DESIGN AND IMPLEMENTATION OF AREA OPTIMIZED CORDIC
PROCESSOR

ARUN MOHAN
PG student, Department of ECE-PG, SONA College of Technology
E-mail: mlarun55@gmail.com

Abstract- This paper describes the design and implementation in VHDL language of a special purpose processor for the calculation of trigonometric functions, based on CORDIC (Coordinate rotation in digital computer) algorithm. CORDIC algorithm implementation has the primary advantage that it is far more economical compared to DSP algorithms in terms of area and power consumption. This algorithm’s implementation has the edge over conventional DSP methods since it requires less complex hardware and due to its flexibility. Since CORDIC architecture provides considerable area optimization, it is ideal for use in Field Programmable Gate Arrays (FPGAs). This property of the CORDIC architecture can be utilized for reducing the number of components in the FFT (Fast Fourier Transform) part of a processor since we are replacing the multiplier part of the FFT with the CORDIC processor. The implementation of this FFT part in an OFDM (Orthogonal Frequency Division Multiplexing) system will eventually result in the reduction of on chip area and hence will lead to more minimization. Thus this paper explores the idea of implementation of a minimized and efficient OFDM receive transmit system in FPGA utilizing the advantages provided by the CORDIC algorithm implementation in the FFT module. Thus we can develop a FPGA system with minimum utilization of CLBs. Modelsim 6.5b is being used for simulation purpose and for synthesis, Xilinx 14.5 is being used. The CORDIC part is implemented in VHDL using SPARTAN3E XC3S500E.

Keywords- CORDIC, FFT, OFDM, DSP.

I. INTRODUCTION

In the modern era, the information interchange is dealing with the transmission and viewing of real time images. The simulation and the processing of real life images are mainly done by the digital signal processing (DSP) systems. While going for real time systems mainly we have to consider the speed and accuracy of the processing system. Calculation of the trigonometric functions is an important task which should be performed by DSP systems. In these systems digital signal processors and microprocessors are performing this task by using the software algorithms developed based on particular system functions. But the software algorithms cannot be so efficient to meet most of the highly demanding tasks of DSP systems.

CORDIC is an algorithm, which is an elegant way for the real time calculation of different trigonometric functions, exponential functions, hyperbolic functions and logs [1],[2]. CORDIC algorithm does not use any calculus based method such as polynomial and rational functions for their calculations. CORDIC algorithm requires only shifters, adders/subtractors, look up tables and interconnections as their components for doing all the tasks. This makes it a hardware efficient algorithm and so it is efficient to include it in FPGAs and microcontrollers where inbuilt multipliers are absent. If multiplications can be done through CORDIC algorithm then for that particular system total area required for components can be optimized a lot in the name of logic gates because the process of multiplication is actually performed using shifting and addition and in CORDIC both are present in its architecture. Thus by using the CORDIC algorithm optimization in area is achieved and thus this algorithm is most suitable for FPGAs. In DSP systems for multiplication purpose, power series and table look up methods are used. This will result in the consumption of a lot of hardware for implementation.

In CORDIC there exist two methods for the calculation of trigonometric and other functions and they are rotation mode and vectoring mode. Fast Fourier Transform (FFT) is an important and very common term coming in the DSP applications [3]. In FFT twiddle factor is another important term and it consists of mainly sine and cosine terms, so by using CORDIC algorithm for the generation of trigonometric functions we can calculate sine and cosine values and we can calculate the twiddle factor. Thus the area optimization is possible in FFT also. Further we can use this FFT in OFDM system and as a result the total system area is optimized for the OFDM system. Due to the simplicity and hardware efficiency, the CORDIC algorithm is most suitable for VLSI implementation.

II. CORDIC ALGORITHM

The COordinate Rotation DiDigital Computer (also known as the CORDIC algorithm) was first described in 1959 by Volder as an elegant way to evaluate trigonometric functions. It was originally developed to replace the analog navigation computer on the B-58 aircraft bomber due to a need for higher accuracy and higher performance. In 1971 Walther extended the CORDIC algorithm to hyperbolic functions, and the
algorithm is today used in many applications such as calculators and robotics [2]. The algorithm belongs to the class of linear convergence algorithms and can likewise be implemented using only shift and add operations making it suitable for VLSI implementations. CORDIC is used for polar to rectangular and rectangular to polar conversions and also for calculation of trigonometric functions, vector magnitude and in some transformations, like discrete Fourier transform (DFT) or discrete cosine transform (DCT). CORDIC can be operated in two modes they rotational mode and vectoring mode. The first mode of operation, called rotation by Volder, rotates the input vector by a specified angle (given as an argument). Here, the angle accumulator is initialized with the desired rotation angle. The rotation decision based on the sign of the residual angle is made to diminish the magnitude of the residual angle in the angle accumulator. In the vectoring mode, the CORDIC rotator rotates the input vector through whatever angle necessary to align the result vector with the x axis. The vectoring function works by seeking to minimize the y component of the residual vector at each rotation. The sign of the residual y component is used to determine which direction to rotate next.

III. TYPES OF CORDIC ARCHITECTURE

CORDIC algorithm for the calculation of sine and cosine values is of three types. Each of these types has their own advantages and disadvantages.

The three types are:
1. Sequential / iterative
2. Parallel / cascaded
3. Pipelined

In Sequential / iterative CORDIC, a single iteration takes place in one clock cycle. It has the main advantages that the hardware complexity is least and it occupies the least area. Also it has the maximum number of clock cycles per iteration and least power consumption. Its disadvantage is that the maximum number of clock cycles is required for the calculation of the output resulting in a slow calculation speed.

In Parallel / cascaded CORDIC algorithm, all the calculations take place within a single clock cycle. Although this algorithm introduces considerable delay, the processing time is reduced compared to the iterative process. This algorithm has the drawback that the amount of hardware required is large and that maximum area is required. Power consumption is highest among the three CORDIC algorithms.

Pipelined CORDIC is the most efficient of the CORDIC algorithms in which the iterations take place in multiple clock cycles. However, different processes take place concurrently such that the execution time is reduced. The number of iterations after which the system gives accurate result can be modeled, considering clock frequency of the system. When operating at greater clock period power consumption in later stages reduces due to lesser switching activity in each clock period. The drawback of this algorithm is that the hardware complexity well as area required is more than sequential architecture. Power consumption is lower than parallel but higher than sequential structure.

IV. MODELING OF ALGORITHM

The mathematical parts of the CORDIC algorithm are basically dependent on rotations and pseudo-rotations. The algorithm can be derived from the rotation transform and the CORDIC algorithm performs a planar rotation.

Graphically, planar rotation means transforming a vector \((x_n, y_n)\) into a new vector \((x_{n+1}, y_{n+1})\).

Using a matrix form, a planar rotation for a vector of \((x_i, y_i)\) is defined as

\[
\begin{bmatrix}
x_f \\
y_f 
\end{bmatrix} = \begin{bmatrix}
\cos \theta & -\sin \theta \\
\sin \theta & \cos \theta
\end{bmatrix} \begin{bmatrix}
x_i \\
y_i
\end{bmatrix}
\]

(1)

The angle rotation can be executed in several steps, using an iterative process. Each step completes a small part of the rotation. Many steps will compose one planar rotation. A single step is defined by the following equation:

\[
\begin{bmatrix}
x_{n+1} \\
y_{n+1}
\end{bmatrix} = \begin{bmatrix}
\cos \theta_n & -\sin \theta_n \\
\sin \theta_n & \cos \theta_n
\end{bmatrix} \begin{bmatrix}
x_n \\
y_n
\end{bmatrix}
\]

(2)

Equation 2 can be modified by eliminating the \(\cos \theta_n\) factor.

\[
\begin{bmatrix}
x_{n+1} \\
y_{n+1}
\end{bmatrix} = \begin{bmatrix}
1 & -\tan \theta_n \\
\tan \theta_n & 1
\end{bmatrix} \begin{bmatrix}
x_n \\
y_n
\end{bmatrix}
\]

(3)
Equation 3 requires three multiplies, compared to the four needed in equation 2.

Additional multipliers can be eliminated by selecting the angle steps such that the tangent of a step is a power of 2. Multiplying or dividing by a power of 2 can be implemented using a simple shift operation.

The angle for each step is given by

\[ \theta_n = \arctan \left( \frac{1}{2^n} \right) \]  

(4)

All iteration angles summed must equal the rotation angle

\[ \sum_{n=0}^{\infty} s_n \theta_n = \theta \]  

(5)

where

\[ s_n = \{-1;+1\} \]  

(6)

This results in the following equation for \( \tan \theta_n = s_n 2^{-n} \)

(7)

Combining equation 3 and 7 results in

\[ \begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \cos \theta_n \begin{bmatrix} 1 & -s_n 2^{-n} \\ s_n 2^{-n} & 1 \end{bmatrix} \begin{bmatrix} x_n \\ y_n \end{bmatrix} \]  

(8)

The algorithm has been reduced to a few simple shifts and additions. The coefficient can be eliminated by pre-computing the final result. The first step is to rewrite the coefficient.

\[ \cos \theta_n = \cos \left( \arctan \left( \frac{1}{2^n} \right) \right) \]  

(9)

The second step is to compute equation 9 for all values of \( n \) and multiplying the results, which we will refer to as \( K \).

\[ K = \frac{1}{P} = \prod_{n=0}^{\infty} \cos \left( \arctan \left( \frac{1}{2^n} \right) \right) \approx 0.607253 \]  

(10)

\( K \) is constant for all initial vectors and for all values of the rotation angle, it is normally referred to as the congregate constant. The derivative \( P \) (approx. 1.64676) is defined here because it is also commonly used.

We can now formulate the exact calculation the CORDIC performs.

\[ \begin{align*}
  x_j &= K(x_i \cos \theta - y_i \sin \theta) \\
  y_j &= K(y_i \cos \theta + x_i \sin \theta)
\end{align*} \]  

(11)

Because the coefficient \( K \) is pre-computed and taken into account at a later stage, equation 8 may be written as

\[ \begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & -s_n 2^{-n} \\ s_n 2^{-n} & 1 \end{bmatrix} \begin{bmatrix} x_n \\ y_n \end{bmatrix} \]  

(12)

At this point a new variable called \( \chi \) is introduced. \( \chi \) represents the part of the angle which has not been rotated yet.

\[ z_{n+1} = \theta - \sum_{i=0}^{n} \theta_i \]  

(13)

For every step of the rotation \( \theta_i \), \( \chi \) is computed as a sign of \( z_n \).

\[ s_n = \begin{cases} -1 & \text{if } z_n < 0 \\ +1 & \text{if } z_n \geq 0 \end{cases} \]  

(14)

Combining equations 5 and 15 results in a system which reduces the not rotated part of angle to zero.

Or in a program-like style:

For \( n=0 \) to \([\inf]\)

If \( (Z(n) \geq 0) \) then

\( Z(n + 1) := Z(n) - \arctan(1/2^n) \)

Else

\( Z(n + 1) := Z(n) + \arctan(1/2^n) \)

End if;

End for;

The \( \arctan(1/2^n) \) is pre-calculated and stored in a table. \([\inf]\) is replaced with the required number of iterations, which is about 1 iteration per bit (16 iterations yield a 16bit result).

If we add the computation for X and Y we get the program-like style for the CORDIC core.

For \( n=0 \) to \([\inf]\)

If \( (Z(n) \geq 0) \) then

\( X(n + 1) := X(n) - (Y(n)/2^n) \)
\( Y(n + 1) := Y(n) + (X(n)/2^n) \)
\( Z(n + 1) := Z(n) - \arctan(1/2^n) \)

Else

\( X(n + 1) := X(n) + (Y(n)/2^n) \)
\( Y(n + 1) := Y(n) - (X(n)/2^n) \)
\( Z(n + 1) := Z(n) + \arctan(1/2^n) \)

End if;

End for;

This algorithm is commonly referred to as driving Z to zero. The CORDIC core computes:
\[ x_j = P(x_i \cos(z_i) - y_i \sin(z_i)) \]
\[ y_j = P(y_i \cos(z_i) + x_i \sin(z_i)) \]
\[ z_j = 0 \]

There’s a special case for driving Z to zero:
\[ x_i = \frac{1}{P} = K \approx 0.60725 \]
\[ y_i = 0 \]
\[ Z_i = 0 \]
\[ x_j = \cos \theta \]
\[ y_j = \sin \theta \]
\[ z_j = 0 \]

Another scheme which is possible is driving Y to zero. The CORDIC core then computes:
\[ x_j = P \sqrt{x_i^2 + y_i^2} \]
\[ y_j = 0 \]
\[ z_j = z_i + \arctan \left( \frac{y_i}{x_i} \right) \]

For this scheme there are two special cases:
1) \[ x_i = x \]
\[ y_i = y \]
\[ z_i = 0 \]
\[ x_j = P \sqrt{x_i^2 + y_i^2} \]
\[ y_j = 0 \]
\[ z_j = \arctan \left( \frac{y_i}{x_i} \right) \]
2) \[ x_i = 1 \]
\[ y_i = a \]
\[ z_i = 0 \]
\[ x_j = P \sqrt{1 + a^2} \]
\[ y_j = 0 \]
\[ z_j = \arctan(a) \]

V. ARCHITECTURAL DESIGN OF THE SYSTEM

CORDIC algorithm designs basically consist of shift and add operations. It is therefore comparatively efficient, resource friendly and easy to implement. Here we use pipelined architecture for designing the system. We use this design methodology in order to increase the throughput and to reduce the latency. The pre-processor mainly deals with the angle calculation and estimation required for the CORDIC rotation and vectoring algorithms. The arithmetic unit performs the actual CORDIC algorithm. The configuration used here is parallel in nature thus giving the design a pipelined structure. This design can perform a CORDIC transformation in each clock cycle, producing a new output every cycle. Arctan ROM is a memory part which stored arctan values for the computation of pseudo rotations. The signals necessary for I/O port definition, including ROM addressing, ready for input (rfi) and output valid (outvalid) are generated by the controller module. The compensation necessary for correcting the scale factor and phase rotation can be introduced by the pre-processor module.

CORDIC processor can be implemented in a number of ways. The serial architecture implementation performs one micro rotation for every clock cycle and output is obtained after n clock cycles. Since the output takes this much clock cycles, this type of architecture is perceived to be slow. In the case of pipelined architecture, iterations are converted into pipeline phases and it consists of n cascaded blocks. Similar to the serial architecture, the first output in this type of architecture is obtained after n clock cycles. But the succeeding outputs are obtained after every clock cycle. In this paper the number of clock cycles used is 15. So after 15 clock cycles, the first output is obtained. Thereafter output is obtained after every clock cycle. Also this paper is designed for 16 bit CORDIC processor.
VI. CORDIC BASED FFT AND ITS IMPLEMENTATION IN OFDM SYSTEMS

Since we can use CORDIC algorithm to calculate sine and cosine values, we can calculate Fast Fourier Transform (FFT) based on it [3].

In FFT, a butterfly unit is used to calculate the necessary output. The butterfly unit consists of a multiplication part which is actually multiplying the twiddle factor with one of the inputs. Twiddle factor is an exponential component and thus consists of sine and cosine terms. The CORDIC technique can be used to calculate these trigonometric values. CORDIC algorithm simplifies the calculation of the trigonometric values thus efficiently simplifying the whole FFT calculation operation and thus area for the implementation of the total system can be optimized.

Orthogonal Frequency Division Multiplexing (OFDM) systems are implemented using FFT and IFFT. By simplifying the FFT part using CORDIC algorithm we can ultimately make a far more efficient and simpler OFDM system implementation in FPGAs.

VII. IMPLEMENTATION OF SYSTEM AND RESULTS

The CORDIC processor was implemented with the necessary units. The units were developed using VHDL and simulated using ModelSim 6.5b and synthesized using Xilinx ISE 14.5. It was implemented in SPARTAN3E XC3S500E FPGA kit.

a. Simulation of CORDIC pipeline architecture

Here we are getting the output after 15 pipeline stages. For getting the first output, we have to wait for 15 clock cycles and after that the output is obtained after every clock cycle. Thus we can increase the throughput and the latency of the system can be reduced. In this section we are actually performing a transformation from polar to rectangular coordinate.

b. Integrated system for final output

This simulation result was obtained after integrating the total system explained above.

CONCLUSION

In this paper we have implemented 16 bit CORDIC processor based on pipelining technique. Thus we are able to create a more efficient processor for calculation of trigonometric functions and we have developed a system using VHDL. Therefore we are able to develop a FPGA system with less number of hardware components because CORDIC needs only shift and addition operators for doing calculations. Further we are able to calculate exponential, square root, division and multiplication using the CORDIC architecture.

REFERENCES