1	0.62687581	11000111011101000101110	0.77911907
28	39.73	10110001100000111110111	1010001110100000101011 1	0.63917058	11000100111000010111001	0.76906499
29	42.365	10111101010010011110010	1010110010000001100001 0	0.67385104	10111101001001100110011	0.73886711
30	45.00	11001001000011111101101	1011010100000100111100 1	0.70710678	10110101000001001111010	0.70710678
31	47.283	11010011010000110011000	1011110000010110001011 1	0.73471334	10101101101010100010110	0.67837769

32	50.009	11011111011100010011111	1100010000100010000110 1	0.76614540	10100100100001011101101	0.64266727
33	51.252	11100100111111110000001	1100011110100111111100 1	0.77990633	10100000001110101011111	0.62589624
34	52.00	11101000010101101001011	1100100110111011000101 0	0.78801063	10011101100110111111111	0.61566147
35	54.50	11110011100000100010001	1101000001101001111000 0	0.81411551	10010100101010001111010	0.58070283
36	56.789	11111101101111000101011	1101011000101111010011 1	0.83665917	10001100001101111010001	0.54772386
37	58.43	10001101000011100101110	1101101000011100110000 0	0.85200117	10000110000001101011011	0.52353987
38	60.00	10000110000010101001001	1101110110110011110101 1	0.86602540	10000000000000000000010	0.50000000
39	61.665	01111110100110100001110	1110000101010011111110 0	0.88018758	01111001100000010001011	0.47462597
40	63.33	01110111001010011010011	1110010011000011011001 1	0.89360652	01110010111001111110101	0.44885116
41	66.00	01101011001110111010100	1110100111011110000111 1	0.91354545	01101000000111111110011	0.40673664
42	67.25	01100101101001011110000	1110110000010101010111 1	0.92220097	01100010111111110111110	0.38671096
43	69.618	01011011000100010101000	1110111111111000111001 0	0.93739128	01011001001010001011100	0.34827757
44	70.97	01010101000001101110000	1111001000000010010101 0	0.94534778	01010011011110001110001	0.32606318
45	72.20	01001111100001111111101	1111001110111110110000 0	0.95212939	01001110010000100000111	0.30569530
46	73.405	01001010001001011010110	1111010101010110010001 0	0.95834732	01001001000111010110010	0.28560473
47	74.61	01000100110000110110000	1111011011010001111111 1	0.96414173	01000011111100000111011	0.26538784
48	76.0525	00111110010100010110101	1111100001110011110011 1	0.97051699	00111101110110100010100	0.24103271
49	77.105	00111001100111011000101	1111100110001011001110 1	0.97478067	00111001001000010101100	0.22316505
50	79.628	00101110010101111011000	1111101111010001000111 1	0.98365957	00101110000101110000000	0.18003846
51	81.41	00100110011000010110011	1111110100100000110110 1	0.98878246	00100110001111001010001	0.14936277
52	82.15	00100011000100101111101	1111110110011001110111 1	0.99062902	00100010111101101110101	0.13658011
53	82.50	00100001100000101010010	1111110111001111010101 0	0.99144486	00100001011010100010101	0.13052619
54	83.257	00011110001000001100010	1111111000111010101011 1	0.99308276	00011110000011101111101	0.11741607
55	84.542	00011000011000101111011	1111111011010110110111 1	0.99542125	00011000010110011000011	0.09511606
56	85.00	00010110010101110001100	1111111100000110100111 1	0.99619469	00010110010011111101100	0.08715574
57	86.25	00010000110000010101001	1111111101110011101011 1	0.99785892	00010000101111100100010	0.06540298
58	87.40	00001011100111011110110	1111111110111100100010 1	0.99897062	00001011100111001110100	0.04536298
59	89.70	00000001010101110010010	1111111111111111000111 0	0.99998617	00000001010101110010010	0.00523596
60	89.80	00000000111001001100001	1111111111111111100110 1	0.99999390	00000000111001001100001	0.00349065

*Decimal point exists in the very left of the binary string

VII. CONCLUSION

The present work focuses mainly to reduce the number of clock cycles required to calculate the value of sine and cosine. The proposed system computes the sine and cosine values correctly up to 21 bits for all the inputs or more in only 6 clock cycles where as CORDIC architecture will require 21 clock cycles for the same. But the system resource utilization by the proposed methodology is more or less same compared to that of the



(a) (b)

Figure 5. Shows accuracy achieved 23, 22 and 21 bits a) Sine values b) Cosine values

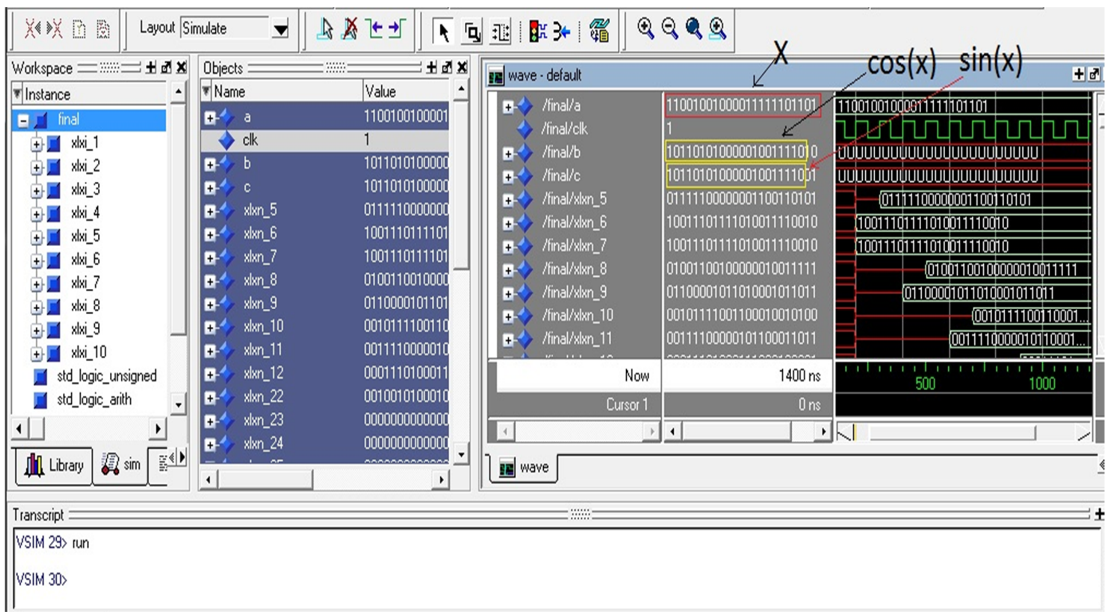


Figure 6. Snapshot of the ModelSim Simulation

CORDIC architecture. This work may be extended to design a complete module having memory unit for storing the data and a control unit to interface the computation unit and the memory unit.

ACKNOWLEDGMENT

Authors are thankful to the Government College of Engineering and Textile Technology, Berhampore, West Bengal for providing infrastructural facilities during progress of the work. Dr. Santanu Halder is thankful to Government College of Engineering and Leather Technology, Kolkata for kindly permitting him to carry on the research work.

REFERENCES

[1] J.E Volder, “The CORDIC Trigonometric Computing Technique”, IRE Transactions on Electronic Computers, EC-8:330-334, September 1959.

[2] Esteban O. Garcia, Rene Cumplido, Miguel Arias, “Pipelined CORDIC Design on FPGA for a Digital Sine and Cosine Waves Generator”, 3rd International Conference on Electrical and Electronics Engineering (ICEEE), 2006.

[3] P. K. Meher, J. Valls, T. B. Juang, K.Sridhan and K. Maharatna, “50 Years of CORDIC: Algorithms, Architectures, and Applications”, IEEE Transactions on Circuits and Systems, Vol. 56, No.9, pp.1893-1907, 2009.
[4] Javier Valls, Martin Kuhlmann, and Keshar K. Parhi, “Evaluation of CORDIC algorithms for FPGA design”, Journal of VLSI signal Processing. Vol. 32, no. 3, pp. 207-222, 2002.
[5] Shenk Al, Calculus and Analytic Geometry Scott Foresman & Co; 4th edition, July 2000.
[6] Milton Abramowitz and Irene A. Stegun, Handbook of Mathematical Functions Dover Publications Inc., New York, 1965.
[7] Frank W.J. Olver, Daniel W. Lozier, Ronald F. Boisvert and Charles W. Clark, The NIST Handbook of Mathematical Functions, Cambridge University Press, Cambridge, UK, 2010.