A Fast Scheme for Complex Rotation and Its ASIC Implementation

Sergiu Condorovici

"Gh. Asachi" Technical University of Iasi
Faculty of Automatic Control and Computer Engineering
53 Dimitrie Mangeron Bldv.,
6600 Iasi
ROMANIA
e-mail: sergiu@tuiasi.ro

Abstract: This paper describes all steps for high speed complex multiplying chip design. The architectures developed and tested for this chip will be used to build an FFT processor. The described architecture allows very high processing speed. In DSP processor manner, the chip processing speed is approx. 40-50 MIPS.
The used algorithm, CORDIC, is presented in Section 2 of the paper. This algorithm can evaluate the results of parallel multiplying by trigonometrical functions without explicitly computing these functions or without explicitly computing the product. The algorithm can be used for many other types of mathematical functions, but this paper personalizes it for complex rotation. Section 3 of the paper addresses number representation and certain problems of the integer number arithmetic. The whole project was carried out using a behavioural description in VHDL language. Section 4 includes the presentation of the architecture. Experimental results are presented in the last Section of the paper.

Keywords: DSP, CORDIC, VLSI, FPGA, hardware, integer number arithmetic

Sergiu Condorovici was born in 1959. He received the MSc. degree in Electronics from the "Gheorghe Asachi" Technical University of Iasi, Iasi, Romania. He is Assistant Professor at the Automatic Control and Computers Faculty, the "Gheorghe Asachi" University in Iasi. Currently, he completes his studies at the same University. His main interests are digital signal processing, VLSI architectures and real time computing structures.

1. Introduction

The mathematical approach presented in this paper, known as CORDIC - **CO**ordinate **R**otation on a **DI**gital **C**omputer - is an iterative algorithm used to evaluate many mathematical functions, such as trigonometrical functions, hyperbolic functions and planar rotation.

The algorithm introduced in [Vol59] and further developed in [Wal71] has many applications today. The interest in this algorithm has lately revived, especially for its real- time DSP applications.

In the DSP processor area, execution time for medium instruction is computed as a haft of the execution time of the MAC - multiply and accumulate - instruction. The MAC instruction will make, usually in parallel, one multiply operation and one addition operation.

Using the same computing mode for medium instruction, computing of the planar rotation defined by 1 in 1 μs time requires a processing speed of 6 MIPS (two additions and four multiplications) only if the rotation angle θ is constant and trigonometrical functions are precalculated. For a variable rotation angle trigonometrical functions must be computed and this operation is estimated at at least 40 MIPS for sin and cos. All in all the required processing speed is of 46 MIPS.

This high processing speed is not very common. High performance general- purpose microprocessors can provide this computing speed only in conjunction with arithmetic co-processors. For the latest DSP processors, the computing speed is of approx. 50 MIPS (Texas Instruments TMS320C54X has 66 MIPS).

The CORDIC algorithm uses only integer addition/subtraction operations and an integer power of 2 multiplications able to evaluate complicated mathematical functions. This feature makes the algorithm very suitable to run for real time processing on not very elaborate hardware configurations.

Out of all mathematical functions which can be calculated using the CORDIC algorithm , this paper is only concerned with planar rotation.

2. Mathematical Approach

For a vector of (x_i, y_i) coordinate, a planar rotation operation using a matrix form, is defined.

$$\begin{bmatrix} x_f \\ y_f \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{1}$$

or with the following set of linear equations

$$\begin{cases} x_f = x_i \cos\theta - y_i \sin\theta \\ y_f = x_i \sin\theta + y_i \cos\theta \end{cases} \tag{2}$$

We can also use a complex form for (1) if we define the vector modulus and the vector phase.

$$\begin{aligned} R &= \sqrt{x_i^2 + y_i^2} \\ \phi &= \arctan\frac{x_i}{y_i} \end{aligned} \tag{3}$$

The following formula represents a planar rotation in a polar system coordinate.

$$\begin{aligned} x_f + j\,y_f &= e^{j\theta} R\, e^{j\phi} \\ &= (x_i + j\,y_i)\, e^{j\theta} \end{aligned} \tag{4}$$

This form of planar rotation is very similar to one complex multiply operation. One operand may be any complex number, while the other one must be an unitary modulus complex number. This class of complex operations is very common with DSP algorithms.

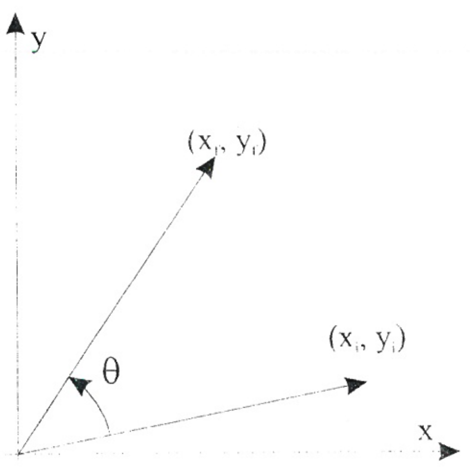

Figure 1. Planar Rotation

Graphically, planar rotation means a transformation of (x_i, y_i) vector into a new one, (x_f, y_f).

The θ angle rotation can be executed in several steps, using an iterative process. Each computing step completes a small part of rotation. The name that will be used for one step is microrotation.

Many microrotations will compose one planar rotation. The angle of a total rotation will be the sum of all elementary microrotation angles.

$$e^{j\theta} = e^{j}\sum_{i=0}^{n}\theta_i \tag{5}$$

One microrotation is defined by the following formula in matrix form:

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} \cos\theta_n & -\sin\theta_n \\ \sin\theta_n & \cos\theta_n \end{bmatrix} \begin{bmatrix} x_n \\ y_n \end{bmatrix} \tag{6}$$

Eq (6) is transformed so as to force to unitary value some matrix elements.

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \frac{1}{\cos\theta_n} \begin{bmatrix} 1 & -\tan\theta_n \\ \tan\theta_n & 1 \end{bmatrix} \begin{bmatrix} x_n \\ y_n \end{bmatrix} \tag{7}$$

The equivalent linear Eqs at (7) need only two multiply operations. Selecting the proper value for a microrotation angle makes it possible to eliminate all multiply operations. To define the matrix elements which differ by 1, the integer power of two, we chose the angle value. Multiplying or dividing by the integer power of two actually means only shift operations.

$$\tan\theta_n = s_n 2^n \tag{8}$$

with

$$s_n \in \{-1; +1\} \tag{9}$$

The new matrix form of the microrotation is

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \frac{1}{\cos\theta_n} \begin{bmatrix} 1 & -s_n 2^{-n} \\ s_n 2^{-n} & 1 \end{bmatrix} \begin{bmatrix} x_n \\ y_n \end{bmatrix} \tag{10}$$

The coefficient preceding the rotation matrix can be transformed as in Eq (11).

$$\frac{1}{\cos\theta_n} = \frac{1}{\sqrt{1+\tan^2\theta_n}} = \\ = \frac{1}{\sqrt{1+(s_n 2^{-n})^2}} = \\ = \frac{1}{\sqrt{1+2^{-2n}}} = C_n \tag{11}$$

It is obvious that computing of C_n coefficient assumes complicated operations, and even if we compute it, two multiply operations must be done. Computing of C_n can be renounced if computing only one global C coefficient, which will include all multiplying operations.

$$C = \prod_{n=1}^{\infty} C_n = \prod_{n=1}^{\infty} \frac{1}{\sqrt{1+2^{-2n}}} \tag{12}$$

The value of C coefficient is constant for all initial vectors or for all values of the rotation angle.

$$C_n = 0.607253 \tag{13}$$

Once computed C value, we can exclude the multiplying by C_n from the microrotation, and redefine the elementary microrotation.

$$\begin{cases} x_{n+1} = x_n - s_n 2^{-n} y_n \\ y_{n+1} = s_n 2^{-n} x_n + y_n \end{cases} \tag{14}$$

It is possible to demonstrate that, for any value of n, the following equation holds:

$$\arctan 2^{-n} \le \sum_{i=n+1}^{\infty} \arctan 2^{-i} \tag{15}$$

In this case for any value of the rotation angle we can find a signs distribution (s_1, s_2, ..., s_n) such as:

$$\lim_{k\to\infty} \sum_{i=1}^{k} s_i \arctan 2^{-i} = \theta \tag{16}$$

Eq (16) suggests that **arctan 2^{-n}** series constitutes a discrete base for the representation of the rotation angle. In this base, all values of the rotation angle can be represented through only +1 or -1 coefficients.

The rotation angle θ will be approximated in the following way.

We compute the difference between θ and the sum of all rotation angles used on microrotations before step n in z_{n+1} variable.

$$z_{n+1} = \theta - \sum_{i=0}^{n} \theta_i \tag{17}$$

At every step this z_n variable represents that part of the rotation angle θ which has not been rotated yet, or the rest of the rotation needed for completing the operation.

For any step the s_n coordinate is computed like a sign of z_n variable:

$$s_n = sign(z_n) = \begin{cases} -1 & if\ z_n < 0 \\ +1 & if\ z_n \ge 0 \end{cases} \tag{18}$$

The computing process of the planar rotation is an iterative one. For this reason Eq (17) must be recalculated in an iterative mode.

$$z_{n+1} = \theta - \sum_{i=0}^{n} \theta_i = \\ = \theta - \sum_{i=0}^{n-1} \theta_i - \theta_n = z_n - \theta_n = \\ = z_n - s_n \arctan 2^{-n} \tag{19}$$

In Figure 2 there is presented an approximation procedure for the planar rotation.

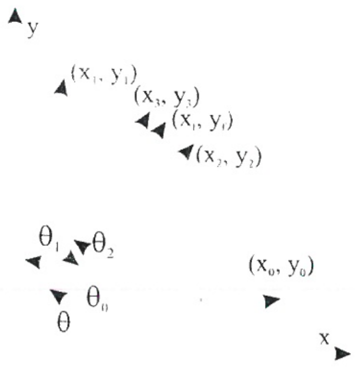

Figure 2. Successive Approximations of

θ Angle

At every step, one elementary θ_n angle is added or subtracted. Following every step the rotation angle taken is closer to θ.

The conclusion of this mathematical approach is that that any planar rotation for any θ angle can be composed of a number of microrotations θ_n.

One microrotation should go through the following computing operations:

- determine the current step sign s_n using Eq (18)
- determine the current step not approximated yet angle z_{n+1} using Eq (19)
- compute the microrotation for the current step using Eq (14).

After completing all the microrotations, it is necessary to multiply them by the C coefficient defined at Eq(13).

3. Specific Considerations About Using the Integer Number Arithmetic

a. Number Representation

We use an 16-bit 2' complement representation for the vector coordinate. With this type of representation, the number's *v* value is expressed by its bits in the following equation

$$v = -2^{15} a_{15} + \sum_{i=0}^{14} a_i 2^i \tag{20}$$

For the rotation angle θ, we also use an 16- bit representation, but otherwise computed.

The function

$$f(x) = \arctan(x) \tag{21}$$

is defined within the DI interval and its values are within the VI interval.

$$DI = (-\infty ; \infty)$$
$$VI = \left[-\frac{\pi}{2} ; \frac{\pi}{2} \right] \tag{22}$$

For this reason the CORDIC algorithm will correctly perform computing only when the rotation angle θ is within the VI interval. This restriction can be removed in the following way.

If

$$\theta \in \left(\frac{\pi}{2} ; \frac{3\pi}{2} \right) \tag{23}$$

then we will compute

$$\theta' = \theta - \pi ; \quad \theta' \in \left[-\frac{\pi}{2} ; \frac{\pi}{2} \right] \tag{24}$$

and the CORDIC algorithm can be used.

$$\begin{aligned} e^{j\theta} &= e^{j(\theta'+\pi)} = e^{j\theta'} e^{j\pi} \\ &= (\cos\pi + j\sin\pi) e^{j\theta'} = \\ &= (-1) e^{j\theta'} \end{aligned} \tag{25}$$

The planar rotation of θ angle is equivalent to the planar rotation of θ' angle followed by sign inversions for the computed coordinate.

Conversion between the radians value of θ angle and the integer value of θ angle is realised by the formula:

$$\theta_{[integer]} = \frac{2^{15}}{\pi} \theta_{[radians]} \tag{26}$$

This conversion mode covers all [-π; π) intervals.

Table I

$\theta_{[integer]}$	$\theta_{[radians]}$
32767	32767/32768 π
16384	π / 2
0	0
-16384	-π / 2
32768	-π (= π)

b. Steps Number

14 equations calculate the current point coordinates by adding or subtracting one value to/from previous point coordinates.

For x_{n+1} the offset is +/- $y_n 2^{-n}$ and for y_{n+1} the offset is +/- $x_n 2^{-n}$.

If a generic number v is represented in an 16-bit two's complement then

$$v\ 2^{16} = 0 \quad for\ any\ v$$
$$v\ 2^{15} = 0 \quad for\ any\ v \neq -2^{15} \qquad (27)$$

After step 16 the computing procedure will stop modify the (x_n, y_n) coordinate point. It is quite possible that step 15 will not modify either.

The most correct way is not to specify the step number. From any state we will be able to execute another step only if the current step modifies the (x_n, y_n) coordinate point.

The conditions

$$x_{n+1} = x_n \wedge y_{n+1} = y_n \qquad (28)$$

are equivalent to

$$2^{-n} y_n = 0 \wedge 2^{-n} x_n = 0 \qquad (29)$$

New microrotations will be executed until the condition at Eq (29) holds.

c. Integer Numbers Shifting

To calculate an **v 2** expression in integers means the n position right shifting of **v**. Free positions on the left of **v** are made 0 if **v** is an unsigned integer, or are replaced by **v** sign if **v** is a signed integer.

In the CORDIC algorithm (x, y) point coordinates are signed integers.

$12345_{(10)} \times 2^{-1} = 0011.0000.0011.1001_{(2)} \times$

$\times 2^{-1} = 0001.1000.0001.1100_{(2)} = 6172_{(10)}$

$-12345_{(10)} \times 2^{-1} = 1100.1111.1100.0111_{(2)} \times$

$\times 2^{-1} = 1110.0111.1110.0011_{(2)} = -6173_{(10)}$

There is a small difference between a positive number dividing and a negative number dividing.

The same computing operation applies to 1 and -1 numbers.

$1 \times 2^{-1} = 1$

$-1 \times 2^{-1} = 1111.1111.1111.1111_{(2)} \times 2^{-1} =$

$= 1111.1111.1111.1111_{(2)} = -1$

With this computing mode, for any negative number **v**

$$if\ |v| < 2^k$$
$$then\ v\ 2^{-k} = -1\ (not\ 0) \qquad (30)$$

This observation must be taken into account when evaluating whether a new step is still necessary.

4. ASIC Implementation

ASIC - Applications Specific Integrated Circuits - is a generic name for a class of technologies used to develop VLSI integrated circuits. Probably, some of the most used and convenient of all these technologies are programmable logic cell based technologies. Also CMOS array based technologies are largely used, but in their case the chip must be produced in millions of items in order to be cost-effective.

In either case, the design of the VLSI chip is possible in two ways.

The first way is to build a logical schemata based on logical components defined by the chip manufacturer. For this, the chip manufacturer will provide component libraries to the chip designer. Chip designer will connect components from libraries to realise VLSI applications.

The second way is to use hardware description languages - HDL - for designing an application. In this case, the VLSI description is a computer software. The source of this software is written with a common text editor, then the source is compiled in order to generate a program module. The program module can be linked to other program modules in the library to generate an executable form of the software. The method which generates this executable form is similar to that generating an executable from any language source code. The differences which might appear are on executing the program. In this program, the execution part is not the processor, but the HDL simulator.

Some languages are used for hardware description. Independent organizations (VHDL and VERILOG) have developed some languages, others have been developed by chip manufacturers (AHDL). The most used language is VHDL - VHSIC Hardware Description Language. In 1987 VHDL became an IEEE standard. In 1993 the IEEE updated the 1987 standard. Today all the applications used to generate a final form of the VLSI chip will admit the VHDL source as input.

Structurally, the VHDL language looks like any parallel programming language, syntactically, it resembles C or PASCAL.

In VHDL there are two kinds of descriptions which may be used: structural description and behavioural description. The structural description assumes that all the components and their connections for one entity are accurately defined. The behavioural description refers to one entity without making any reference to the entity components. It only specifies the reactions of the entity to different external stimuli. VHDL allows any combination of the two types of descriptions. We can give a behavioural description to a functional block and later use its behavioural description in a structural one, where the block is connected with other entities.

If we plan to use only the structural description in VHDL, this can be done without writing any single program line. It is possible to use a schematic capture application to draw the schemata. From the schematic file we can extract the list of connections and the list of components - netlist file - in one common used format like EDIF. The netlist file can be automatically converted to a structural VHDL using a computer software.

For chip designers, the VHDL language is what C is for software designers, that is an universal language. First of all, it guarantees the portability of any application, that is one application can physically use any kind of support, no matter if it is a CMOS area, or an FPGA Xilinx, or an PLD Altera.

To implement the schema for a complex rotation, the behavioural description in VHDL has been used.

The CORDIC entity port is presented in Figure 3. It uses six input signals (input coordinates x and y, the rotation angle z, the reset signal RES, the clock signal CLK and the data load signal LOAD) and three output signals (output coordinates x and y, and the END_CYCLE signal).

```
ENTITY cordic IS
PORT (
    clk, load,
    res: IN STD_ULOGIC;
    x_in, y_in,
    z_in: IN SIGNED (15 downto 0);
    x_out,
    y_out: OUT SIGNED (15 downto 0);
    end_cycle: OUT STD_ULOGIC);
END cordic;
```

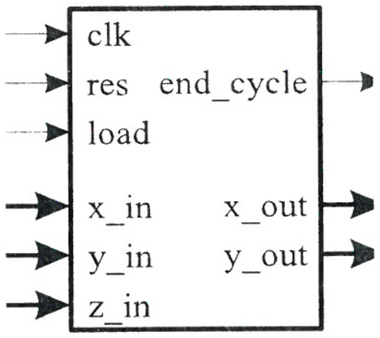

Figure 3. CORDIC Entity

The main elements of the behavioural description are:

- a process for controlling accumulator x and accumulator y operations. This process has a CLK signal and a LOAD signal in the sensitivity list. The accumulators synchronically load the corresponding ALU output (ALU-X or ALU-Y) or the corresponding data input (x_in, y_in)
- a process for controlling accumulator z. This process has a CLK signal and a LOAD signal in the sensitivity list. The functions of the process are similar to those in the previous process. Moreover this process generates the operator control signal for all the three ALUs
- a process for CONTROL BLOCK. This process has CLK, LOAD and RES signals in the sensitivity list, and generates state machine. The RES signal globally initializes the chip. The LOAD signal starts compute the cycle. The states diagram of this process contains 16 states, which are coded from 0 to 15. The 15th state is a wait state. From this state, if LOAD signal gets active, the computing cycle will start. The first state in the computing cycle is 0 state. In a computing cycle, states are sequential.

Transition from n state to n+1 state can take place only if the computed cycle is not over. If the computed cycle is ended, transition will take place in the 15th state. The state machine signal generated in this process is used by all the other elements in the VHDL description

- a set of concurrent assignments which computes the ALU_X value as a function of accumulator x, accumulator y and of machine state. In this operation accumulator y must be shifted by a number of positions determined by machine state. The shift operation must be done statically and not sequentially because of the sequential shift of npositions needs n clock periods. For doing this, we need a big combinational structure. Also, these concurrent assignments do the summing up or the subtraction of the accumulator stored operands and of the shifted operands. The operator control is generated in the accumulator z process
- a set of concurrent assignments which compute the ALU_Y value as a function of accumulator x, accumulator y and of machine state in a way similar to the above
- a set of concurrent assignments which compute the ALU_Z value as a function of accumulator z, and of machine state. Machine state is used as an address for a LUT - look up table - which contains the value of

$$LUT(n) = 2^{15} \arctan 2^{-n} \quad (31)$$

The LUT Table has less than 16 nonzero entries and is arranged as a combinational structure.

Table II. Equivalent processing speed for CORDIC chip

N	Equivalent MIPS
14	26.94
8	53.83

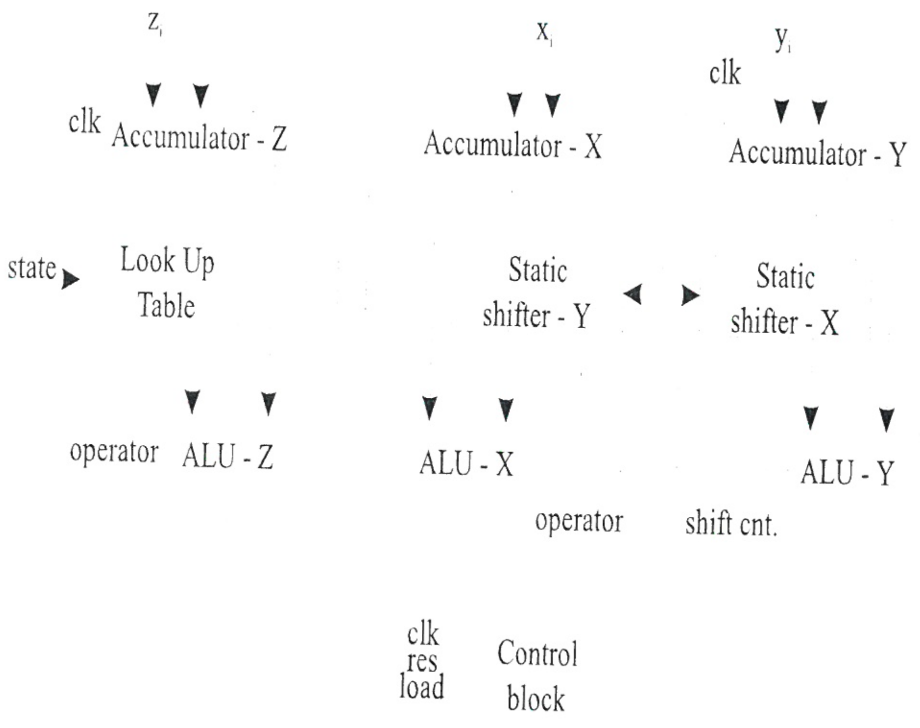

Figure 4. Block Diagram for the CORDIC Chip

Table III

* *DEVICE SUMMARY * *

Chip/ LCs POF	Device	Input Pins	Output Pins	Bidir Pins	LCs	% Utilized
Cordic	EPF8820ATC144 - 2	51	32	1	641	95 %
User Pins:		51	32	1		

Total dedicated input pins used:	4/4	(100%)
Total I/O pins used:	82/108	(75%)
Total logic cells used:	641/672	(95%)
Average fan - in:	3.42/4	(85%)
Total fan - in:	2197/2688	(81%)

Timing Analyzer Report

Max. Clock period:	121.9 ns
Max. Frequency:	8.20 MHz

At this phase of the project global C constant multiplying was not imperious and therefore, it was not implemented. For an application which might imply that, it is possible to perform multiplying by a constant combination, without using many resources.

5. Experimental Results

The main features of the chip are presented in Table III. An ALTERA chip was used. The maximum working frequency which results is 8.20 MHz.

The clock periods number needed to complete a computing cycle is approximately

$$N \approx \log_2(\max(|x_i|,|y_i|)) \quad (32)$$

A full computing cycle time is

$$t_c = \frac{N}{f_{\max}} \quad (33)$$

REFERENCES

1. [Vol59] VOLDER, J., **The CORDIC Computing Technique**, IRE TRANSACTIONS ON COMPUTERS, September, 1959.

2. [Wal71] WALTHER, J., **A Unified Algorithm for Elementary Functions**, Joint Computer Conference Proceedings, 1971.

3. [Lin90] LIN, H.X. and SIPS, H.J., **On-line CORDIC Algorithms**, TRANSACTIONS ON COMPUTERS, 1990.