Calculation Method for Trigonometric Functions in FPGAs

Andrés Figueroa F. [1], J.J. Raygoza-Panduro[1], Ehecatl J. Chávez M.[1], S. Ortega-Cisneros[2], Jorge Rivera D.[1], Bernardino Castillo T.[2]

Abstract— **Trigonometric functions are used to solve several computing problems. The algorithms that determine these functions require a good precision and accuracy for these systems, and it is also necessary to have shorter response times and smaller latencies.**

This paper proposes a methodology for trigonometric functions implementation in reconfigurable FPGA devices, using preloaded data tables in ROM memory blocks on the device's embedded memory.

The developed modules can be handled as independent modules and can reconfigure precision characteristics through firmware upgrades.

Keywords—**Trigonometric function, FPGA, Memory blocks.**

I. INTRODUCTION

Among the several tasks of computers, one example is processing data information for calculating mathematical functions. It has been observed in some surveys [1] that dedicated hardware and parallelized modules in FPGA devices can show better development for these tasks.

Systems that calculate trigonometric mathematical functions consider some characteristics, such as precision and range in order to present desired results. The circuit interconnections increase and the utilization area on the FPGA is determined by these characteristics[7].

II. IMPLEMENTATION

Implemented functions are the sinusoidal functions, where sine and cosine functions have a mathematical relation. The cosine function is also a sine function with a phase-shift of 90 degrees. Because of this relation and using the same methodology, both developed modules have a similar construction.

The methodology followed to determine sine and cosine trigonometric functions is related to pseudo-operations using preloaded data tables in ROM memory blocks on the FPGA device's embedded memory [8]. Some purposes are to reduce maximum delay time and data-flow latency and simplify implemented algorithms for required results [9],[11].

Departamento de Electrónica, CUCEI, Universidad de Guadalajara[1], (ffaandy@gmail.com), (ehecatl.joel@gmail.com), {juan.raygoza, jorge.rivera }@cucei.udg.mx,

Centro de Investigación y de Estudios Avanzados del I.P.N. Unidad Guadalajara[2], {sortega, toledo}@gdl.cinvestav.mx

Building a value-matrix, where the values are calculated using algorithms that involve sine and cosine trigonometric modules are developed, and the goal is to estimate speed and precision for continuous operation.

A. *Arithmetic Module Description*

Each arithmetic module performs the calculation of two functions (sine and cosine). The result is obtained from a pre-loaded table, located in an embedded ROM memory module available in the FPGA device (Virtex4, model XC4vlx15-12sf363). The "result" from the operation is selected from the table through a process that converts the input angle value into a memory address.

B. *Black Box Description*

The desired specifications for the device lead to the following design, seen as a black box device (Fig. 1).

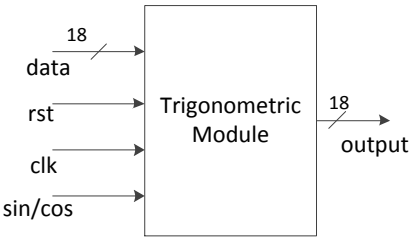

Fig. 1. Black Box Diagram for the Trigonometric Module.

C. *Ports*

- Data: 18-bits input bus, seen Figure 2.
- Reset: Reset input signal.
- Clk: Clock input signal.
- Sin/cos: Input select sine or cosine function.
- Output: 18-bits output bus that will show the result of a cosine/sine operation over a given angle, seen Figure 3.

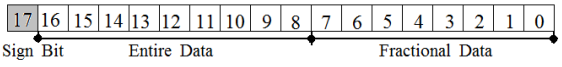

Fig. 2 Input Bus array for the Trigonometric Module.

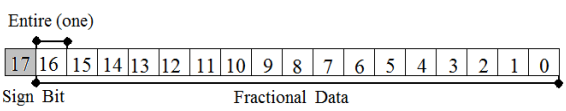

Fig. 3 Output Bus array for the Trigonometric Module.

D. Functional Description

According to the required specifications, the following operating characteristics will be developed:

When data is placed in the data bus device (data) representing a certain degree, it is accepted as a signed number in a two complement format, with a precision of 0.05 °, the output port (output) output will show the value of the trigonometric function, stored in an embedded memory address. This output value corresponds to the angle of entry.

E. Block Diagram

The method chosen to develop the device consists of developing 3 individual modules that will be contained in a *top-level module.*

Figure 4 shows the functional blocks of the trigonometric module. The Complement/Locator block receives a value degree and the two complement is done if it is a negative degree. This block is also needed to locate the memory address where the function value was stored. The ROM memory is an embedded memory created with the IPCore Generator tool. Due to angle precision, memory space grows when a better precision for the arithmetic module is needed.

The supplement block receives the sign, entire and "douta" signals and produces final module response output for the trigonometric module.

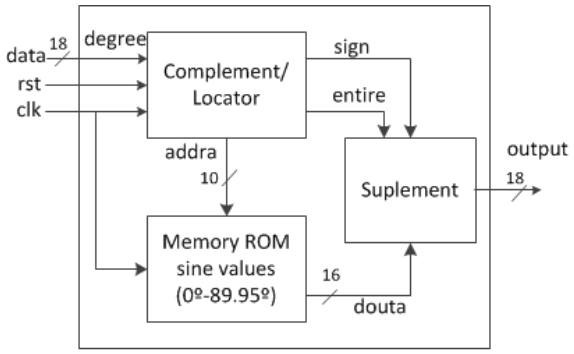

Fig. 4 Block diagram for the Trigonometric Module.

III. TRIGONOMETRIC FUNCTIONS

Figure 5 shows the unit circle with radio ***r***, whose center is at the origin of ***x*** and ***y*** axes of an orthogonal system.

The sine of a real number *t*, where **| *t* |** is the length of the arc shown, is given by the magnitude of ***a*** or the coordinate of the point $\mathbf{P_1}$, while the cosine of ***t*** is given by the magnitude of ***b*** or the ***x*** coordinate of the point $\mathbf{P_1}$ [2].

t being the arc of the unit circle shown, the range of values that it may acquire is between 0° and 360°.

As shown, the values of the magnitudes for the sine and cosine will always be between **-1** and **1**.

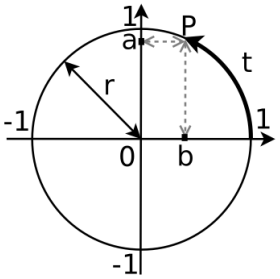

Fig.5. Unitary Circle.

A. Sine and Cosine Functions

The features for the development of the module that calculates the function $sin(t)$ are as follows:

- Odd Function:

$$sen(-t) = -sen(t) \quad (1)$$

- Maximum: 90º (1)
- Minimum: 270º (-1)

The features for the development of the module that calculates the function $\cos(t)$ are as follows:

- Even Function:

$$\cos(t) = \cos(-t) \quad (2)$$

- Maximum: 0º, 360º (1)
- Minimum: 180º (-1)

B. Relationship between Sine and Cosine Functions

To calculate the values of the cosine function, we use this trigonometric equivalence:

$$\cos(t) = \sin(t + 90^o) \quad (3)$$

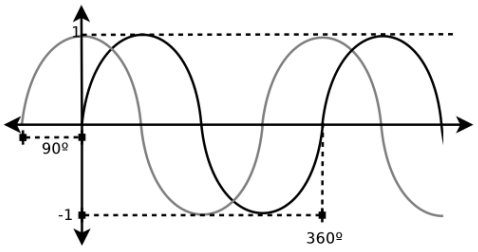

Fig.6. Cosine and Sine functions and its correspondence.

For more information about trigonometric functions, please refer to [3].

IV. GENERAL OPERATION

To develop the desired circuit, the chosen method consists of avoiding the arithmetic of trigonometric functions. To achieve this goal, we used a series of values, preloaded into ROM memory modules. We will use the memory modules available in the FPGA (Virtex-4)[3]. These modules will receive the value of the arc ***t*** and based on this number the system will access a memory locality and its value will be presented at the output port.

A. *Trigonometric Functions Calculation Method*

In order to reduce the size of the memory used, instead of a block of rows with values for every possible input value ***t***, we use the properties of symmetry and frequency of functions (1), (2) y (3).

Both functions are periodic, so there is no point in calculating values beyond $t = 360^{o}$. The sine function is odd, so it is not necessary to allocate results for negative values of ***t***. We simply change the sign of the output that would be obtained for the same positive value of ***t***. The cosine function is even, so the results relay only the magnitude of t and not on its sign. By having an input accuracy of 0.05 °, twenty memory addresses are needed to calculate the values of trigonometric function for every input degree. Even calculating only positive values from 0 to 360 ° for ***t***, the device will need an embedded ROM memory space of 7200 block addresses.

To calculate any angle, it is sufficient to know the values of the functions for the first 90° and then convert the requested angle to its complementary or supplementary angle. Thus, the amount of used ROM memory blocks is reduced to 1800 instead of 7200. To calculate the values of the cosine function, we will use the same equivalence trigonometric of (3).

B. *Complementation Methodology*

The methodology used for the complementation operated on the signals representing the angle is as follows:

• If the angle is negative, with a two complement, it is complemented to its positive value.

• The input angle in positive form is multiplied by twenty, because it is the memory space needed to calculate trigonometric function values for every degree.

• If the angle is greater than 90 º, the system proceeds to get its complementary / supplementary angle below 90 º.

• This angle is converted to a value that represents a memory location into the ROM memory in the FPGA.

• For the Cosine module the entry angle will add up to 90 º.

C. *Sine and Cosine module description*

The SINE module shown in Figure 7 and the COSINE module shown in Figure 8 calculate sine and cosine functions. Both circuits have a strong mathematical relationship, as they have a very similar structural design described in (3).

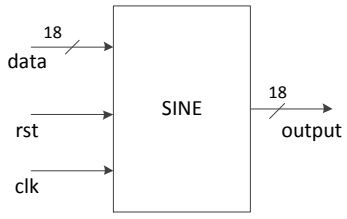

Fig.7. Sine module.

Shown in figure 2 and 3, these modules present an 18-bit format: 1 bit for sing, 9 bits for entire data and 8 bits for fractional data. The output is presented an 18-bit format as well: 1 bit for sing, 1 bit for entire data and 16 bits for fractional data.

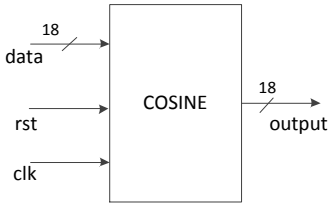

Fig.8. Cosine module.

Figures 9 and 10 show implemented blocks for sine and cosine functions, the most important difference between them is the COMPL ANGULO block, that allows the COSINE module to shift 90 degrees from the values for the SINE module, as mentioned in (3).

The sign signal at output port for the SINE module depends on an input data angle sign. If the input angle is negative, with a two complement format, this angle is complemented in COMPL block in order to get a positive value.

For that reason, if the input angle is negative, the entire and fractional data are complemented in this block.

The SUM block determines the addresses in MEM memory, in which one is saved sine function values for 0 to 90 degrees, and is used to determinate the output value for sine or cosine modules at the output port.

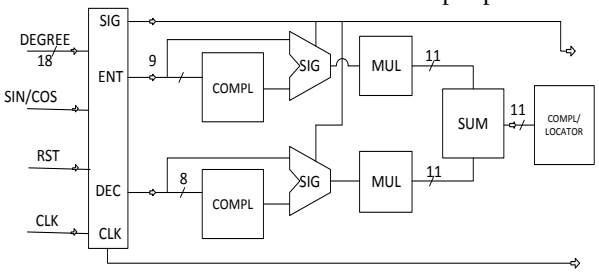

Fig.9. Sine/Cosine module. Part A.

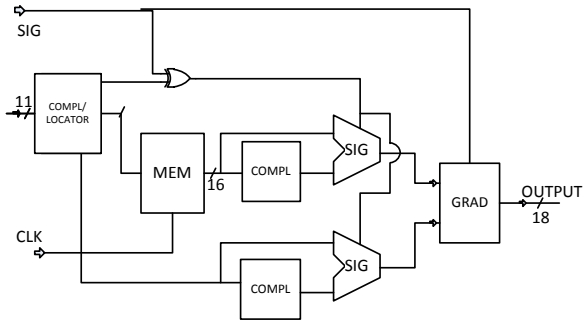

Fig.10. Sine/Cosine module. Part B.

Angle complementation is described as the way to find sine function output values for an entire cycle using only the first quadrant values saved in the embedded memory in the FPGA.

Part of an angle complementation code in a hardware description language VHDL is shown:

```
 --If angle = 90°
    if (grad_x20 = "00011100001000" or
grad_x20 = "10001100101000") then
       complemento <= "00000000000000";

 -- If angle = 270°
    elsif (grad_x20 = "01010100011000" or
grad_x20= "10101000110000") then
       complemento <= "00000000000000";

 -- If angle <  90°
    elsif (grad_x20 < "0011100001000")
then
       complemento <= grad_x20;

 -- If angle > 90°
    elsif (grad_x20 > "00011100001000"
and grad_x20 < "00111000010000") then
 complemento      <=      "00111000010000"-
grad_x20;

 -- If angle >= 180°
     elsif (grad_x20 >= "00111000010000"
and grad_x20 < "01010100011000") then
     complemento     <=     grad_x20     -
"00111000010000";

 -- If angle  > 270°  and < 360°
     elsif (grad_x20 > "01010100011000"
and grad_x20 < "01100000111000") then
     complemento  <=  "01110000100000"  -
grad_x20;

 -- If angle  >= 360°
     elsif (grad_x20 >= "01110000100000")
then
     complemento     <=     grad_x20     -
"01110000100000";

     end if;
 end process;

 signo <= signo_s(1) xor signo_s(0);
 direccion_c  <=  complemento  (entero+1
downto 0);
```

If the angle is equal to 90º, the entire output data is active in "1" and the value temporal register is in "0", it is the register that addresses memory ROM.

If the angle is equal to 270º, the entire output data is active in "1", the sign bit is in "1" and the value temporal register is in "0". If there are other angles, the entire output data is in "0" and the process continues with angle complementation.

If the angle is less than 90º, the value temporal register addresses memory ROM and the sign output data is in "0". In the second quadrant, if the angle is greater than 90º and less than 180º, the value temporal register is 180º minus the angle value sign of the output data is in "0".

If the angle is greater than 180º and less than 270º, the value temporal register is angle value minus 180º, and the sign output data is in "1". It is the third quadrant.

If the angle is greater than 270º and less than 360º, the value temporal register is 360º minus angle value, and the sign output data is in "1". It is the fourth quadrant.

If the angle is equal or greater than 360º, the value temporal register is angle value minus 360º, the sign output data depends on the value of the register. It is the first quadrant again.

At the end, the output bit sign receives a value that depends on the original bit sign and quadrants.

Figure 11 describes a Flow Chart of the angle complementation block.

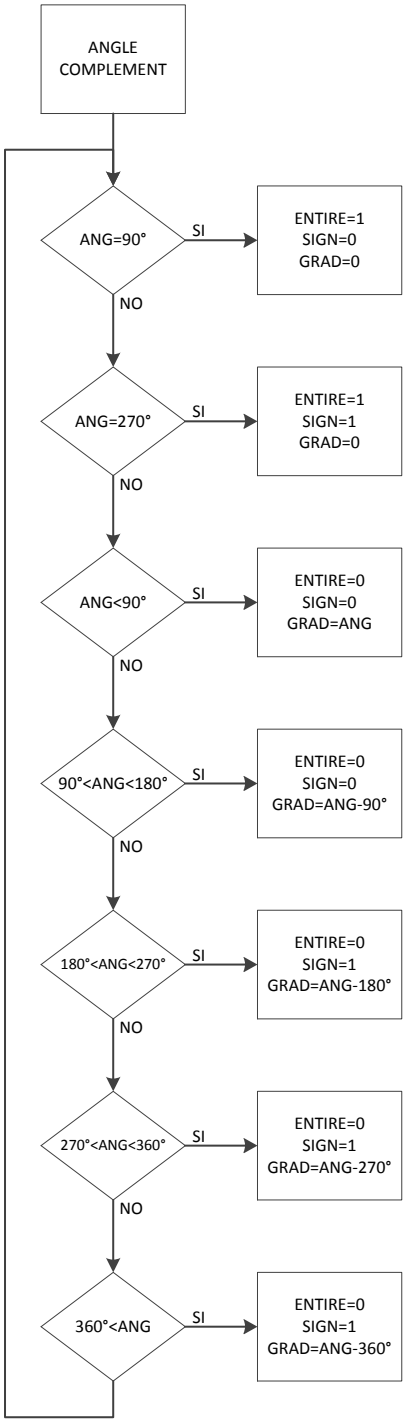

Fig.11. Flow chart angle complementation

V. CUMULATIVE ERROR

An application of trigonometric modules is shown in the implementation of the equation systems (3) and (4). Both equations use the implemented trigonometric functions and are mutually dependent. The calculated data will be feedback to the system. The original error

increases while it is multiplied across a number of cycles. This is known as a cumulative error [5]. This error will be contrasted with the error from a computer numerical calculation system, in this case the MATLAB mathematical computing software.

$$z_1(k+1) = cos(a \cdot t_0) z_1(k) + sin(a \cdot t_0) z_2(k) \quad (4)$$

$$z_2(k+1) = -sin(a \cdot t_0) z_1(k) + cos(a \cdot t_0) z_2(k) \quad (5)$$

A. *Implementing Equations (4) and (5)*

To successfully implement the above equation systems, we developed a hardware array to calculate these functions [10].

The elements of the functions (4) and (5) are separated into sub-functions, which are implemented individually.

$$\lambda = \begin{bmatrix} \alpha(k+1) \\ \beta(k+1) \end{bmatrix} = \begin{bmatrix} \cos(\gamma) & \sin(\gamma) \\ -\sin(\gamma) & \cos(\gamma) \end{bmatrix} \bullet \begin{bmatrix} \alpha(k) \\ \beta(k) \end{bmatrix} \quad (6)$$

Where: $\gamma = a \cdot t_0$

Seen as a hierarchical arrangement of arithmetic modules, the functions (6) can be seen as the block diagram of Figure 12.

1) DELTA Module: the DELTA module is developed as a set of three sub-modules: **GAMMA**, **ALPHA** and **BETA**. The results of calculations made by ALPHA and BETA are sent out to two records, RZ1 and RZ2.

2) GAMMA Module: The GAMMA module receives the entry values from the input buses a (10 bits) and t_0 (4 bits), delivering the result of the multiplication in a 14-bit format, 1 sign bit, 3 bits for the entire number and 10 bits for the fractional number.

3) ALPHA and BETA Modules: ALPHA and BETA modules calculate values for the z_1 and z_2 outputs respectively.

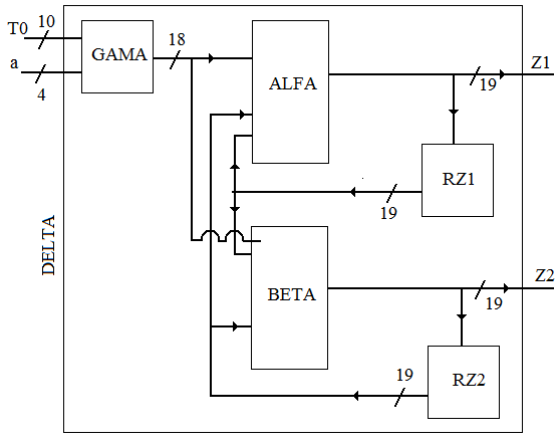

Fig.12. Delta Module Overview.

B. *Results*

The result shows how the initial error increases, as it gets feedback into the system with each iteration.

Figure 13 shows in thick lines how the graph of the device's response separates from the Matlab's response graphs in fine line (ideal).

After 1000 iterations, the plotted functions provided by the developed system, increasingly move away from the optimal results.

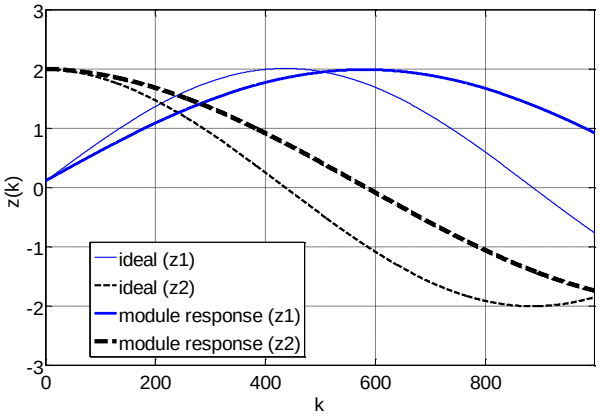

Fig. 13. Accumulated error.

VI. STEADY STATE ERROR

The test of steady-state error to implement is to take an initial value of sine / cosine, multiply it and feed it back into the system, monitoring its response. Unlike the previous test, here a new trigonometric function of the task is calculated in each cycle. The goal is to start with a given value, from which the system reaches a "stable" condition, i.e. to obtain a value of arc ***t*** which provides a result which when multiplied by a constant α, results again in the value of arc ***t***, and when this state is reached, we measure the error in the output value [6].

The starting value is $\sin(10^o)$, which, after a defined number of iterations multiplied by a value of 128, becomes stable. The value α =128 was chosen because it provides a "multiplication" based on a shift register, reducing the error.

Thus, the equation that determines the operation of this system is given by:

$$z(k+1) = sin(128 \cdot z(k)) \quad (7)$$

Where $k \in R$, $k = 1,2,3,4 \ldots n$

A. *Results*

The resulting error in steady state for the trigonometric module is 5.58% which is much larger than the error in the ideal system.

Figure 14 shows the response from the ideal system (fine) versus the implemented system (dash).

Table I shows the number of iterations needed to reach the rise time Tr, the time required for z(k) signal to change to a specified value, and settling time Ts, is the

time elapsed at which the module output z(k) has entered and remained within a specified error band.

In the implemented system, Tr takes 23 elapsed cycles (k) and Ts is 126 k-cycles. For the ideal system, Tr takes 4 elapsed cycles (k) and Ts is 141 k-cycles

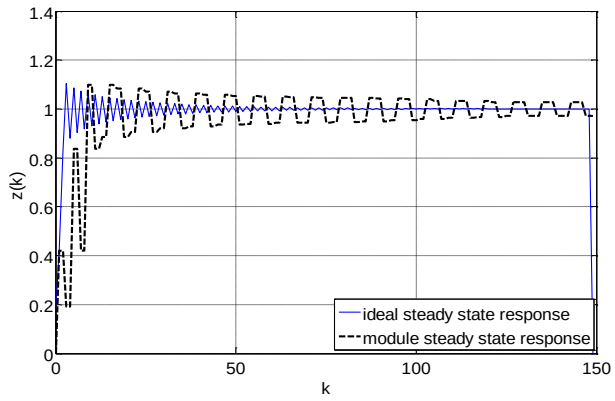

Fig.14. Steady State Error

Even if the accumulated error increases as the system is fed back, pulling the system from the ideal response function, the computation time for the functions is much smaller in the FPGA (on the order of 30ns in the worst case) than in the numerical computation software Matlab (18.865 s in the best case observed), in an i5M480 2.67Ghz processor with 6GB memory computer system.

TABLE I
RESULTS FOR STEADY TIME ANALYSIS

Parameter	**Elapsed cycles**	
	Matlab	**FPGA (Virtex 4)**
Rise time (Tr)	4	23
Settling time (Ts)	141	126

VII. CONCLUSIONS

In the present work, modules developed for trigonometric functions implemented present time execution advantages building a matrix-value, also an efficient utilization in reconfigurable FPGA devices. Considering these module's characteristics, we could use them to solve some algorithms that involve trigonometric sine and cosine functions in real time.

ACKNOWLEDGMENT

We thank the economic support program PROMEP of the Secretariat of Public Education of México (SEP).

REFERENCES

[1] H. Ajorloo, H. Ebrahimi. *"Optimizing Pipelines of Trigonometric Functions for FPGAs". IEEE* 1-4244-1190-4 Jul. 2007.

[2] D. E. Joyce. (1997). *Trigonometric Functions: Arbitrary angles and the unit circle*. [Online]. Available: http://www.clarku.edu/~djoyce/trig/ functions.html

[3] R. N. Auffman, V. C. Barker. *Collage Algebra and Trigonometry*, 7th ed., Ed. USA: Houghton Mifflin Company, 2008.

[4] (2010), *Xilinx In-Depth Tutorial*, Xilinx Home Page. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_1/ise_tutorial_ug695.pdf

[5] C. Pozrikidis. *Fluid Dynamics: Theory, Computation, and Numerical Simulation*. Springer, 2009.

[6] K. Ogata. *Discrete Time Control Systems*, 2nd ed., Ed. México: Prentice Hall.

[7] Amruta, G.; Yogita, P.; Puja, P.; Sriniwas Shastry, P.V.; , "Low latency and high accuracy archtectures of cordic algorithm for cosine calculation on FPGA," *Electronics, Circuits and Systems, 2008. ICECS 2008. 15th IEEE International Conference on* , vol., no., pp.478-481, Aug, 2008.

[8] Ghosh, A.; Paul, S.; Bhunia, S.; , *"Energy-Efficient Application Mapping in FPGA through Computation in Embedded Memory Blocks,"* VLSI Design (VLSID), 2012 25th International Conference on , vol., no., pp.424-429, 7-11 Jan. 2012.

[9] Xie Baozhong; Chen Tiequn; , *"Sine wave algorithm based on 2nd offset and its implementation in FPGA,"* Electronic Measurement & Instruments (ICEMI), 2011 10th International Conference on , vol.3, no., pp.173-176, 16-19 Aug. 2011.

[10] J.-M. Muller, *Elementary Functions, Algorithms and Implementation*, 2nd ed. Birkh¨auser, 2006.

[11] Bhuria, S.; Muralidhar, P.; , "FPGA implementation of sine and cosine value generators using Cordic Algorithm for Satellite Attitude Determination and calculators," *Power, Control and Embedded Systems (ICPCES), 2010 International Conference on* , vol., no., pp.1-5, Dec., 2010.