Trigonometric Function Generation Using CORDIC Algorithm In FPGA

Guhan Rajasekar , Harivignesh S, DESE, IISc

I Introduction and Motivation

- CORDIC (Coordinate Rotation Digital Computer) is a hardware-efficient iterative method which uses rotations to calculate a wide range of elementary functions like inverse trigonometric functions such as arctan, arcsin, arccos, hyperbolic and logarithmic functions, polar to rectangular transform, Cartesian to polar transform, multiplication, and division.
- In this mini-project, the focus is on generating sine and cos of given angle using CORDIC algorithm.
- The main motivation behind using this algorithm is to perform the elementary operations in a *hardware friendly* manner.
- When we say hardware friendly way, it means that the algorithm avoids the use of multipliers and relies only on shifts and additions and subtractions.
- Before discussing the CORDIC algorithm to find sine and cos of a given angle, let us consider the other methods which can be used to implement the same functionality:
  - **Taylor Series**: Given a value of x(in radians), the sine or cos of x can be found out as follows :

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots$$
$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

Figure 1: Taylor Series Expansion of cos(x) and sin(x)

    * Disadvantage with the taylor's series is that it is an infinite series. Moreover, finding out higher powers of x and computing factorials of larger numbers are computationally intensive. So using taylor's series is not a very hardware friendly solution to finding out the sine and cosine of a given angle.
  - **Look Up Table:**In this method we store a look up table holding a list of values of x(in radians or in degrees) and its corresponding values of sine and cosine. When we require cosine or sine of a given x, we directly fetch the required value from the look up table. Else we use a sophisticated interpolation formula to find out the sine or cosine value for given x using the values present in the look up table. But using a complex interpolation formula is once again not a very hardware efficient solution.
- Due to the above reasons, it is clear that we need an algorithm that can perform the elementary operations even when there are hardware constraints. CORDIC algorithm serves this purpose.

II Background Study

- Generalized Hyperbolic CORDIC (GH CORDIC) is a simple and efficient algorithm to calculate trigonometric functions, hyperbolic functions, square roots, multiplications, divisions, and exponential and logarithms with arbitrary base. In this mini-project, the focus will be on generating sine and cosine value for a given angle using CORDIC algorithm.
- CORDIC algorithm has two modes namely
  - Rotation Mode
  - Vectoring Mode
- In this project, rotation mode is used to compute cosine and sine values of given angle.

- When a vector (x_{in}, y_{in}) is rotated by an angle θ the resultant vector is given by,

$$x_R = x_{in}\cos(\theta) - y_{in}\sin(\theta)$$
$$y_R = x_{in}\sin(\theta) + y_{in}\cos(\theta)$$

- If $x_{in} = 1$ and $y_{in} = 0$, then $x_r = cos(\theta)$ and $y_r = sin(\theta)$. The x and y coordinates of the rotated vector will give the sine and cosine of that angle.

- The rotation equations can be written in matrix multiplication form as:

$$\begin{bmatrix} x_R \\ y_R \end{bmatrix} = \cos(\theta)\begin{bmatrix} 1 & -\tan(\theta) \\ \tan(\theta) & 1 \end{bmatrix}\begin{bmatrix} x_{in} \\ y_{in} \end{bmatrix}$$

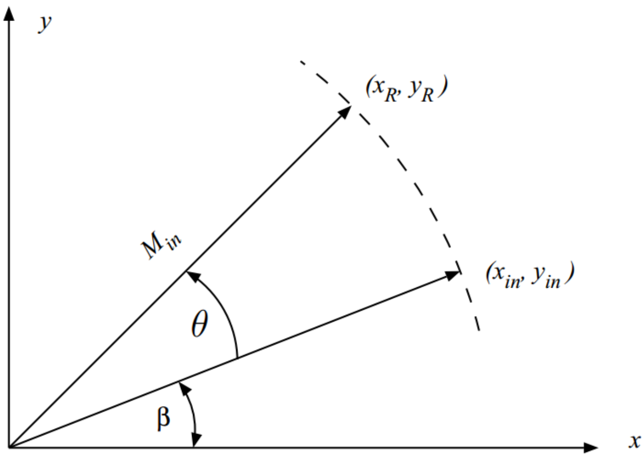

Figure 2: Rotation of a vector by an arbitrary angle θ

- The above rotation operation involves several multiplication and addition operations. These can by bypassed using CORDIC algorithm.

- CORDIC algorithm works based on two key ideas as follows:
  - **1). Breaking down the rotation angle into smaller elementary angles:**
    - The rotation angle is divided into smaller elementary angles provided the following condition is satisfied:

$$\theta_d = \sum_{i=0}^{n} \theta_i$$

    - Here θ_i for i = 0,1,2,3....n form the set of elementary angles.
  - **2). Choosing the elementary angles such that** $\tan\theta_i = 2^{-i}$.
    - By choosing the smaller elementary angles to such that $\tan\theta_i = 2^{-i}$, the matrix multiplication reduces to mere right shift operations and a few addition operations. This is very hardware friendly.

- We also need to multiply $\cos\theta_i$ for every rotation. Since this is a common factor both to the x and y, it can be incorporated as system gain at the very beginning of the algorithm.

- The angle rotated each time gets smaller and smaller hence $\cos\theta_i$ tends to unity and converges to a constant value which can be pre-computed. if the number of iterations in the algorithm is fixed.

- Let K be the scaling factor due to $\cos\theta_i$ and it is computed as follows:

$$K \approx \cos(45^\circ) * \cos(26.565^\circ) * \ldots * \cos(0.895^\circ) = 0.6072$$

- We stop with 0.6072 as successive values of $\cos\theta_i$ will be extremely close to 1 and hence do not make any appreciable difference. This ofcourse comes at the expense of accuracy. If we need more accuracy then we need to consider more values of $\cos\theta_i$ in K. This is a designer's choice.
- To avoid the multiplication of this scaling factor K at the end, we can scale the unit vector along the x-direction by K at the starting itself. Therefore the cordic algorithm is given by:

$$x[i+1] = x[i] - \sigma_i 2^{-i} y[i]$$
$$y[i+1] = y[i] + \sigma_i 2^{-i} x[i]$$

where $\sigma_i \in \{+1,-1\}$ determines the sign of the angle of rotation.

2.1 Negative feedback mechanism of CORDIC

- The value of $\sigma_i \in \{+1,-1\}$ is depends on the total angle of rotation.
- If the total angle of rotation is less than the target angle, then we need to increase the angle of rotation further. In this case, $\sigma_i = +1$.
- If the total angle of rotation is more than the target angle, then we need to decrease the angle of rotation. So $\sigma_i = -1$.
- Thus the final set of equations for the CORDIC algorithm becomes,

$$x[i+1] = x[i] - \sigma_i 2^{-i} y[i]$$
$$y[i+1] = y[i] + \sigma_i 2^{-i} x[i]$$
$$z[i+1] = z[i] - \sigma_i \tan^{-1}\left(2^{-i}\right)$$

.

- The third equation in the above set of equations will help us accumulate the angle of rotations. By comparing this accumulated angle with the target angle, the value of σ_i will be computed at each iteration.
- Here we will need the values of the angles rotated in each iteration $\tan^{-1}(2^{-i})$ which are pre-computed and stored in a look up table as the number of iterations in the algorithm are fixed.

2.2 Extending CORDIC algorithm to angles that do not lie in the First Quadrant

- If the target angle is in **quadrant 2 (between 90^o and 180^o)**, we subtract 90^o to bring the converted angle to first quadrant. We then perform CORDIC on the converted angle. In this method the negative of x-coordinate result and the positive of the y-coordinate result give us sine and cosine of the target angle respectively.
- If the target angle is in **quadrant 3 (between 180^o and 270^o)**, we subtract 180^o to bring the converted angle to first quadrant. We then perform CORDIC on the converted angle. In this method the negative of x-coordinate result and the negative of the y-coordinate result give us cosine and sine of the target angle respectively.
- If the target angle is in **quadrant 4 (between 270^o and 360^o)**, we subtract 360^o from the target angle to bring the converted angle to quadrant 1 and perform CORDIC on the converted angles. The x and y coordinates of the result give us the cos and the sine values of the target angle respectively.

2.3 Block Diagram of Hardware Implementation of CORDIC

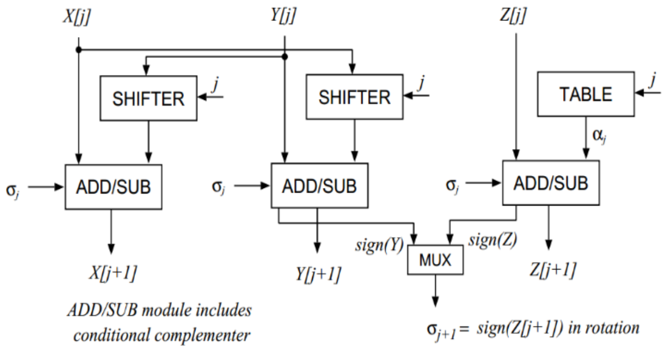

Figure 3: Block Diagram Of CORDIC

III Innovative Features and Design Goals

- Add on features and design goals include the following:
  - Extending CORDIC algorithm to angles that do not lie in quadrant 1.
  - Pipelining the design for higher throughput.

IV Estimated Timeline for each individual task

4.1 Task 1

- Implement the CORDIC algorithm to find the sine and cosine of an arbitrary angle located in the I^{st} quadrant.
- Estimated deadline for this task is 24/3/2024.

4.2 Task 2

- Modify the algorithm to compute the sine and cosine for angles located in any quadrant.
- Estimated deadline is 30/3/2024

4.3 Task 3

- Pipeline the design to improve throughput.
- Estimated deadline is 7/4/2024

References

- Wikipedia article on CORDIC: https://en.wikipedia.org/wiki/CORDIC
- Article on CORDIC in ”allaboutcircuits.com” : https://www.allaboutcircuits.com/technical-articles/an-introduction-to-the-cordic-algorithm/
- NPTEL video:” CORDIC Algorithm ” by Prof Nitin Chandrachoodan , IITM. Lecture Series : *Mapping Signal Processing Algorithms to Architecture*.
- NPTEL video: ” CORDIC Architecture ” Prof. Indranil Hatai . Lecture Series: *Architectural Design of Digital integrated circuits*.