



CORDIC Algorithm Implemented as a Virtual Component

Grzegorz Potok, Włodzimierz Wrona
Technical University of Lodz
Department of Electrical Engineering
ul. Willowa 2, 43-300 Bielsko-Biala, Poland
potok@aristo.pb.bielsko.pl

Adam Bitniok, Wojciech Sakowski
Evatronix S.A.,
ul. 1 Maja 8, 43-300 Bielsko-Biala, Poland
ipcenter@evatronix.com.pl

ABSTRACT

The paper presents the CORDIC Algorithm, which has been implemented as an virtual component (IP core) in a VHDL simulation environment. The core is packaged as a soft (VHDL) macro and it implements all transcendental functions. Analysis of the accuracy of the algorithms implemented shows that the CORDIC functions are equivalent to the accuracy of a Pentium coprocessor.

KEY WORDS

Simulation, IP core, CORDIC functions

1. Introduction

The CORDIC algorithm was first described by Volder [1] as a solution for easy hardware implementation of trigonometric functions. Walther [2] extended this concept to other transcendental functions. There are numerous concise descriptions of different flavors of CORDIC algorithm in later publications (e.g. [5]). Authors of these papers discuss iterative equations that enable computations of trigonometric, hyperbolic, inverse trigonometric, inverse hyperbolic, linear functions and vector magnitude.

CORDIC algorithms were used in HP35 pocket calculators [3]. Recently examples of FPGA-based implementation of these algorithms were published [4]. Optimizations of the algorithm performance at the expense of silicon area were described in [5]. The CORDIC algorithm is used for implementation of the following functions: $\sin(x)$, $\cos(x)$, $\text{tg}(x)$, $\text{arctg}(x/y)$, $2^x - 1$, $\text{ylog}_2(x)$, $\text{ylog}_2(x+1)$.

Hardware implementation of this algorithm is based on datapath with adders, shifters, registers and Read Only Memory (containing a number of pre-computed constant factors). Controller enables realization of all transcendental functions with the same datapath structure, but with different sequence of basic operations.

2. Virtual Components

Over last three decades integrated circuits of ever growing complexity were used as components of electronic systems that were assembled on the printed circuit boards. Now such systems may be integrated into a single silicon die. This dramatic increase of complexity of integrated circuits, that can be manufactured today with state-of-the-art semiconductor technology, challenges the designer productivity. The only way to take advantage of what technology offers is to reuse existing designs of functional blocks in order to create complex systems.

The data format for description of such reusable designs may be different, but they share a common characteristic: from a point of view of their users they are *components*. But as they are just sets of design data rather than actual pieces of hardware they are referred to as *virtual components*.

One of the possible forms of virtual components are functional descriptions of hardware blocks in hardware description languages. These descriptions are *synthesizable*, which means that an appropriate design software tool may automatically synthesize logical circuit structures that implement the function described.

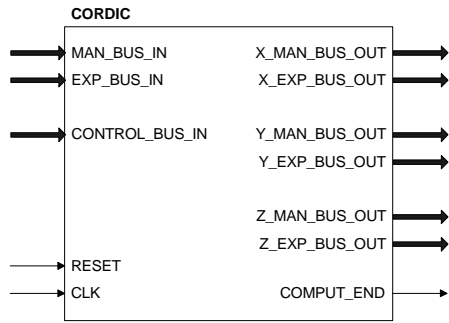
3. CORDIC Module

The CORDIC module we described here was developed as a part of a numeric coprocessor compliant to the Intel 80C187 chip. It is separated from the main arithmetic unit of the coprocessor. It implements floating point operations on numbers formatted according to the IEEE STD 754. Arguments may be normalized and denormalized numbers. Mantissa and exponent of the result are calculated during computations.

Fig. 1 illustrates the interface of the CORDIC module. Mantissa and exponent of the arguments are written to the input registers of the module. The control bus determines the function to be computed and the mode of computation. The signal COMPUT_END is raised once the computation is over. The results (mantissas and exponents) are at that time available in the output registers (X, Y or Z).



Arguments of the transcendental functions are formatted as extended precision floating point numbers (80-bit wide). But computations in the CORDIC module are realized in the datapath that is widened with extra bits. Such approach is necessary to avoid degradation of accuracy and to keep the error of the result below acceptable values [8].



MAN_BUS_IN	Argument mantissa
EXP_BUS_IN	Argument exponent
CONTROL_BUS_IN	Control signals
CLK	System clock
RESET	System reset
X_MAN_BUS_OUT	Result(s) mantissas
Y_MAN_BUS_OUT	
Z_MAN_BUS_OUT	
X_EXP_BUS_OUT	Result(s) exponents
Y_EXP_BUS_OUT	
Z_EXP_BUS_OUT	

Figure 1. Interface of the CORDIC module.

The CORDIC module operates in either rotation or vectoring mode. Trigonometric functions as well as exponential function are computed in the rotation mode. Vectoring mode is used to compute logarithmic and arctangent functions.

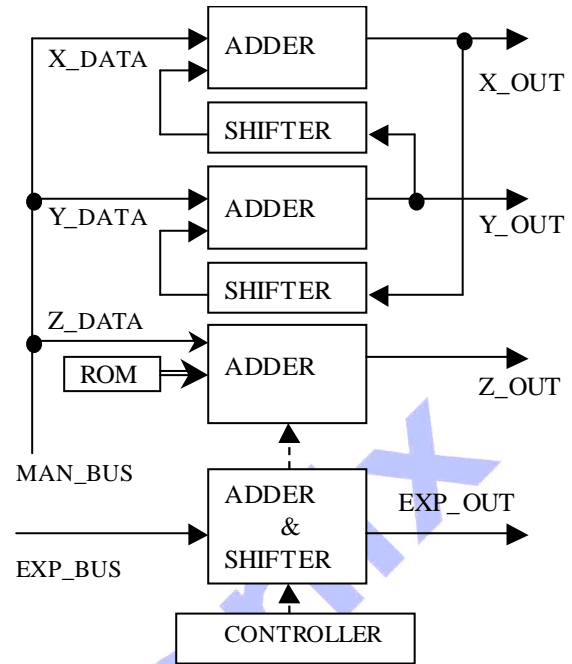
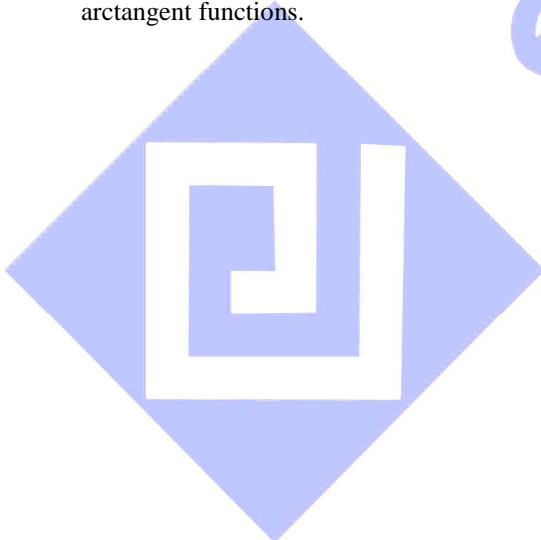


Figure 2. Architecture for CORDIC implementation

The CORDIC algorithm is implemented in VHDL as a design entity, which is composed of several components. It uses only 3 adders for mantissa and exponent manipulation, 2 shift registers for mantissa, coefficient ROM and controller. A simplified architecture for the CORDIC implementation is shown in Figure 2.

The controller ensures that input data are converted to the proper format, that the output value is then computed and that the result gets normalized at the end. During the "conditioning" of the input argument its mantissa is initially shifted. The main computation algorithms consist of a series of shift operations, and additions or subtractions. At each clock cycle the controller determines by how many bits the shift operation should execute and what operation (addition or subtraction) should take place.

The maximum number of clock cycles needed to complete computation of different functions by the the CORDIC module is shown in the Table 1.





run the same set of tests with a low-level structural model.

Function	Max number of clock cycles
$\sin(x)$	439
$\cos(x)$	439
$\text{tg}(x)$	379
$\text{arctg}(x/y)$	343
$2^x - 1$	417
$y \log_2(x)$	412
$y \log_2(x+1)$	412

Table 1. The maximum number of clock cycles needed to compute the transcendental functions

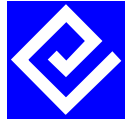
4. The simulation

The CORDIC module discussed in this article was developed as a part of the virtual component that is functionally equivalent to the Intel 80C187 numeric coprocessor. Testing of this 80187-compliant virtual component comprised of functional testing of its I/O behavior, verification of proper handling of all corner cases and accuracy of all computational algorithm implemented (the full FP instruction set). In order to achieve this 3344 tests were written. They were divided into groups related to the coprocessor instructions: simple arithmetic operations, comparisons, data transfer, control instructions of several categories including rounding modes and conversion between different precision formats. Special tests were developed to check for non-maskable exception handling and different addressing modes. Tests of instructions that execute transcendental functions enabled us to test the CORDIC module.

All tests were run first with the original 80C187 chip. This was done by means of a hardware modeler interfaced to the CADAT simulator (once manufactured by Racal-Redac). This framework enabled us to gather reference results (produced by the real chip). They were recorded by the CADAT simulator into text files.

An exact copy of this testing environment was implemented as a VHDL testbench in which our VHDL model of the coprocessor was simulated. The text files with reference results were used to spot differences between the original chip and our model. It took 24 hours to simulate the execution of all tests with the RTL (register transfer level) functional model.

Once the FPGA prototype was designed, a post-layout simulation could take place. It allowed us to verify that the synthesis process and place and route of the FPGA circuit produced a correct logic structure of the coprocessor. It took three computers and three weeks to



The software used for simulation was Active-HDL (ver. 3.6) from Aldec company. Computers were Athlon-based machines with clock rates of 700 MHz to 1 GHz.

5. Accuracy analysis

Accuracy of the results produced by CORDIC module for each transcendental function was investigated with two testing methods. The first was simulating the RTL model of the CORDIC module with Active-HDL. The simulated results were compared to the results obtained with the hardware model of the original 80C187 chip. This hardware model was tested in a hardware modeler and the CADAT simulator.

Several hundred of the tests showed that the results produced by our core were identical to the 80C187chip or that there was a difference of mantissas that equaled 0000 0000 0000 0001 (hex decimally). In the second phase of testing the results of our core were compared with the numeric coprocessors of newer generations: 486DX4, K6, Celeron & Pentium III.

Two example results of computations obtained with different processors (mentioned above) and with our core for FSIN function are shown in Table 2.

The second method of verification is based on the comparison of the result generated with C80187 core and the exact value. In order to be able to perform the tests in a reasonable time for a large amount of data, and to automate the analysis process we developed a testing board that communicates with a PC. The board contains an Intel 80C186 microcontroller and an 80C187 coprocessor or an equivalent FPGA prototype of our C80187 core.

Argument 1 ($\pi/16$)		3FFC C90F DAA2 2168 C234
Results	C80187	3FFC C7C5 C1E3 4D30 55B 2
	80C187	3FFC C7C5 C1E3 4D30 55B 1
	486DX4	3FFC C7C5 C1E3 4D30 55B 1
	K6	3FFC C7C5 C1E3 4D30 55B 2
	Celeron	3FFC C7C5 C1E3 4D30 55B 2
	Pentium III	3FFC C7C5 C1E3 4D30 55B 2
Argument 2 ($\pi/8$)		3FFD C90F DAA2 2168 C234
Results	C80187	3FFD C3EF 1535 754B 168 C
	80C187	3FFD C3EF 1535 754B 168 C
	486DX4	3FFD C3EF 1535 754B 168 C
	K6	3FFD C3EF 1535 754B 168 D
	Celeron	3FFD C3EF 1535 754B 168 C
	Pentium	3FFD C3EF 1535 754B 168 C

Table 2. The comparison of FSIN results for different coprocessors.



The test program is uploaded from the PC to the testing board. The program executes a given transcendental function for a set of input arguments. These arguments were obtained by means of a generator of pseudorandom numbers. Each set of arguments contains one thousand numbers. The results are written to the memory of the testing board and once the testing program is finished they are transmitted to the PC.

The accuracy of numeric algorithms may be expressed in *ulp* units (units in the last place) [7]. Let us assume that $f(x)$ is an accurate value for argument x . And the computed value is denoted $f_{comput}(x)$. Then the error expressed in *ulp*-s is defined by the equation:

$$e_{ulp} = \left| \frac{f(x) - f_{comput}(x)}{2^{k-63}} \right|$$

where k is an integer number that satisfies the condition:

$$1 \leq 2^{-k} f(x) < 2$$

Reference values were calculated by means of a math program DERIVE. This program allowed us also to calculate the maximum error for each transcendental function. For all executed tests the computational error was kept below 1 *ulp*.

6. Conclusions

Despite the vast literature found about the CORDIC algorithms, a lot of detailed problems (rarely discussed in the papers had to be solved) to achieve a viable implementation of the algorithm.

As it was previously mentioned the CORDIC algorithm is based of shifting and adding operations. An important problem in its implementation is balancing the cost and achievable performance of barrel shifters. For each subsequent iteration the shift of the arguments supplied to the adder grows by one. At the 50th iteration this means a shift by 50 bits. Careful design of the controller allowed to minimize the size of the shifters used in the design.

The other issue to overcome was the fact that the shifting of the arguments by many bits lead to a loss of accuracy. A careful sizing of the registers that hold intermediate results enabled us to avoid accumulation of errors.

A proper choice of "magic" constants used during computations of transcendental functions proved also crucial for meeting the accuracy requirements.

The CORDIC module is implemented as a separate virtual component (soft macro). Several versions will be available for different precision formats. These macros may be used as peripheral units for microcontrollers to secure execution of floating point operations at speed unachievable with software implementation.

AUTHORS

G. Potok and W. Wrona are with the Department of Electrical Engineering Technical University of Lodz, ul. Willowa 2, 43-300 Bielsko-Biala, Poland

W. Sakowski and Adam Bitniok are with Evatronix Co., ul. 1 Maja 8, 43-300 Bielsko-Biala, Poland

E-mail:

potok@aristo.pb.bielsko.pl,
wrona@evatronix.com.pl,
ipcenter@evatronix.com.pl

The authors wish to thank CELERITOUS Co. for help while verifying the CORDIC implementation in Xilinx Virtex XCV800 device.

REFERENCES

- [1] J. Volder, *The CORDIC computing technique*, IRE Trans. Computers, 1959.
- [2] J. Walther, *A unified algorithm for elementary functions*, Joint Computer Conference Proc., v. 38, 1971.
- [3] D. Cochran, *Algorithms and accuracy in the HP35*, Hewlett-Packard J., June 1972.
- [4] R. Andraka, *A survey of CORDIC algorithms for FPGA based computers*, ACM, Monterey, 1998.
- [5] J. Duprat, J.M. Muller, *The CORDIC algorithm: new results for fast VLSI implementation*, IEEE Trans. Computers, v. 42, 1993.
- [6] *80C187 80-bit Math Coprocessor*, Intel Corporation, 1992.
- [7] *Intel Architecture Software Developer's Manual*, v. 1, Intel Corporation, 1997.
- [8] D. Goldberg, *Computer arithmetic*. In: Patterson D.A., Hennessy J.L. *Computer Architecture a quantitative approach*, San Mateo, CA: Morgan Kaufman 1990.