International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 3 Issue 9, September- 2014

# A VLSI Implementation of Floating Point CORDIC Coprocessor

Roshny G. Kumar
Assistant Professor
Department of Electronics and Communication
Mar Baselios College of Engineering and Technology
Thiruvananthapuram, Kerala, India

K. Padmakumar
Sci/Eng 'SF'
Flight Computer Division, VSSC, ISRO
Thiruvananthapuram, Kerala, India

***Abstract*— A floating point CORDIC coprocessor is designed which accepts inputs in the IEEE 754 double precision format. These are internally converted into 128 bit fixed point representation by a preprocessor. The CORDIC module operates in either the rotation or vectoring modes for circular, linear or hyperbolic systems to produce the result. The results after completion of the CORDIC iterations are reconverted into the IEEE 754 format by a postprocessor. Improved precision is obtained by the use of 128 bits during CORDIC iterations. This project has been implemented in Quartus II.**

***Keywords*— CORDIC, coprocessor, circular, linear, hyperbolic, IEEE 754**

## I. INTRODUCTION

Microprocessors are fast for number-crunching but when it comes to computing trigonometric functions for navigation systems conventional architectures are not fast enough. Jack E. Volder introduced COordinate Rotation DIgital Computer or CORDIC algorithm in 1959 in order to compute these functions. It was later on extended by John Walther to include hyperbolic and transcendental functions.

The CORDIC algorithm is a simple and efficient algorithm designed to meet these requirements with simple shift and addition operations. CORDIC algorithm has been adapted into a variety of applications like the 8087 math coprocessor, the HP-35 calculator, radar signal processors and robotics. It is also useful in computing Discrete Fourier, Discrete Cosine, Singular Value Decomposition, matrix inversion and solving linear systems.

This paper implements a floating point CORDIC coprocessor that can be used for the computation of trigonometric, linear, hyperbolic functions and other transcendental functions like square roots and exponentiations. The inputs to the system are 64 bit and are in IEEE 754 double precision floating point format. It is internally converted to 128 bit fixed point format. The result is reconverted into 64 bit floating point format after the CORDIC iterations.

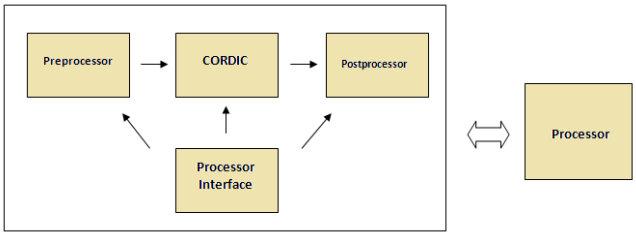


Fig. 1. Design Architecture

Fig. 2.

## II. DESIGN ARCHITECTURE

The architecture for the coprocessor is as shown in Fig. 1. It consists of a preprocessor, the CORDIC, postprocessor and processor interface modules. The design is interfaced with a DSP processor.

### *A. Preprocessor*

The preprocessor converts the floating point inputs from the processor into fixed point representation. The inputs follow the IEEE 754 double precision standardized format. This 64 bit representation is internally converted into 128 bit fixed representation to counter the loss of precision that occurs as a result of CORDIC iterations. This module also generates a 6 bit operation code or opcode based on the input from the processor to control the operation of the CORDIC module.

### *B. CORDIC*

CORDIC algorithm is based on ancient geometric principles wherein all trigonometric functions can be computed through a series of vector rotations. The CORDIC rotator has two modes of operation: the rotation mode and the vectoring mode. In the rotation mode, the vector is rotated through a specified angle. The angle accumulator is initialized with the required angle and vector is rotated through fixed angles until the desired angle is obtained. In the vectoring mode, the input vector is rotated through whatever angle is necessary to align the result vector with the x axis [1].

The CORDIC algorithm was initially developed for the circular coordinate system by Jack Volder [2]. Later on it was extended to linear and hyperbolic systems [3]. The generalized CORDIC algorithm can be written as:

$$x_{i+1} = x_i - \mu d_i y_i 2^{-i} \quad (1)$$
$$y_{i+1} = y_i + d_i x_i 2^{-i} \quad (2)$$
$$z_{i+1} = z_i - d_i e_i \quad (3)$$

where $\mu=1$ and $e_i=\tan^{-1}(2^{-i})$ for circular system, $\mu=0$ and $e_i=2^{-i}$ for linear system and $\mu= -1$ and $e_i=\tanh^{-1}(2^{-i})$ for hyperbolic system [4].

In this design, the CORDIC module operates on the 128 bit inputs from the preprocessor. The opcode generated by the preprocessor determines whether the CORDIC operates in circular, linear or hyperbolic modes and activates it accordingly. It also determines whether the rotation or vectoring mode is used.

Normally serial shifters are used for shifting purposes. This requires more than one clock cycle to complete a single iteration of the CORDIC. Here barrel shifters are used which ensure that an iteration requires only one clock cycle.

This module implements 53 iterations of the CORDIC equations. Circular, linear or hyperbolic iterations are chosen according to the opcode and the corresponding lookup tables are selected. Some functions like tan and tanh require compound iterations. In the case of tan, iterations in circular rotation mode are followed by iterations in linear vectoring mode. State machines are used for control of the CORDIC module. Fig. 2 shows the architecture followed for the CORDIC module.

### C. *Postprocessor*

The postprocessor converts the results from the CORDIC into 64 bit floating point format. The result obtained after CORDIC iterations is 128 bits long. It is normalized and rounded to the standardized format. Here rounding to the nearest even scheme is used. The results are reconverted into the sign, exponent, mantissa format as described by the IEEE 754 double precision format for floating point numbers [5], [6].

### D. *Processor Interface*

The processor interface module acts as an interface between the coprocessor and the processor. It also acts as a control for the whole system. This module receives the data inputs from the processor. Each of the data inputs are 64 bits. The interface module also receives a 4 bit command from the processor which specifies the function required to be computed by the CORDIC. It operates by means of state transitions.

## III. VERIFICATION METHODOLOGY

The verification of the RTL design is done using the golden model method. It uses three reference models for validating the design: 1) The golden model 2) The algorithm reference model and 3) The hardware equivalent model in software.

### A. *The Golden Model*

The golden model serves as the standard for comparing the results of all the other models. It is implemented using the standard built-in functions in C like sin, cos etc. All results obtained are validated against the golden model.

### B. *The Algorithm Reference Model*

In this model the CORDIC algorithm has been implemented for decimal numbers. This model serves as a study of the algorithm. It was developed in different stages. First, the algorithm was implemented for circular system in the rotation mode. This was followed by the implementation of the circular system in vectoring mode. This was followed by the implementation of the algorithm for linear and hyperbolic systems in a similar manner. Double iterations of the hyperbolic system were used to achieve convergence.

### C. *The Hardware Equivalent Model*

This model works on binary numbers. The GCC tool was used to implement this. The model used the pre and post processors for conversion of floating point numbers in decimal to fixed numbers in binary and vice versa.

All the above models were implemented in C.

## IV. SIMULATION RESULTS

Fig. 3 shows the simulation result for the integrated module. Initially, the processor interface accepts the inputs from the processor. Once all the inputs are available, the other modules are activated one after the other at the right times. In the figure, the data inputs provided are **data1=3fe0c15237db38a0** and **data2=0000000000000000** and **cmd=0000** for sine and cosine computation. The final results are **done=1**, **result= 3febb67af83461c2** for cosine and **result2= 3fe0000008a68cc2** for sine.

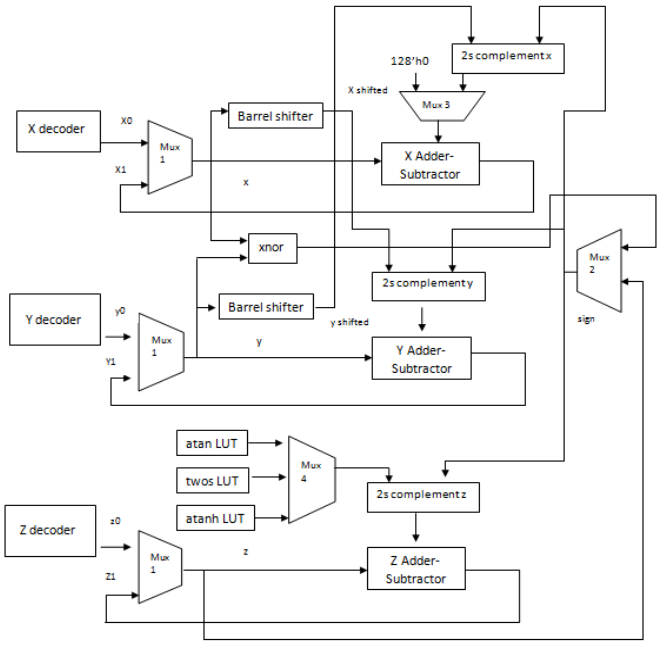


Fig. 3. CORDIC Architecture

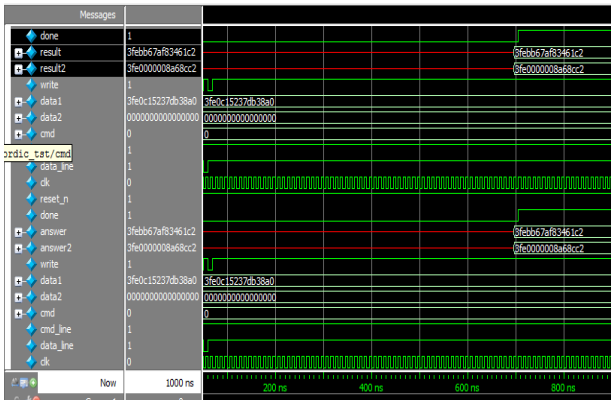


Fig. 4. Simulation Result

## V. CONCCLUSION

A CORDIC coprocessor has been designed which accepts 64 bit floating point inputs. These inputs are internally converted into 128 bit fixed point representation. The CORDIC iterations are performed in circular, linear or hyperbolic case for rotation or vectoring modes based on the requirement and the results are reconverted to floating point format. The RTL diagram of the design is shown in Fig. 4.

Currently the design works for inputs between $-90^0$ to $+90^0$. As a future work, it is planned to extend this range so that it works for any angle. Redundant arithmetic can also be used for CORDIC implementation. This can help to improve the range further. Normally radix 2 is used. Implementation using higher radices can also be attempted.

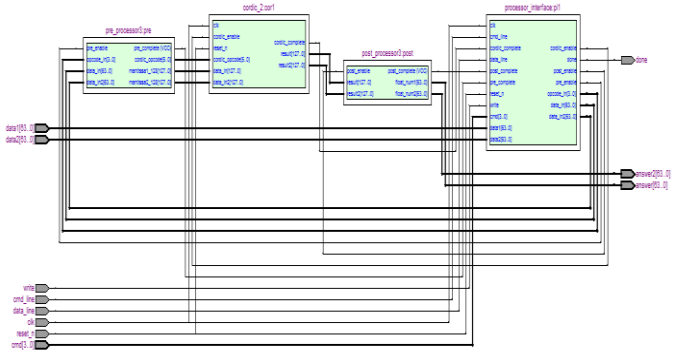

Fig. 5. RTL diagram

## REFERENCES

[1] Ray Andraka, "A Survey of CORDIC Algorithms for FPGA based Computers", Andraka Consulting Group, Inc, North Kingstown, RI02852, 2011.

[2] J.E. Volder. "The CORDIC Trigonometric Computing Technique", IRE Transactions on Electronic Computing, vol EC-8, pp 330-334, Sept 1959.

[3] J.S. Walther, "A Unified Algorithm for Elementary Functions", Proc. Spring Joint Computers Conference, pp. 379-385, 1971.

[4] Behrooz Parhami, Chapter 22, "Computer Arithmetic: Algorithms and Hardware Designs", 2nd edition, Oxford University Press, New York, 2010

[5] Israel Koren, Chapter 4, "Computer Arithmetic Algorithms" ,2nd edition, A.K. Peters, Natick, MA, 2002

[6] W. Kahan, "Lecture notes on the Status of IEEE Standard 754 for Binary Floating Point Arithmetic", October 1, 1997