Iterative Mode Hardware Implementation of CORDIC Algorithm

Raphael A. Camponogara Viera, Paulo César C. de Aguirre, Leonardo Londero de Oliveira and João Baptista Martins
{raphaelviera, paulocomassetto, leonardo}@mail.ufsm.br, batista@inf.ufsm.br

Grupo de Microeletrônica - GMICRO
Universidade Federal de Santa Maria - UFSM

Abstract
This paper presents different hardware implementations of CORDIC (Coordinate Rotation Digital Computer) algorithm in iterative mode. CORDIC is a shift-add algorithm for computing a wide range of functions including trigonometric and logarithmic, for instance. We have used a C model of the CORDIC to validate hardware design functionally. The algorithm was described in VHDL, synthesized and tested with a SPARTAN3E XC5S500E FPGA. After that, they were synthesized using standard cells. We have achieved the maximum frequency of 420 MHz for the 16 iterations approach using the FPGA and 125 MHz in X-FAB XC06 standard cells synthesis.

1. Introduction
The Digital Signal Processing (DSP) has been dominated by low cost microprocessor-based-systems. While these systems offer much flexibility, they are not fast enough for the current DSP applications because yours software algorithms do not meet the demanding [1]. In order to solve this problem, hardware dedication systems can be used for high speed DSP applications.

Trigonometric functions is one of the mainly tasks performed in DSP applications. Among the existing hardware algorithms for trigonometric solutions the CORDIC (Coordinate Rotation Digital Computer) algorithm is one of the most used. Several works and hardware implementations of CORDIC algorithm are presented and the literature, like [2], [3] and [4]. These alternatives exist for implementing "shift-and-add" algorithms functions in FPGA-based systems like logarithm and exponential functions.

This work focuses on a hardware implementation of the Bit-Parallel iterative mode (rotational) CORDIC algorithm with 8, 16 and 32 iterations. To validate this architecture a CORDIC algorithm software implementation was developed. The hardware architectures were implemented in a Spartan3E XC3S500E FPGA and synthesized to X-FAB XC06 standard cells. A comparative study on the performance of these implementations is presented.

The paper is organized as follows. Next we introduce the basis of CORDIC algorithm. Section 3 describes the implementation of the serial mode CORDIC in hardware and software. Section 4 provides simulation and synthesis results. Finally, we conclude the paper in Section 5.

2. The CORDIC Algorithm
The CORDIC algorithm is an iterative method for computing elementary functions like sine, cosine, and arctangent. It was introduced by Volder [5] [6]. The method can also be easily extended to compute square roots, hyperbolic functions as well polar to rectangular and rectangular to polar conversions [7]. It works by reducing the calculation into a number of micro-rotations for which the arctangent value is precomputed and loaded from a table. This method reduces the computation to addition, subtraction, compares and shifts [6]. Those functions are easily performed by a Field Programmable Gate Array (FPGA).

2.1. Basis of CORDIC

The algorithm is derived from the general rotation transform that gives the equations for the vector \((X_{n+1}, Y_{n+1})\) in terms of vector \((X_n, Y_n)\) and \(\theta\) as:

\[
X_{n+1} = X_n \cos(\theta) - Y_n \sin(\theta) \\
Y_{n+1} = X_n \sin(\theta) + Y_n \cos(\theta)
\]

In order to simplify the calculations, the CORDIC algorithm now uses a number of rotations, each one can be easily computed to build up the \(\theta\) angle. We decompose the \(\theta\) angle as a sum of the angles, so that:

\[
\theta = \sum_n \alpha_n
\]

and each \(\alpha_n\) is chosen as: \(\alpha_n = \pm \tan^{-1}(2^{-n}) \equiv \tan(\alpha_n) = 2^n\)
Rearranging the equations for a single rotation in terms of \( \tan(\alpha_n) \):

\[
X_{n+1} = X_n \cos(\alpha_n) - Y_n \sin(\alpha_n) = \cos(\alpha_n)[X_n - Y_n \tan(\alpha_n)] \\
Y_{n+1} = X_n \sin(\alpha_n) + Y_n \cos(\alpha_n) = \cos(\alpha_n)[X_n \tan(\alpha_n) + Y_n]
\]  

(4)

Using \( \tan(\alpha_n) = \pm 2^n \) and let \( \sigma_n = \pm 1 \), the equations for \( X_{n+1} \) and \( Y_{n+1} \) become:

\[
X_{n+1} = \cos(\alpha_n)(X_n - \alpha_n 2^n X_n) \\
Y_{n+1} = \cos(\alpha_n)(Y_n + \alpha_n 2^n Y_n)
\]  

(5)

If we ignore for the moment the \( \cos(\alpha_n) \) terms, then these equations may be implemented with an adder and a shifter. The solution to the term \( \cos(\alpha_n) \) is to ignore it! This in effect means that the values computed by the CORDIC algorithm are \( 1/\cos(\alpha_n) \) greater than they should be at each iteration. The total scaling factor as \( K \rightarrow n \) is:

\[
K = \prod_{k=0}^{n} \cos(\alpha_n) = \prod_{k=0}^{n} \sqrt{1 + \sigma_n^2 2^{-n}} \approx 2^{1.6468}
\]  

(6)

The value of \( K \) depends on the number of iterations and the accuracy of the number in the look-up table. Note that this scaling factor is constant, independent of the angle. In many applications, the scaling factor can be ignored. The angle of rotation is accumulated by summing \( \sigma_n \tan^1(2^n) \). The values of \( \sigma_n \tan^1(2^n) \) are tabulated and then added to an accumulator. CORDIC is normally used in one of two modes: rotation mode and vectoring mode.

**Rotation Mode**: In this mode the initial vector \((X_{in}, Y_{in})\) is rotated by an angle \( \theta \). The resulting values of \( x \) and \( y \) provide the trigonometric functions, and follows the sequence below:

1. An initial vector \((X_{in}, Y_{in})\) is rotated by an angle \( \theta \);
2. The angle accumulator is initialized with the value of \( \theta \);
3. A sequence of rotations where each angle is of magnitude \( \tan^1(2^n) \) is applied;
4. At each step the sign of the angle \( \sigma_n \) is chosen to reduce the angle accumulator;
5. So \( \sigma_n \) is simply chosen according to the value of the angle accumulator (\( Z \) is the angle accumulator).

The equations for rotation mode are:

\[
X_{n+1} = X_n - Y_n \sigma_n 2^n \\
Y_{n+1} = Y_n + X_n \sigma_n 2^n \\
Z_{n+1} = Z_n \sigma_n \tan^1(2^n)
\]  

(7)

where \( \sigma_n = \pm 1 \) if \( Z_n < 0 \), +1 otherwise. At the end of the algorithm, the results are:

\[
X = K[X_{in} \cos(Z_0) - Y_{in} \sin(Z_0)] \\
Y = K[Y_{in} \cos(Z_0) + X_{in} \sin(Z_0)] \\
Z = 0
\]  

(8)

\[
K = \prod_{k=0}^{n} \sqrt{1 + 2^{-2n}}
\]  

(9)

These equations provide both the length of the vector and add the tangent to any initial angle placed in angle accumulator. The sine and cosine functions are performed in the rotation mode. In order to compute the sine or cosine of an angle \( \theta \), the Z register will be initialized with \( 0 \), \( Y \) with \( 0 \), and the X with \( 1 \). The X register corresponds to the scaled cosine value, and the Y register to the sine value. We can determine the unscaled value by dividing the final values by \( K \) (1.6468). For example, after 16 iterations, the algorithm will be as (8) and the results of sine and cosine:

\[
\sin(30^\circ) = \frac{Y_{16}}{K} = \frac{0.8237}{1.6468} = 0.5001 \approx 0.50 \quad \cos(30^\circ) = \frac{X_{16}}{K} = \frac{1.4261}{1.6468} = 0.8660 \approx 0.87
\]

The method to compute polar to cartesian coordinate transformation is the same to compute sine and cosine. The transformation is defined by:

\[
x = r \cos(\theta) \\
y = r \sin(\theta)
\]

To perform this transformation is necessary to load again the \( \theta \) into \( Z \), load \( X \) with \( K \) and \( Y \) with \( 0 \).
3. C Modeling and Hardware Implementations

The iterative (in Bit-Parallel mode) CORDIC algorithm was implemented in C language and VHDL. We have used C modeling to validate the hardware architecture and also find errors that could lead to different results between both implementations [8].

3.1. Software Implementation

The software implementation was developed in C language and provides circular, rotational and hyperbolic functions. The same input vectors for the software algorithm were applied to the hardware input architectures in order to evaluate the functional behaviors.

3.2. Hardware Implementations

The Bit-Parallel CORDIC hardware architecture was described in VHDL and uses a single hardware block to perform the $n$ iterations. This block is composed for three adders/subtractors, three registers, two shifters, a ROM and some multiplexers as can be seen in Fig. 1. Then, there is a latency of $n$ clock cycles between the input and output data. This implementation is faster than the Bit-serial one because the second has a latency of $(n \times n)$ between the input and outputs data [3].

Typically, about 10 to 20 iterations are enough to give a good result with a lower error rate. More iterations provide more accuracy but request longer adders and consequently reduce the maximum frequency operation and increasing the logic area cell.

In this work three different implementations were developed. The first one performs 8 iterations, the second, 16 iterations and the last, 32 iterations. An evaluation between these implementations is presented in Section 4.

![Fig. 1 – Block diagram architecture](image-url)

4. Simulation and Synthesis Results

The designed modules were synthesized for two different technologies: Spartan3E XCS500E FPGA and X-FAB XC06 5V CMOS Process, using Xilinx ISE 12.1 and Synopsys Design Compiler tools, respectively.

The simulations were carried out using the ISE Simulator and the validation of these one was performed comparing the hardware and software outputs. These results are shown in Tab.1. There is a slightly difference between the hardware and software results because the constant factor (K) was ignored in VHDL implementation.

<table>
<thead>
<tr>
<th>Entries</th>
<th>Software</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_{in}$</td>
<td>$Y_{in}$</td>
<td>$Z_{in}$</td>
</tr>
<tr>
<td>8 bits</td>
<td>100</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>288</td>
<td>4</td>
</tr>
<tr>
<td>16 bits</td>
<td>100</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>288</td>
<td>4</td>
</tr>
<tr>
<td>32 bits</td>
<td>100</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>288</td>
<td>4</td>
</tr>
</tbody>
</table>
Tab.2 shows the FPGA synthesis results for the entire developed module considering maximum frequency and resource utilization (number of CLBs, LUTs and IOBs) for three different hardware implementations with 8, 16 and 32 iterations.

<table>
<thead>
<tr>
<th></th>
<th>XC3S500E</th>
<th>Used</th>
<th>Utilization</th>
<th>Maximum Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 bits</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of CLB</td>
<td></td>
<td>24</td>
<td>Less than 1%</td>
<td>264 MHz</td>
</tr>
<tr>
<td>Number of 4 input LUTs</td>
<td></td>
<td>9</td>
<td>Less than 1%</td>
<td></td>
</tr>
<tr>
<td>Number of bonded IOBs</td>
<td></td>
<td>25</td>
<td>10%</td>
<td></td>
</tr>
<tr>
<td>16 bits</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of CLB</td>
<td></td>
<td>76</td>
<td>Less than 1%</td>
<td>194 MHz</td>
</tr>
<tr>
<td>Number of 4 input LUTs</td>
<td></td>
<td>31</td>
<td>Less than 1%</td>
<td></td>
</tr>
<tr>
<td>Number of bonded IOBs</td>
<td></td>
<td>49</td>
<td>21%</td>
<td></td>
</tr>
<tr>
<td>32 bits</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of CLB</td>
<td></td>
<td>100</td>
<td>Less than 1%</td>
<td>177 MHz</td>
</tr>
<tr>
<td>Number of 4 input LUTs</td>
<td></td>
<td>43</td>
<td>Less than 1%</td>
<td></td>
</tr>
<tr>
<td>Number of bonded IOBs</td>
<td></td>
<td>97</td>
<td>41%</td>
<td></td>
</tr>
</tbody>
</table>

Tab. 3 presents the number of logic cells and equivalent gates (based in a NAND2) for three different frequency operations for standard cells synthesis. This approach provides a tradeoff between maximum frequency operation and logic cells number.

<table>
<thead>
<tr>
<th></th>
<th>Logic Cells</th>
<th>NAND2 Equivalent Gates</th>
<th>Maximum Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 bits</td>
<td>1095</td>
<td>3417</td>
<td>125 MHz</td>
</tr>
<tr>
<td>16 bits</td>
<td>10826</td>
<td>21200</td>
<td>125 MHz</td>
</tr>
<tr>
<td>32 bits</td>
<td>51725</td>
<td>83808</td>
<td>100 MHz</td>
</tr>
</tbody>
</table>

5. Conclusion
In this paper, we have presented a hardware implementation of the CORDIC algorithm in a Bit-Parallel iterative mode. This approach uses less hardware than a Bit-Serial or pipelined structure [3]. The initial study has compared FPGA and logic synthesis and standard cells results. The standard cells synthesis proves that this design can operate at higher frequencies (more than 100 MHz) even for a 600 nanometer technology. Bit-Parallel iterative mode takes \( n \) clock cycles, where \( n \) is the number of algorithm iterations. In this work, were implemented three different architectures, the first one with 8 iterations, the second one with 16 iterations and the third one with 32 iterations. It was proved that more iterations you have, more precision is given and more clock cycles are needed to process an input data.

6. References