International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 1 (2018) pp. 21-29

Optimized Method for Sine and Cosine Hardware Implementation Generator, using CORDIC Algorithm

A. Ait Madi[1,2] and A. Addaim[1]

[1]ISET Laboratory, Ibn Tofail University, National School of Applied Sciences, Kenitra, Morocco.
[2]ERSI Laboratory, Sidi Mohammed Ben Abdellah University, Faculty of Sciences and Technology Fez, Morocco.

[2]Orcid: 0000-0002-9314-5932

Abstract

In this paper, we propose an efficient high level hardware implementation methodology for computing the trigonometric sine and cosine functions values of given angle using a more clever method called CORDIC (COordinate Rotation Digital Computer) algorithm.

Here we used the XSG (Xilinx System Generator), which provides a set of optimized Simulink blocks for several hardware operations. It automatically generates VHDL (Very High Speed integrated circuits Hardware Description Language) code corresponding to the proposed Sine and Cosine generator.

The proposed generator has been implemented on FPGA (Field Programmable Gate Arrays) Xilinx Spartan 6 XC6SLX16-3CSG324 device. System simulation is carried out using Xilinx ISE design Suite 14.3 and ISim tools.

The Hardware/Software co-simulation is performed to validate the functionality of the proposed design using Simulink environment. The proposed high level methodology is useful for developing hardware implementation solution in short time with optimized hardware resources.

Keywords: CORDIC, Sine, Cosine, XSG, FPGA, VHDL, Xilinx.

INTRODUCTION

NON-linear functions such as sine, cosine, tangent... appear in many problems (e.g Biometrics, Biomedical, Image processing, Robotics..., for end application such as Radar...). To compute these non-linear functions three techniques are used in the literature. The first is Taylor series which are slow and require floating point. The second is lookup tables which are fast but require memory or limited precision. The third one is CORDIC [1], [2] which is the best known widely used solution for hardware efficient implementation. It is generally faster than other approaches when a hardware multiplier is not available. On the other hand, when a hardware multiplier is available such in a DSP (Digital Signal Processor), lookup tables or Taylor series are generally faster than CORDIC. In recent years, the CORDIC algorithm has been used extensively for various applications. Due to the simplicity of the involved operations, the CORDIC algorithm is very well suited for VLSI implementation, especially in hardware implementations based on FPGA devices [3], [4], [5], [6], [7]. A very interesting review of CORDIC algorithm and its corresponding architectures for several applications is presented in [8], [9].

This CORDIC algorithm can be implemented using special arithmetic units such as shift registers, adders, subtractors and special interconnect, instead of multiplication operations which are much cost in term of digital hardware implementation. This implementation can be achieved by developing a VHDL program by hand. This method is a very difficult task and takes more time. To reduce the impact of these issues, high level synthesis (HLS) [10], [11] can be used to convert an algorithmic description such as C program to HDL code (hardware description language). A very interesting other method is the use of Simulink environment for hardware implementation in FPGA. This environment offers the possibility for performing hardware/software co-simulation and verifying implementation functionalities. Designing and developing by this method takes short time.

For implementation in a Xilinx FPGA targets (used in this paper), we can use XSG for DSP which offers a highly optimized library of blocks [9] that can be simulated within Simulink and then compiled for FPGA implementation. This XSG design flow offers higher performance than HDL Coder, because each block is pre-optimized IP (Intellectual Propriety) for Xilinx FPGAs.

This paper is organized in the following manner. Section 2 presents an overview of CORDIC algorithm. Related and proposed works are respectively presented in sections 3 and 4. The results and simulations are shown in the section 5. Finally, the section 6 concludes the paper.

CORDIC ALGORITHM AN OVERVIEW

CORDIC algorithm was first introduced by Volder [1] for the computation of trigonometric functions, multiplication, division, and later generalized to hyperbolic functions by Walther [2].

A. Conventional CORDIC Algorithm

The basic idea of CORDIC is to rotate a vector over given angle. This vector is rotated through a fixed number of steps called iterations. As seen in the figure 1, if a vector V , having coordinates x_0 and y_0 , is rotated through an angle ϕ , we obtain a new vector V_ϕ with coordinates x_ϕ and y_ϕ .

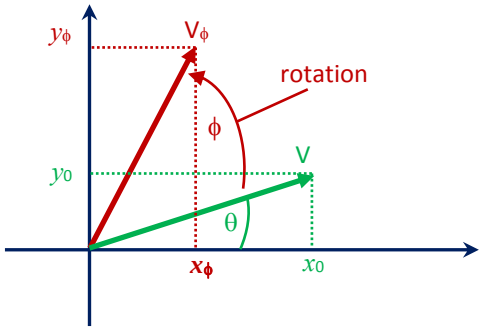

Figure 1: An illustration of rotation's vector V by the angle ϕ

If we set the initial coordinate values as $x_0 = r\cos\theta$ and $y_0 = r\sin\theta$, where r and θ are respectively the magnitude and angle corresponding to the vector V, the final coordinate values according to the rotated vector V_ϕ are given, bellow, by the matrix 1.

$$V_\phi = \begin{bmatrix} x_\phi \\ y_\phi \end{bmatrix} = \begin{bmatrix} r\cos(\theta+\phi) \\ r\sin(\theta+\phi) \end{bmatrix} = ROT(\phi)\begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \quad (1)$$

Where, $ROT(\phi)$, is the rotation matrix, given by the equation 2.

$$ROT(\phi) = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} \quad (2)$$

In CORDIC algorithm, as shown in the equation 3, we decompose the given rotation angle ϕ on micro-rotation ϕ_i.

$$\phi = \sum_{i=0}^{n-1} \phi_i \quad (3)$$

Where n is the number of iterations required to successively rotate a vector to reach the angle ϕ. The rotation matrix, $ROT(\phi)$, will be computed by using equation 4.

$$ROT(\phi) = \prod_{i=0}^{n-1} ROT(\phi_i) \quad (4)$$

As shown in the figure 2, at each iteration i, this micro-rotation ϕ_i is performed by the vector V_i, to obtain a new vector V_{i+1} specified by its coordinate values given by the matrices 5 and 6.

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} \cos\phi_i & -\sin\phi_i \\ \sin\phi_i & \cos\phi_i \end{bmatrix}\begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad (5)$$

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \cos\phi_i \begin{bmatrix} 1 & -\tan\phi_i \\ \tan\phi_i & 1 \end{bmatrix}\begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad (6)$$

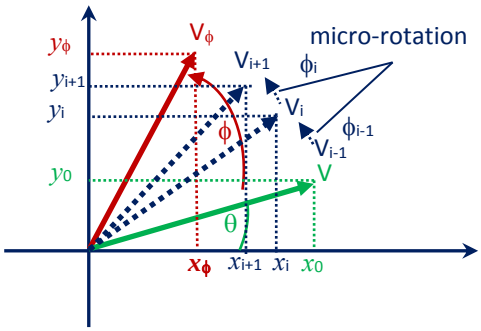

Figure 2: An illustration of micro-rotation's vector

If $\tan\phi = 2^{-i}$, multiplications could be performed by a simple shift operations in digital computer hardware. The vector V is rotated by the desired angle ϕ in a sequence of smaller rotations corresponding to the angle $\phi_i = \tan^{-1}(2^{-i})$.

From equation 6, we can obtain two iterative equations shown in matrix 7.

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = k_i \begin{bmatrix} x_i - y_i d_i 2^{-i} \\ y_i + x_i d_i 2^{-i} \end{bmatrix} \quad (7)$$

Where k_i is the CORDIC gain defined by $k_i = \cos\phi_i = \cos(\tan^{-1}(2^i))$. By applying some trigonometric rules we can write k_i as shown in the equation 8.

$$k_i = \frac{1}{\sqrt{1+2^{-2i}}} \quad (8)$$

The variable d_i is the direction of micro-rotation angle, for i iterations, which can takes +1 or −1.

B. Rotation Mode in CORDIC Algorithm

Two basic CORDIC modes are known leading to the computation of different functions, the rotation mode and the vectoring mode [1], [2]. In our work, we consider CORDIC algorithm in its rotation mode in order to compute the trigonometric functions especially cosine and sine. The CORDIC gain k_i can be ignored in the iterative process and then applied afterward with the final value, k_n,which was computed on offline for all iterations and given by equation 9.

$$k_n = \prod_{i=0}^{n-1} k_i = \prod_{i=0}^{n-1} 1/\sqrt{1+2^{-2i}} \quad (9)$$

In order to compensate the gain, to scale the result, some applications avoid the use of the gain in favor of the final value of scaling factor, A_n, which was computed also on offline for all iterations and given by equation 10.

$$A_n = 1/k_n = \prod_{i=0}^{n-1} \sqrt{1+2^{-2i}} \tag{10}$$

In rotation mode, the objective is to rotate the given initial vector from its initial position to its final position corresponding respectively to a given angle $\theta(x_0, y_0)$ and $\phi(x_\phi, y_\phi)$, through series of iterations. The micro-rotation ϕ_i at each iteration is made to reduce the final accumulator angle value (see the expression in the equation 13), noted z_n, to zero. The final x_ϕ and y_ϕ coordinate values of a point on the unit circle corresponding to the desired input angle ϕ must be found. As seen in the figure 3, the pseudo-rotation produces a vector V'_{i+1} with the same angle as the rotated vector V_{i+1}, but with a different magnitude. After performed a micro-rotation ϕ_i, we obtain a vector V'_{i+1}, which is defined by its coordinate values shown in the CORDIC iterative equations presented by the matrix 11.

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} x_i - y_i d_i 2^{-i} \\ y_i + x_i d_i 2^{-i} \end{bmatrix} \tag{11}$$

The computation of the accumulator angle adds a third equation to a CORDIC algorithm, which is defined by equation 12.

$$z_{i+1} = z_i - d_i \tan^{-1}(2^{-i}) \tag{12}$$

where the micro-rotation direction value $d_i = \text{sgn}(z_i)$.

In the first iteration, the accumulator angle $z_i = z_0$ is initialized with the desired input rotation angle ϕ. The remaining angle, by which the vector needs to be rotated after the completion of iteration i, is indicated by z_{i+1}. After a sufficient number of iterations, we obtain the equation 13.

$$z_n = z_0 - \sum_{i=0}^{n-1} d_i \tan^{-1}(2^{-i}) \tag{13}$$

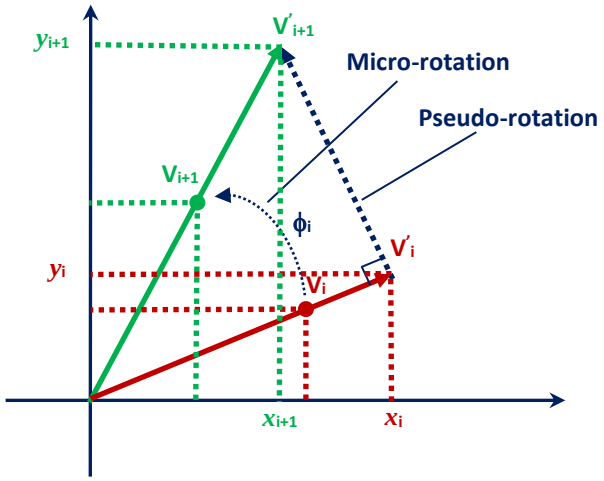

Figure 3: An illustration of micro-rotation and pseudo-rotation concept

If $z_n = 0$ holds, then the total accumulate rotation angle is equal to the input angle ϕ given in equation 14.

$$\phi = z_0 = \sum_{i=0}^{n-1} d_i \tan^{-1}(2^{-i}) \tag{14}$$

To drive z_n to zero value, micro-rotation, which depends on the sign of parameter z_i, should be clockwise or counterclockwise at each iteration. If z_i is positive, the next micro-rotation will be clockwise, otherwise it will be counterclockwise.

An example of accumulator angle z_i computation is shown below in table 1. In this example, we consider a given input angle $\phi = 30°$. After 10 iterations, the final accumulator angle z_{10} is going to $+0.0708547792$, which approaches zero value. Therefore, the final CORDIC rotation angle will be near $\phi = 30°$.

The first three iterations to illustrate pseudo-rotations process in order to reach the rotation angle near $\phi = 30°$, is given in figure 4. In this figure, we consider that the specific micro-rotation angles ϕ_i shown in table 1 are represented by their rounding format. We start with the initial vector V_0, defined by its coordinate values $(0,1)$. For the first iteration, the positive micro-rotation angle $\phi_0 = +45°$ is performed by the initial vector V_0, to obtain the vector, V_1. Since $\phi_0 = +45° > 30°$, then in the second iteration, the negative micro-rotation angle $\phi_1 = -26.6°$ is performed by the vector V_1 to obtain the vector V_2. The accumulate angle is

now $\phi_0 + \phi_1 = +45° - 26.6° = 18.4°$,which is smaller than $30°$.

Table 1: Example of micro-rotations process to force the final accumulator angle to zero

i	ϕ_i	z_i	d_i	$z_{i+1} = z_i - d_i\phi_i$
0	45.0	+30.0	+1	-15.0
1	26.56505118	-15.0	-1	+11.56505118
2	14.03624347	+11.56505118	+1	-2.47119229
3	7.125016349	-2.47119229	-1	+4.653824059
4	3.576334375	+4.653824059	+1	+1.077489684
5	1.789910608	+1.077489684	+1	-0.712420924
6	0.89517371	-0.712420924	-1	+0.182752786
7	0.447614171	+0.182752786	+1	-0.264861385
8	0.2238105	-0.264861385	-1	+0.041050885
9	0.111905677	+0.041050885	+1	+0.0708547792

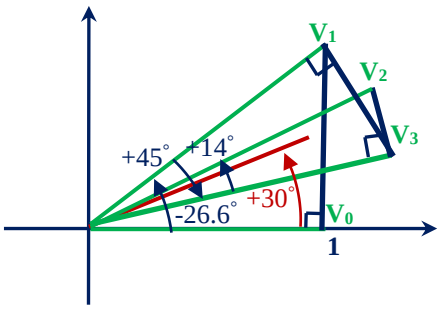

Figure 4: The first three micro-rotations to converge toward the given angle, $\phi = 30°$

In the third iteration, the positive pseudo-rotation angle $\phi_2 = +14°$ is applied to rotate V_2 to V_3. In this step, the accumulate angle is $\phi_0 + \phi_1 + \phi_2 = 32.4°$. The computation process continue so on, the more iteration we take, the better the approximation that we can make by successive micro-rotations. After 10 iterations the obtained final accumulate angle value, according to the directions of the micro-rotations is given in equation 15.

$$\phi = 30^\circ \approx \sum_{i=0}^{9} d_i\phi_i = 30.1^\circ \tag{15}$$

Once z_n is driving to zero, the final x_n and y_n values leading to the computation of the coordinate values of the vector corresponding to the given input angle φ are given in the equations 16 and 17.

$$x_n = A_n\left(x_0 \cos z_0 - y_0 \sin z_0\right) \tag{16}$$

$$y_n = A_n\left(y_0 \cos z_0 + x_0 \sin z_0\right) \tag{17}$$

Since we have $z_0 = \phi$, the final coordinate values, x_ϕ and y_ϕ, corresponding to a given input angle ϕ are provided by equations given in 18 and 19.

$$x_\phi = \frac{x_n}{A_n} = x_0 \cos z_0 - y_0 \sin z_0 \tag{18}$$

$$y_\phi = \frac{y_n}{A_n} = y_0 \cos z_0 + x_0 \sin z_0 \tag{19}$$

If we start the CORDIC algorithm with $x_0 = 1/A_n$ and $y_0 = 0$, the cosine and the sine values of the given angle ϕ will be, at the end of the process, $x_n = \cos z_0 = \cos\phi$ and $y_n = \sin z_0 = \sin\phi$.

The flowchart given in figure 5 illustrates the steps of CORDIC algorithm for computing sine and cosine functions.

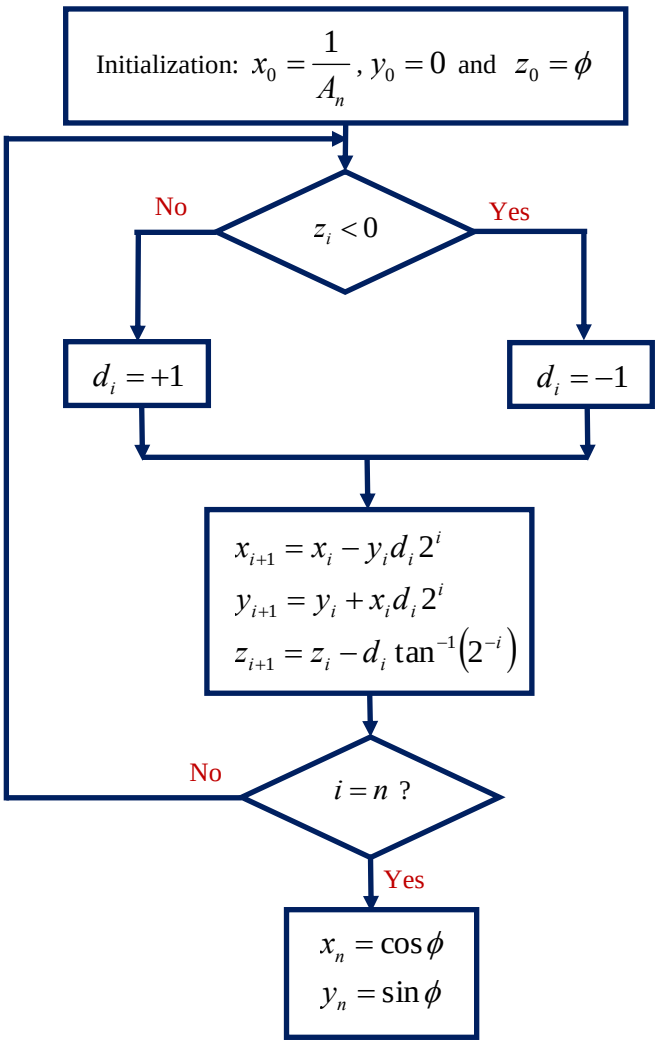

Figure 5: Flowchart of CORDIC algorithm for sine and cosine generator

RELATED WORK

Many different architecture and hardware implementations of the CORDIC algorithm were made for trigonometric functions in the literature, especially, for computing the sine and cosine functions values for a given angle. Vipin Tiwari et al. [3], presented a hardware design that calculates sine and cosine values of a given angle using CORDIC algorithm. The values of both input and the two outputs are represented in fixed-point notation, with11 bits for the given input angle and16 bits for the sine and cosine outputs.

Manjunath et al. [5] proposed a simplified pipelined CORDIC architecture model for computing sine and cosine values. The proposed CORDIC algorithm model helps in achieving reduced On-chip area and power consumption. Its maximum frequency closes 228.7 MHz.

Tanya Vladimirova et al. [6], presented synthesis results of iterative and cascaded CORDIC sine and cosine generators, for both Actel and Xilinx FPGAs. The bit-lengths used were 12,14,16,24,32 bits for the iterative designs and 12,14,16 bits for the cascaded designs.

Kavya Sharat et al. [7] proposed FPGA CORDIC algorithm implementation for calculating the sine and cosine of an angle represented using bit-length equal to 8 bits. The maximum frequency was 241.106 MHz.

Ajakida Eski et al. [12] presented the real time implementation of CORDIC algorithm, in ARM Cortex MO, PSoC4, CY8C4245AXI-483 architecture, to realize the valuation of the sine and cosine functions compared to the polynomial approximation algorithm of Taylor series. A detailed comparison between the two methods shows that CORDIC algorithm is 2.3 times faster and occupies 47.1% less memory.

Among the works presented in the literature, there is very few of results presented for hardware sine and cosine generators using High level methodologies.

PROPOSED WORK

We proposed here high level methodology to generate hardware cosine and sine functions using Xilinx Spartan 6 XC6SLX16I-3CSG324 device. Rotation mode is considered for CORDIC algorithm. The angle is given as input and its corresponding rectangular coordinates, as outputs, are computed using the steps shown in the flowchart above given in figure 5. The proposed work is illustrated in figure 6. As seen in this figure, we used the Xilinx CORDIC 5.0 block [10]. This optimized CORDIC block deploys coarse rotation to perform input angle rotation from the full circle into the first quadrant. The coarse rotation stage is required as the CORDIC algorithm is only valid over the first quadrant. An inverse coarse rotation stage rotates the output sample into the correct quadrant. We select a CORDIC fully parallel configuration with maximum pipelining.

The CORDIC input angle (in radians unit), *in_pha*, is expressed with a length of 8 bits [5] represented in fixed-point 2's complement with an integer part size of 3 bits [10], [11] and fractional part size of 5 bits.

The outputs (*out_sine* and *out_cosine*) width has been configured with fixed-point 2's complement numbers with size of 16 bits [3], [4] arranged with integer parts size of 2 bits [10], [11] and the fractional parts size of 14 bits.

The Xilinx single port ROM (Read only Memory) [10], which allows read access to the memory space through a single port, is used to store128 angle values as a memory depth with data width of 8 bits. Its memory content is initialized by the initial vector $\left[-\pi\times(64:-1:1)/64, \pi\times(0:64)/64\right]$ to cover the range of $-\pi$ to π, supported eventually by Xilinx CORDIC block in radians phase format. By using a free running Xilinx Counter block [10], with size of 7 bits to address a ROM memory, we can select stored angle values one-by-one and then forwarded also one-by-one to the CORDIC block at the positive edge of each clock pulse.

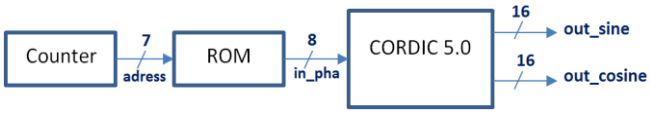

Figure 6: Proposed architecture for sine and cosine generator

RESULTS AND SIMULATIONS

To carry out the simulation results, we use the XSG design flow given in the figure 7. In the MATLAB/Simulink GUI (Graphic User Interface) environment, we configure the sine and cosine generator Xilinx blocks parameters. Using the XSG, the GUI allows both RTL (Register Transfer Level) and test bench VHDL automatic code generation. We insert the hardware design in the loop in order to verify and simulate its functionality, by using Xilinx ISE design suite14.3 and ISim tools, and MATLAB/Simulink environment for displaying the hardware co-simulation results.

Figure 8 presents the VHDL automatic generated component corresponding to the sine and cosine generator.

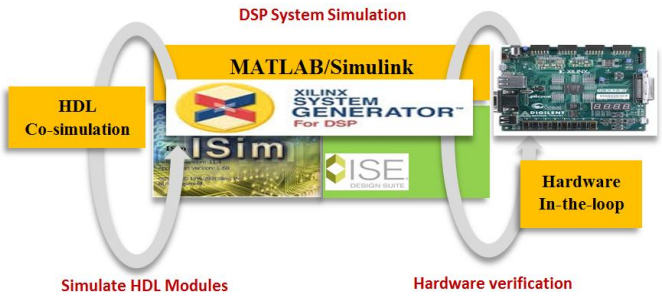

Figure 7: Xilinx System generator design flow

```
component cordic_5_0_sin_cos_generation_co_simulation_cw  port (
    ce: in std_logic := '1';
    clk: in std_logic; -- clock period = 10.0 ns (100.0 Mhz)
    in_val: in std_logic;
    in_pha: out std_logic_vector(7 downto 0);
    out_cosine: out std_logic_vector(15 downto 0);
    out_sine: out std_logic_vector(15 downto 0);
    out_val: out std_logic
  );
end component;
```

Figure 8: The automatic generated component

Examples of simulation results are shown in the figure 9 (for displaying the inputs) and figure 10 (for displaying the outputs). To see the complete simulation, the two figures will be concatenated. The inputs and outputs are represented in hexadecimal. In figure 9, *in_val* signal is initially set to be kept high till the outputs are available. The test vectors applied to the input, *in_pha*, are the contents of the ROM memory. As seen if we concatenated the two figures 9 and 10, after 19 periods of clock, *clk*, *out_val* signal is set to indicate that the outputs are available.

To verify the proposed design functionality, we take an example from these figures: for *in_pha*=9f=10011111= −3.03125 rad, the corresponding outputs are *out_cosine*= c064 = 1100000001100100 = −0.993896484375 and *out_sine*= f8f5 = 1111100011110101 = −0.11004638671875. These values are in agreement with MATLAB/Simulink. The verification of the remaining examples in these figures, confirms that the VHDL behaves identically to MATLAB/Simulink.

By using the XSG block for DSP co-simulation mode, we include the hardware design in the loop as shown in figure 7. Figures 11, 12, 13 and 14 present the inputs/outputs for Simulink and FPGA design. As seen in these figures the carried out results from hardware/software co-simulation using both MATLAB/Simulink and hardware in the loop are in agreement. The plot graph in the figure 15 which presents *out_sine* from FPGA as function of *out_sine* from Simulink is a linear function with slope and intercept equal respectively to 1 and 0. Consequently, the hardware design generated by the proposed high level methodology works well with optimized hardware resources.

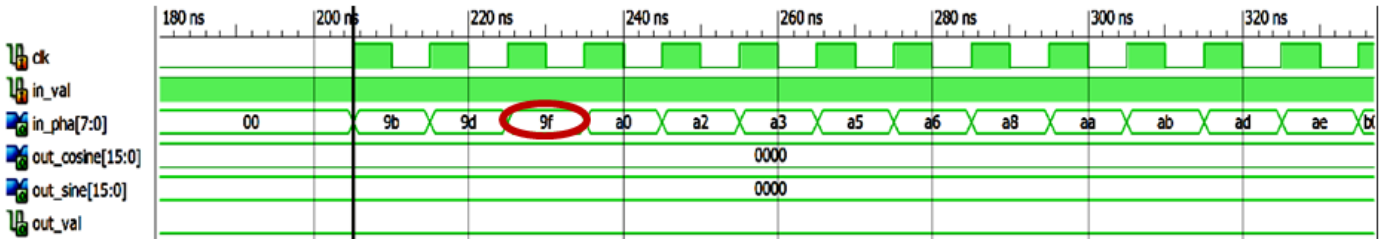

Figure 9: Example of Simulation result carried out from Xilinx Isim tool environment: for displaying the inputs

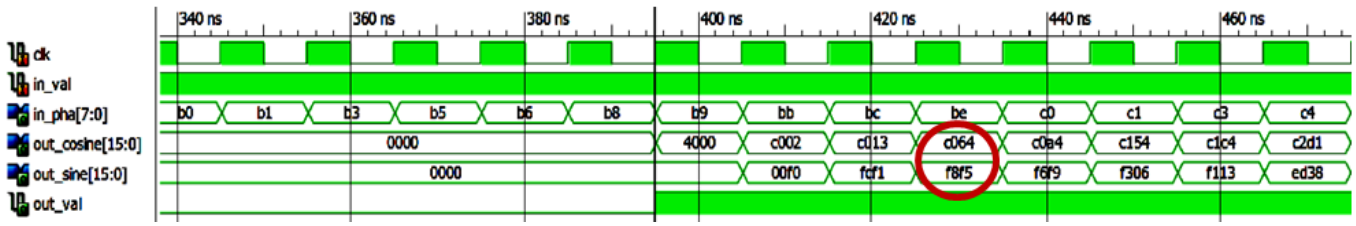

Figure 10: Example of Simulation result carried out from Xilinx Isim tool environment: for displaying the outputs

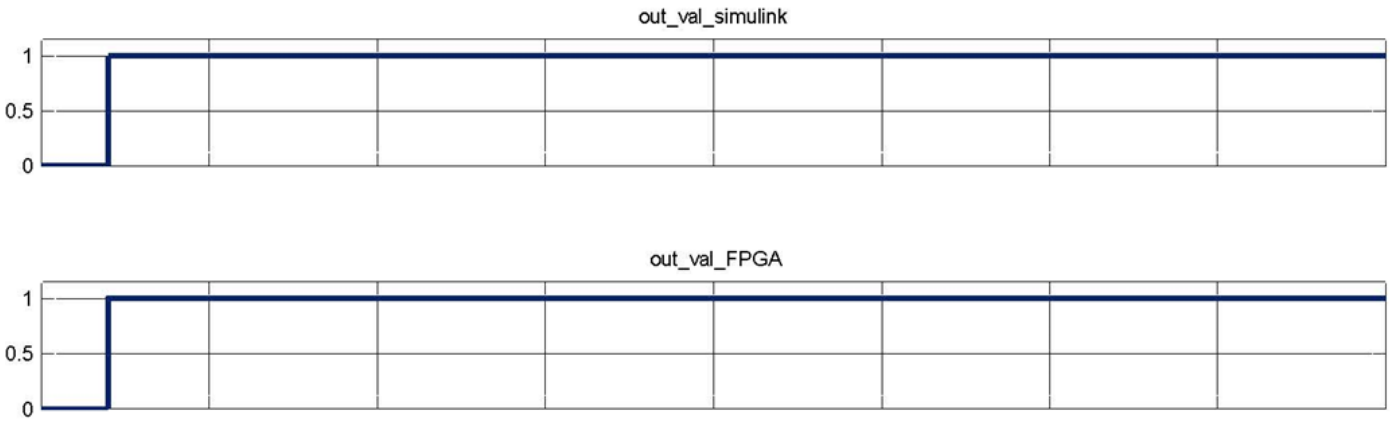

Figure 11: Validation outputs From Simulink and FPGA

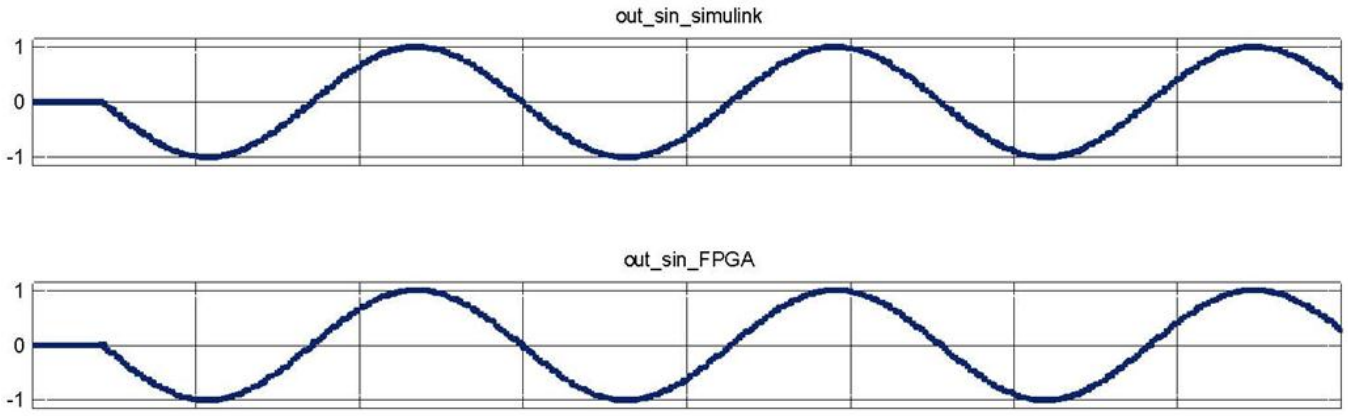

Figure 12: Sine outputs From Simulink and FPGA

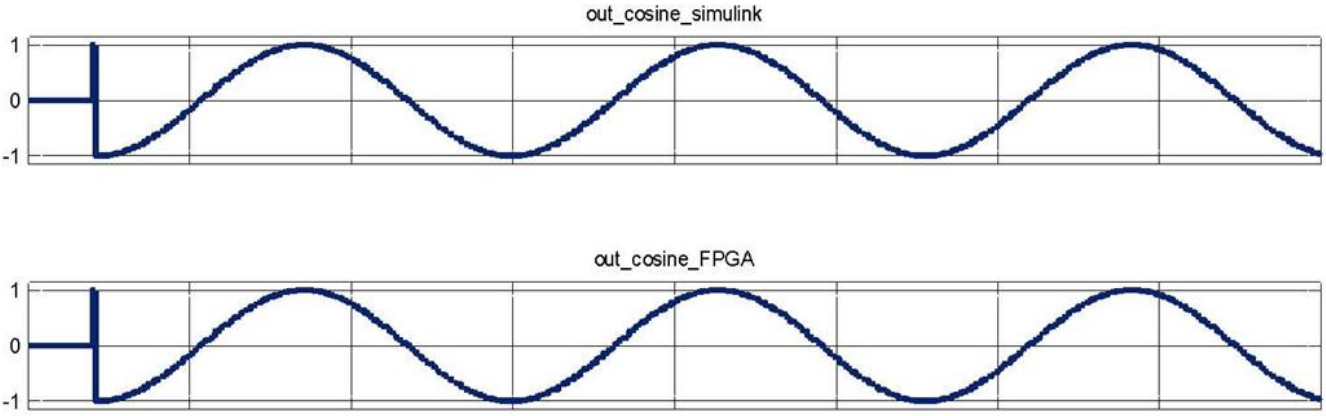

Figure 13: Cosine outputs From Simulink and FPGA

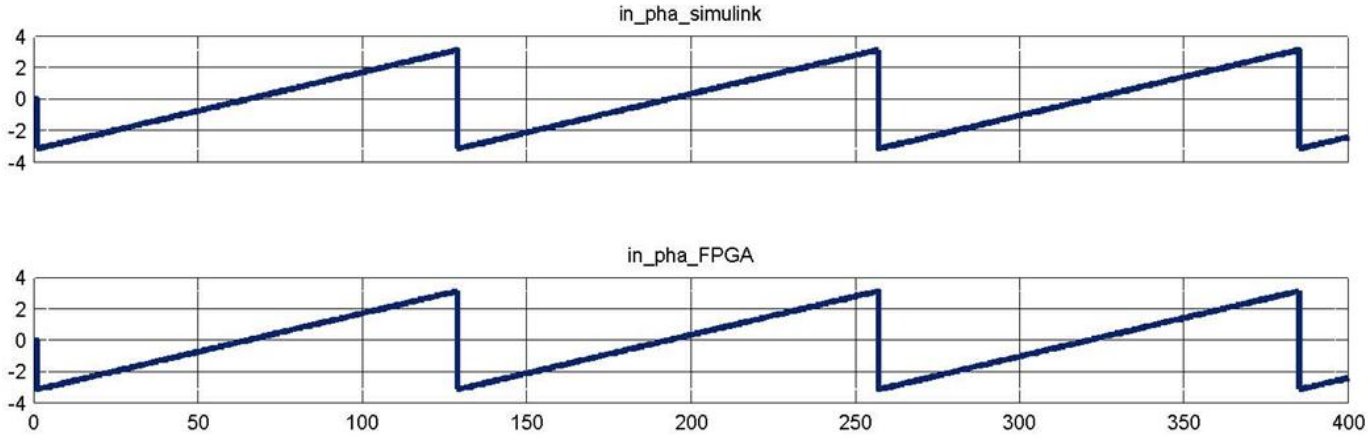

Figure 14: Test vectors applied to the inputs of both Simulink and FPGA design

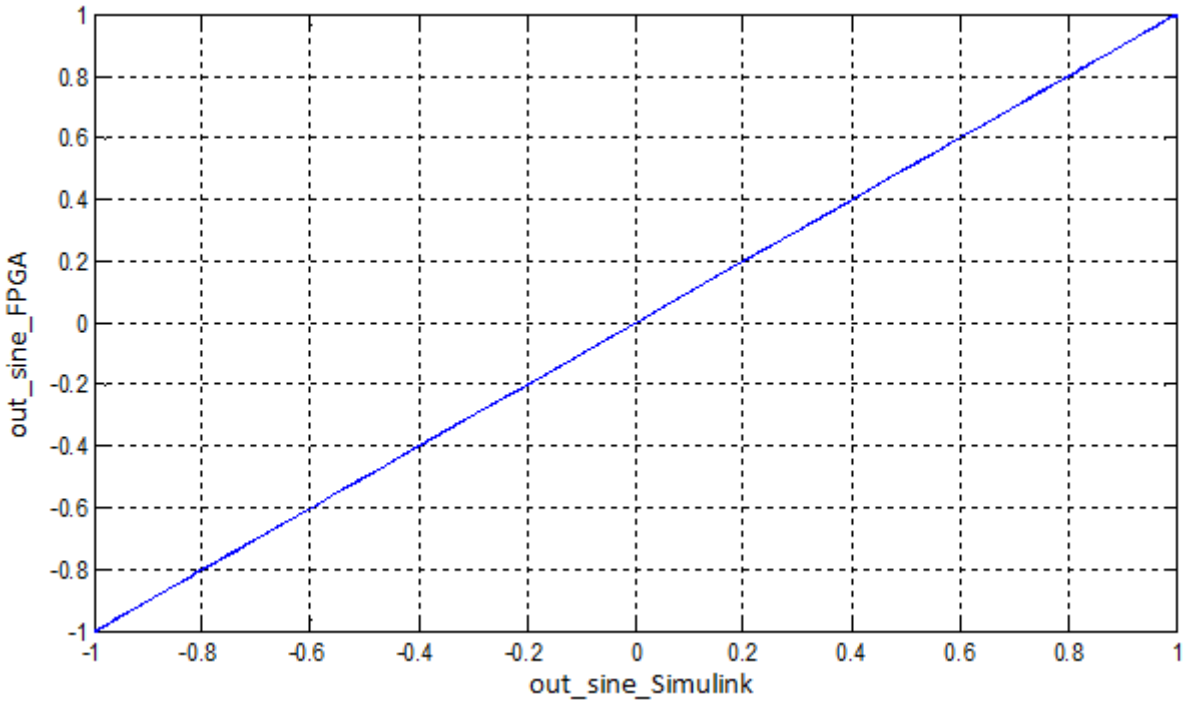

Figure 15: Output from FPGA as function of output from Simulink

CONCLUSION

The CORDIC algorithm presented here is the best known solution for computing the linear, trigonometric and hyperbolic functions in hardware implementations. In this work, we have presented a high level methodology to implement sine and cosine generator by using optimized CORDIC 5.0, ROM and Counter Xilinx blocks. The MATLAB/Simulink and Xilinx ISE design suite 14.3 tools are used to implement the proposed architecture in Xilinx Spartan 6 XC6SLX16-3CSG324 device. The carried out results, by using hardware/software co-simulation, shows that our architecture has worked very well with optimized hardware resources and maximum frequency approaching 317.46 Mhz. We conclude that the adopted high level methodology is efficient and useful for developing hardware implementation solution in short time.

REFERENCES

[1] J. E. Volder, 1959, "The cordic trigonometric computing technique", IRE Transactions on Electronic Computers, 8(3), pp. 330-334.

[2] J. S. Walther, 1971, "A unified algorithm for elementary functions", Spring Joint Computer Conf., pp. 379-385.

[3] Vipin Tiwari, Nilay Khare , 2014, "Hardware for Calculation of SIN and COSINE Angle using CORDIC Algorithm", International Journal of Computer Applications, 87(6).

[4] Ranjita Naik, Riyazahammad Nadaf , 2015 , "Sine-Cosine Computation Using CORDIC Algorithm", International Journal of Advanced Research in Computer and Communication Engineering 4(9).

[5] C.R.Manjunath,Sathiyapriya, 2014, "FPGA Implementation of Sine and Cosine Value Generators using CORDIC Design for Fixed Angle Rotation", International Journal of Computer Applications, International conference on Innovations in Information, Embedded and Communication Systems.

[6] Tanya Vladimirova and Hans Tiggeler, 1999, "FPGA Implementation of Sine and Cosine Generators Using the CORDIC Algorithm", Proc. Of Military and Aerospace Applications of programmable devices and technologies conference (MAPLD99), session A2, Military and aerospace applications, pp. 28-30.

[7] Kavya Sharat, B. V. Uma and D. M. Sagar, 2014, "Calculation of Sine and Cosine of an Angle using the CORDIC Algorithm", International Journal Of Innovative Technology and Research(IJITR), 2(2).

[8] Rajkumar Tomar, Praveen Kumar Singh and Krishna Raj,2013, "A Review of CORDIC Algorithms and Architectures with Applications for Efficient Designing", International Journal of Scientific and Engineering Research (IJSER), 4(8).

[9] Pramod K. Meher, Javier Valls, Tso-Bing Juang, K. Sridharan, and Koushik Maharatna, 2009, "50 Years of CORDIC: Algorithms, Architectures, and Applications", IEEE Transactions On Circuits and Systems-I O : Regular Papers, 56(9).

[10] Vivado Design Suite Reference Guide Model-Based DSP Design using System Generator, UG958 (v2012.3) November 16, 2012,

https://www.xilinx.com/support/documentation/sw manuals/xilinx2012 3/ug958-vivado-sysgen-ref.pdf

[11] CORDIC v6.0 LogiCORE IP Product Guide Vivado Design Suite, PG105 October 5, 2016. https://www.xilinx.com/support/documentation/ipdocu mentation/cordic/v6 0/pg105-cordic.pdf

[12] Ajakida Eski, Denald Komici, Orion Zavalani,2017, " Evaluation of CORDIC Algorithm for the processing of sine and cosine functions", International Journal of Business and Management Invention (IJBMI) 6(3), PP. 50-54.