International Journal of Research and Development in Applied Science and Engineering (IJRDASE)

Implementation of CORDIC based Processor using VHDL

Raghawendra Sharma
robie2007@gmail.com

***Abstract*-Digital signal processing (DSP) algorithms exhibit an increasing need for the efficient implementation of complex arithmetic operations. The computation of trigonometric functions, coordinate transformations or rotations of complex valued phasors is almost naturally involved with modern DSP algorithms. In this thesis one of the most computationally high algorithm called the Discrete Cosine Transform is implemented with the help CORDIC(Co ordinate Rotation Digital computer) algorithm which results in a multiplier less architectures and comparison is made between the DCT using Chen's algorithm and DCT using CORDIC as well as new CORDIC algorithm.**
One such simple and hardware efficient algorithm is CORDIC, proposed by (Volder, 1959) [1]. CORDIC uses only shift-and-add arithmetic with look-up-table to implement different functions. By making slight adjustments to the initial conditions and the LUT values, it can be used to efficiently implement Trigonometric, Hyperbolic, Exponential functions, Coordinate Transformation etc. using the same hardware. Since it uses only shift-add arithmetic, VLSI implementation of such an algorithm is easily achievable and also hardware requirement is less. All these lead to efficient area utilization.

Key Word- CORDIC, Pipelined Architecture, Modelsim, VHDL, Xilinx.

1. Introduction:

The digital signal processing landscape has long been dominated by the microprocessors with enhancements such as single cycle multiply-accumulate instructions and special addressing modes. While these processors are low cost and offer extreme flexibility, they are often not fast enough for truly demanding DSP tasks. The advent of reconfigurable logic computers permits the higher speeds of dedicated hardware solutions at costs that are competitive with the traditional software approaches. Unfortunately algorithms optimized for these microprocessors based systems do not map well into hardware (such as FPGA's). While hardware efficient solutions often exist, the dominance of the software systems has kept these solutions out of the spotlight. Among these hardware-efficient algorithms is a class of iterative solutions for trigonometric and other transcendental functions that use only shifts and adds to perform. The trigonometric functions are based on vector rotations, while other functions such as square root are implemented using an incremental expression of the desired function. The trigonometric algorithm is called CORDIC an acronym for Coordinate Rotation DIgital Computer. The incremental functions are performed with a very simple extension to the hardware architecture and while not CORDIC in the strict sense, are often included because of the close similarity. The CORDIC algorithms generally produce one additional bit of accuracy for each iteration [4].

For a long time the field of Digital Signal Processing has been dominated by Microprocessors. However, there were drawbacks in their processing capabilities, because they were unable to provide designers the advantage of single cycle multiply-accumulate instruction as well as special addressing modes. Although these processors are cheap and flexible, they are relatively slow when it comes to performing certain demanding signal processing tasks e.g. Image Compression, Digital Communication and video Processing. The rapid advancements have been made in the field of VLSI and IC design. As a result special purpose processors with custo-architectures have come up. Higher speeds can be achieved by these customized hardware-efficient algorithms exist which map well onto these chips and can be used to enhance speed and flexibility while performing the desired signal processing tasks [5].

2. CORDIC

CORDIC or Coordinate Rotation Digital Computer is a simple and hardware-efficient algorithm for the implementation of various elementary, especially trigonometric, functions. Instead of using Calculus based methods such as polynomial or rational functional approximation, it uses simple shift, add, subtract and table look up operations to achieve this objective. The CORDIC algorithm was first proposed J. E. Volder in 1959 [1]. It is usually implemented in either Rotation or Vectoring mode. In either mode, the algorithm is rotation of an angle vector by a definite angle but in variable direction. This fixed rotation in variable direction is implemented through an iterative sequence of addition/subtraction followed by bit shift operation. The final result is obtained by appropriately scaling the result obtained after successive iterations. Owing to its simplicity the CORDIC algorithm can be easily implemented on a VLSI system [6]. The block diagram of the CORDIC processor is shown in figure 1.

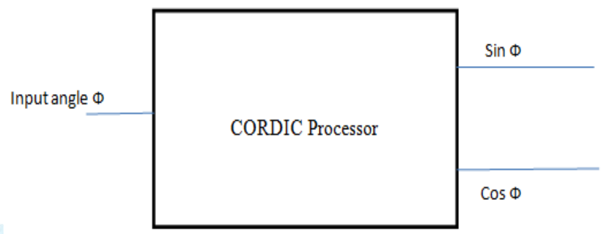

Figure 1: Block Diagram of CORDIC Processor

3. The CORDIC Algorithm

Figure 2 shows the graphical representation of CORDIC algorithm. The basic concept of the CORDIC computation is to decompose the desired rotation angle into weighted sum of a set of predefined elementary rotation angles such that rotation through each of them can be accompolished with simple shift-and-add operations [2].

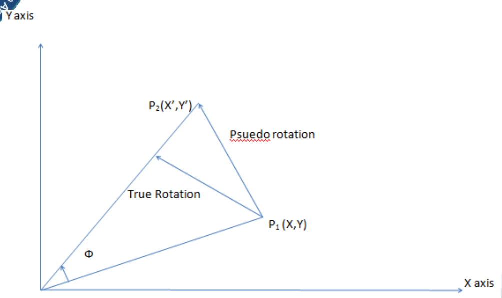

Figure 2: Graphical representation of CORDIC algorithm

In the above figure it has been considered an initial vector $P_1(X,Y)$, which is rotated by angle Φ to get the final vector $P_2(X',Y')$. Rotating a vector in a Cartesian plane by the angle Φ (anti-clockwise) can be arranged such that:-

$$x' = x\cos\Phi - y\sin\Phi$$
$$y' = y\cos\Phi + x\sin\Phi$$

The above equations are further reduced to:-

$$x' = \cos\Phi(x - y\tan\Phi)$$
$$y' = \cos\Phi(y + x\tan\Phi)$$

If the rotation angles are restricted such that $\tan(\Phi) = \pm 2^{-I}$ the multiplication by the tangent term is reduced to a simple shift operation. Arbitrary angles of rotation are obtainable by performing a series of successively smaller elementary rotations. Those angular values are supplied by a small lookup table (one entry per rotation) or are hardwired, depending on the implementation type. However, the required micro-rotations are not perfect rotations, they increase the length of the vector (pseudo-rotation) and in order to maintain a constant vector length (true rotation), the obtained results have to be scaled by a factor K. Removing the scaling constant from the iterative equations yields a shift-add algorithm for vector rotation.

4. CORDIC Architecture:

CORDIC has following architectures:-

A. Folded Word Serial Architecture:

A folded world serial design is also known as iterative bit-parallel design which is obtained simply by duplicating each of the three difference equation shown in hardware.

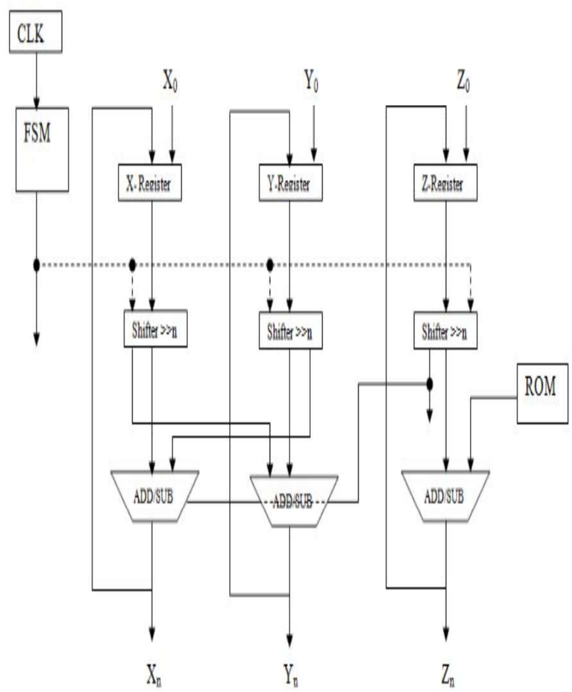

Figure 3: Folded Word serial Architecture

B. Unfolded Parallel Design:

The iterative nature of the CORDIC processor discussed above demands that the processor has to perform iterations at n times the data rate. The iteration process can be unfolded so that each of n processing elements always performs the same iteration. A direct application of the unfolding transformation is to design parallel processing architectures from serial processing architectures. An unfolded CORDIC processor is shown in figure 4.

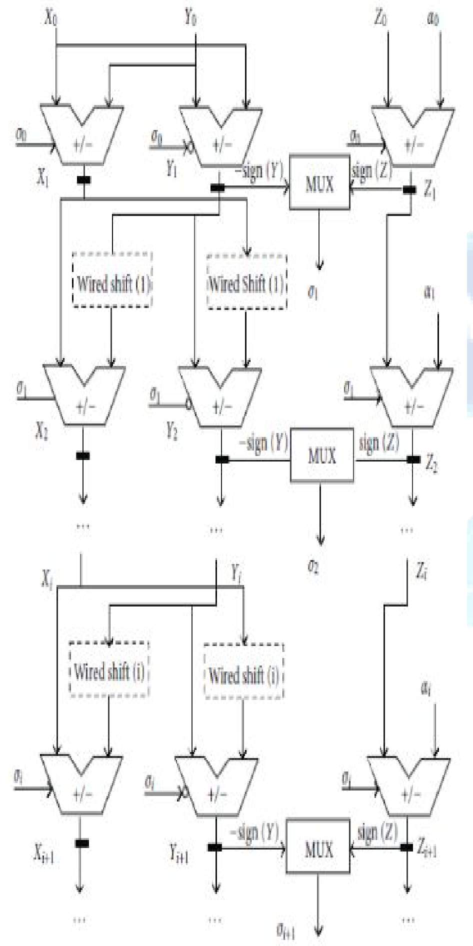

Figure 4: Unfolded Parallel Architecture

5. Binary Angular Measurement:

The CORDIC algorithm is designed using VHDL and all the input values are given in binary format and hence, the output generated is also in binary. The initial coordinates (x,y) can be easily represented in binary format by the decimal to binary conversion rule (division by 2). But the angular value Φ, needs to be converted into binary format following a certain standardization, which is given in the following table:-

Table 1 : Standard Angular Repesentation

Angle Value (degree)	Decimal Equivalent	Binary Representation
180	2048	100000000000
90	1024	010000000000
45	512	001000000000
22.5	256	000100000000
11.25	128	000010000000
5.625	64	000001000000
2.8125	32	000000100000
1.40625	16	000000010000
0.703125	8	000000001000
0.3515625	4	000000000100
0.17578125	2	000000000010
0.08789063	1	000000000001

In the above table it has been considered: 180º is equivalent to decimal value 2048, i.e. 180º ≡ 2048 and the binary representation is in a 12-bit format. In the previous chapter it has been discussed about breaking down the angle Φ into elementary rotation, such that, tan Φ = $\pm 2^{-i}$. The restricted set of angles in binary format in table satisfying the above rule, with the generated error.

Table 2 : Required Angle-Error Table

Required Angle (degree)	Obtained Binary Value	Angular equivalent of Binary Value (degree)	Generated Error (degree)
45	001000000000	45	0
26.6	000100101111	26.23085938	0.03085938
14.03	000010100000	14.0625	0.0325
7.125	000001010001	7.119140625	0.005859
3.576	000000101000	3.515625	0.060375
1.7899	000000010100	1.7578125	0.0320875
0.895	000000001010	0.87890625	0.01609375
0.4476	000000000101	0.439453125	0.0081468
0.2238	000000000011	0.263671875	0.039871
0.1	000000000001	0.087890625	0.012109

A. Assumptions

While providing the inputs to the initial co-ordinates it has been assumed that, if the input angular value ≤ 45º, then the inputs (x,y) are (1,0) or else (0,1).

B. Design Steps:

At first it is assumed that initial vector is rotated by 45º.

STEP1: adding the '0' with the first stored angle (45º) and storing the value in a variable t_0.

STEP2: Now comparing t_0 with the given input table

- If the input angle > t_0, then comparator output, z_0 ='0' and clockwise rotation is done.
- If the input angle < t_0, then comparator output z_0 = '1' and anticlockwise rotation is done

STEP3: Input to shift registers are the given initial co-ordinates or the input to a particular iteration step: x_i, y_i.

Now depending on the iteration step (value of i), x_i and yi are right bit shifted and then added or subtracted depending upon the value of the decision vector z_0, with y_i and x_i respectively to generate x_{i+1} and y_{i+1}.

STEP4: the value of t_0 is added to the next stored angle (26.6º) or the next stored value (26.6º) is subtracted from t_0, depending on the value of z_0. The result generated is stored in variable t_1.

STEP5: Steps 2 to 4 are repeated for the required number of iterations to get the desired result.

STEP6: on completion of the entire iterative process, the scaling factor K, discussed in previous chapter is introduvced in the design. The result obtained from step5, i.e. the final output value of x and y is right bit-shifted by the value of K to get the accurate result. The constant term is desired using controlled shift registers and adders to generate the value almost equal to 0.60725.

6. Result and Discussion:

Both the architectures are designed on Xilinx using VHDL and the results are computed on Intel Pentium G630, RAM 4 GB, 32 bit operating system using ModelSim simulator.

As angle has to be input in hexadecimal form. Hence, converting angle (degree) into Hexadecimal form

For 30 degree ---

1 degree= Pi/180 rad

30Degree= 30*pi/180 ⇔ 0.5423……. (D)⇔ D*2^22 ⇔2182A4(H) --- 2 bits reserved for quadrant(Sign)

Angle: 2182A4(H)

Sin: 200003(H)⇔ H/2^22⇔D

Cos: 376cf9(H)

Figure 5 illustrates the simulation result for 0 degree (000000 H). It shows sin = 000000H and cos = 4000003(H) = 1.0000007152557373046875.

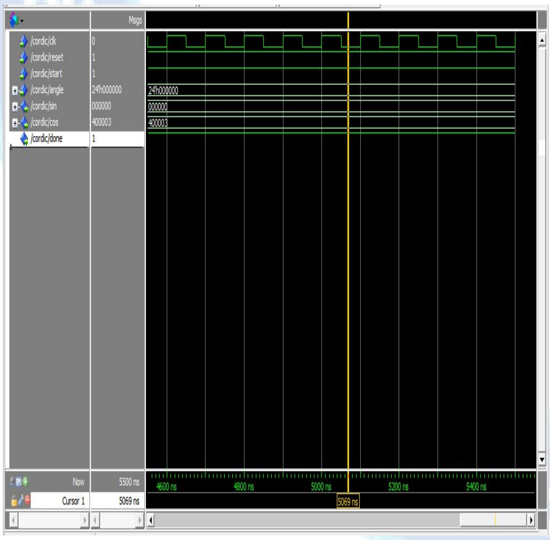

Figure 5: Simulation result for 0º

Figure 6 illustrates the simulation result for 30 degree (2182A4 H). It shows sin = 200003 (H) = 0.50000071525573730 and cos = 376CF9 (H) = 0.8660261631011962.

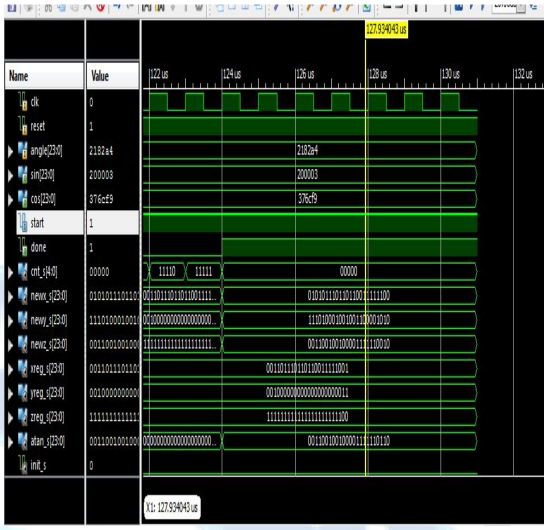

Figure 6: Simulation result for 30º angle

Figure 7 illustrates the simulation result for 45 degree (3243F6 H). It shows sin = 2D413A(H) = 0.707106113433837890625 and cos = 2D410(H) = 0.7071075439453125.

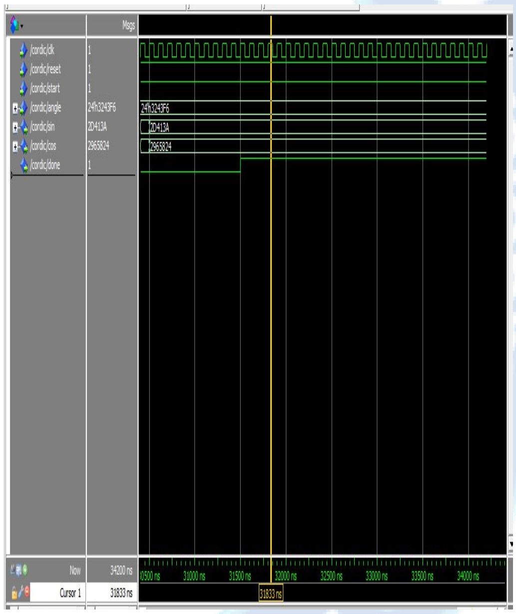

Figure 7: Simulation result for 45º

Figure 8 illustrates the simulation result for 60 degree (430548 H). It shows sin = 376CF9 (H) and cos = 200003(H).

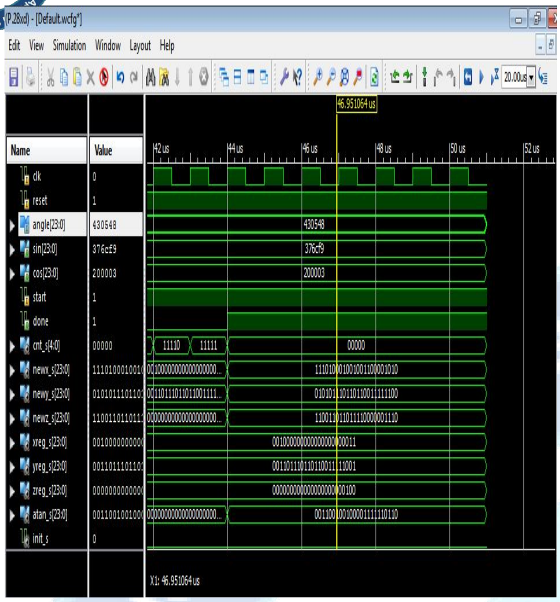

Figure 8: Simulation result for 60° angle

Figure 9 illustrates the simulation result for 90 degree (6487ED H). It shows sin = 400000(H) and cos = FFFFFC(H).

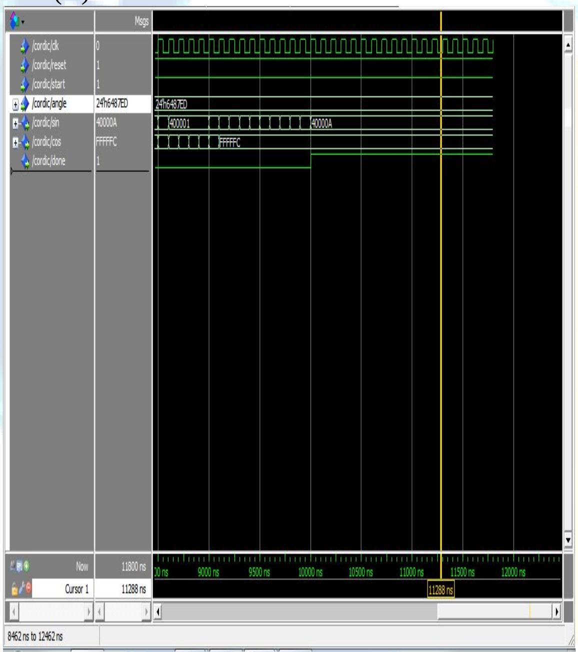

Figure 9: Simulation result for 90°

All the above result are summarized in table 3. Angle are fed in hexadecimal forms and also the values of sin and cosine are in hexadecimal form.

Table 3: Results for Word Serial CORDIC

Angle in degree	Angle in Hex	Sin (H)	Cos (H)
0	000000	000000	400000
30	2182A4	200003	376CF9
45	3243F6	2D413A	296584
60	430548	376CF9	07CCB4
90	6487ED	400000	FFFFFC

Below figure 10 shows external RTL of CORDIC and table 4 shows the device utilization.

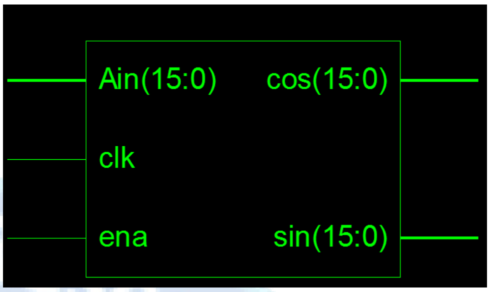

Figure 10: External RTL of CORDIC

Table 4: Device Utilization Summary of Pipelined CORDIC

Device Utilization Summary (Estimated Values)			
Logic Utilization	Used	Available	Utilization
No. of Slices	397	2352	16%
No. of Slice FFs	745	4704	15%
No. of 4 i/p LUTs	725	4707	15%
No. of bonded IOBs	50	288	17%
No. of GCLKs	1	4	25%

7. Conclusions:

The CORDIC algorithms presented in this project are well known in the research. The trigonometric CORDIC algorithms were originally developed as a solution for real time navigation problems. The original work is credited to Jack Volder. The CORDIC algorithm has found its way into diverse application including the 8087 math coprocessor, the HP-35 calculator, radar signal processors and robotics. CORDIC rotation has also been proposed for computing Discrete Fourier, Discrete Cosine, Singular value decomposition.

CORDIC Word Serial Architecture offers low Cost in comparison with pipelined architectecture as it utilizes less resources. Hence these finds application in math Processor and handheld calculators where low cost is primary requirement. Whereas, Pipelined architecture finds application in navigation devices used in ships and air-planes due to its high speed and accuracy.

References:

[1] Volder, J. (1959). The CORDIC Trigonometric Computing Technique. IRE Transactions on Electronic Computing(8), 330-334.

[2] Butner, Y. W. (1987). A new architecture for robot control. IEEE International Conference on Robotics and Automation, 664-670.

[3] Banerjee, A. S. (2001). FPGA realization of a CORDIC based FFT processor for biomedical signal processing. Microprocessors and Microsystems, 131–142.

[4] Pramod K. Meher, J. V.-B. (2009). 50 Years of CORDIC:Algorithms, Architectures and Applications. IEEE.

[5] S. Sathyanarayana, S. R. (2007). Unified CORDIC based processor for image processing. International Conference on Digital Signal Processing, 15, 343-346

[6] Jean-Claude Bajard, S. K.-M. (1994). BKM: A New Hardware Algorithm. IEEE, 43(8), 955-963.

[7] D. S. Cochran (1997). Algorithm & accuracy in the HP 35, Hewlwtt-Packard Journal,23,10.