Indian Journal of Science and Technology, *Vol 8(19), DOI: 10.17485/ijst/2015/v8i19/76856, August 2015*

ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Design and Implementation of a Generic CORDIC Processor and its Application as a Waveform Generator

V. Soumya[1*], Raghavendra Shirodkar[2], A. Prathiba[1] and V. S. Kanchana Bhaaskaran[1]

[1]School of Electronics Engineering, VIT University Chennai Campus, Chennai-600127, Tamil Nadu, India; soumya.v2013@vit.ac.in, prathiba.a@vit.ac.in

[2]NXP Semiconductors India PVT LTD, Bangalore-560045, Karnataka, India; raghavendra.shirodkar@nxp.com

Abstract

Background: With the advent in hand held mobile computing devices, the demand for high performance compact processors is increasing. In this work a processor is designed with hardwired instructions for elementary mathematical functions like sine, cosine, sinh, cosh, division and multiplication. **Methods:** The processor employs Coordinate Rotation Digital Computer (CORDIC) algorithm for efficient hardware implementation of the above mentioned instructions. The parallel and pipelined implementation of the processor is carried out. The pipelined processor is configured as waveform generator. The novelty of this work is the integration of both trigonometric and hyperbolic operations in the same processor. **Findings:** ASIC Implementation is carried out with 40nm technology libraries. The parallel processor so designed operates at maximum frequency of 24.23 MHz and pipelined processor operates at maximum frequency of 261.36 MHz. **Conclusion:** This increase in operating frequency is achieved at the cost of increased silicon area and optimal power dissipation. The waveform generator generates sine, cosine waves of 3.5 MHz and sine hyperbolic, cosine hyperbolic waves and exponential waves of 7.9 MHz. The limitation being the waveform generator generates waves of constant frequency. Additional circuit is required in generating waves of different frequencies.

Keywords: Coordinate Rotation Digital Computer (CORDIC), Parallel Architecture, Pipelined Architecture, Waveform Generator

1. Introduction

The targeted application of the designed CORDIC processor is in (DSP) Digital Signal Processing applications, which demand real time processing of data. The hardware implementation of such elementary mathematical functions can be carried out in a wide variety of ways. The most popular ones are as follows.

- Lookup table based approach
- Taylor Series based approach
- CORDIC based approach

1.1 Lookup Table based Approach

This is the fastest means of computation. However, the precision is directly proportional to the size of lookup table. Higher the desired accuracy, larger is the size of look up table which in turn demands more on-chip memory. This exponentially increases the area requirement.

1.2 Taylor Series based Approach

This is based on the Taylor series expansion of the desired function. This kind of implementation is proved area efficient. However, it takes a comparatively longer time while converging to the desired accuracy.

1.3 CORDIC based Approach

This acts as a bridge between both of the above approaches. The attractive feature of this algorithm is that the above listed mathematical operations can be evaluated through mere shift and add operations and using a small look up

*Author for correspondence

table for constants[1]. Thus, this algorithm is very popular among the hardware designers.

Due to the advantage cited above, the CORDIC algorithm is used in this work, for the hardware implementation of sine, cosine, sine hyperbolic, cosine hyperbolic and exponential functions, and the multiplication and the division instructions of the processor. The CORDIC being an iterative algorithm as will be explained in Section 2 can be implemented using the parallel or the pipelined architecture.

1.4 Parallel CORDIC

The implementation as a parallel CORDIC is completely combinatorial. The number of stages is equal to the number of iterations. The output of the previous stage is fed as input to the next stage. The critical path is thus long, and it reduces frequency of operation as a result. This implementation avoids the initial latency and the result is available in one clock cycle.

1.5 Pipelined CORDIC

The parallel CORDIC is pipelined by inserting registers between each of the stages. Due to this insertion of the pipelining registers, the critical path is reduced to the path between two registers. Thus, it drastically increases the frequency of operation. However, it has the initial latency equal to the number of stages.

In this work, the ASIC implementation of the processor with both the parallel and pipelined CORDIC is carried out. The performance parameters, namely, the area, the frequency of operation and the power dissipation are measured and duly compared.

1.6 Waveform Generators

The function generators or the signal generators are the electronic components that generate the various repeating or non-repeating waves. These signals are needed in calibrating, testing, measuring and trouble-shooting the electronic equipments. The most popular way of designing the digital waveform generator is through the LUTs. However, these LUTs consume a huge area and resources when used in ASIC implementation. In Reference[2] a DDS (direct digital synthesizer) is designed that generates trigonometric waves using basic CORDIC which operates at 380MHZ. Reference[3] explains design of a sine and cosine waveform generator which operates at 13.3MHz. A Scale free Hyperbolic CORDIC algorithm is designed in[4] which operates at 56.38 MHz. FPGA Implementation of a RA CORDIC processor is discussed in Reference[5].

In this work, the pipelined CORDIC processor combined with a small LUT is used to implement a waveform generator that generates sine, cosine, sine hyperbolic, cosine hyperbolic and exponential waves. The processor is also provided with hardwired instructions for elementary functions like multiplication, division, sine, cosine and exponential functions which facilitate the use of the processor for generic applications. The processor also has instructions for generating both trigonometric and hyperbolic waves unlike processors in references[2,3,6] which either generates trigonometric waves or hyperbolic waves.

2. Algorithms

2.1 Introduction to CORDIC

The CORDIC algorithm was devised using the basic foundations of the two dimensional geometry. Consider a vector with unit magnitude [1, 0] as shown on Figure 1 (a). When this vector is rotated by an angle Φ, the x projection gives sine of angle Φ and the y projection gives the cosine of angle Φ.

On generalizing this analysis, consider a vector with initial vector position (x, y) as shown in Figure 1 (b). This

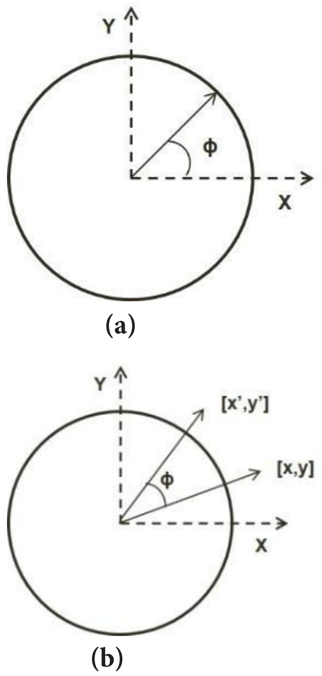


Figure 1. **(a)** Circle with unit vector [0 1] and **(b)** Circle with vector [x, y].

vector is rotated by an angle Φ. The new vector position is then given by

$$x' = x \cos(\Phi) - y \sin(\Phi) \quad (1)$$

$$y' = y \cos(\Phi) + x \sin(\Phi) \quad (2)$$

On rearranging the terms in equations (1) and (2)

$$x' = \cos(\Phi)\, [x - y \tan(\Phi)] \quad (3)$$

$$y' = \cos(\Phi)\, [y + x \tan(\Phi)] \quad (4)$$

Here, assume $\tan(\Phi)=2^{-i}$. This assumption results in the equations to be useful only for the discrete set of angles. In order to make the above set of equations more generic, the algorithm is made iterative as shown below.

$$x_{i+1} = K_i\, [x_i - y_i{}^* d_i{}^* 2^{-i}] \quad (5)$$

$$y_{i+1} = K_i\, [y_i + x_i{}^* d_i{}^* 2^{-i}] \quad (6)$$

$$z_{i+1} = z_i - \tan^{-1}(2^{-i}) \quad (7)$$

$$K_i = \cos(\tan^{-1}(2^{-i}))$$

$$K = \Pi\, (1+2^{-2i})^{-1/2}$$

as i tends to infinity K= 1.6468 the CORDIC can be operated in two modes, namely,

- Rotation mode
- Vector mode

2.1.1 Rotation Mode

The vector is rotated by the desired angle. The direction of rotation is decided by the angle of iteration.

$$x_{i+1} = K_i\, [x_i - y_i{}^* d_i{}^* 2^{-i}] \quad (8)$$

$$y_{i+1} = K_i\, [y_i + x_i{}^* d_i{}^* 2^{-i}] \quad (9)$$

$$z_{i+1} = z_i - \tan^{-1}(2^{-i}) \quad (10)$$

where $d_i = -1$, if $z_i<0$
$+1$, if $z_i>0$

After completion of the desired number of iterations, the result obtained is given by

$$x_n = K\, [x_0 \cos(z_0) - y_0 \sin(z_0)] \quad (11)$$

$$y_n = K\, [x_0 \sin(z_0) - y_0 \cos(z_0)] \quad (12)$$

$$z_n = 0$$

On starting the algorithm with the initial parameters $x_0 = 0.607(1/K)$ and $y_0 = 0$, the cosine and sine of the initial angles are made available in x and y variables after completion of n iterations. The more is the number of iterations, higher is the accuracy.

2.1.2 Linear Mode

The vector is rotated towards the x axis. The direction of rotation is decided by the y projection of the vector.

$$x_{i+1} = K_i\, [x_i - y_i{}^* d_i{}^* 2^{-i}] \quad (13)$$

$$y_{i+1} = K_i\, [y_i + x_i{}^* d_i{}^* 2^{-i}] \quad (14)$$

$$z_{i+1} = z_i - \tan^{-1}(2^{-i}) \quad (15)$$

Where $d_i = +1$ if $y_i<0$
-1 if $y_i>0$

After completion of the desired number of iterations, the result obtained is represented as follows.

$$x_n = K\sqrt{(x_0{}^2+y_0{}^2)} \quad (16)$$

$$y_n = 0$$

$$z_n = z_0 + \tan^{-1}(y_0/x_0) \quad (17)$$

On starting the algorithm with the initial parameters $z_0= 0$, the magnitude and the direction of the vector is available in x_n and y_n. The above explained algorithm was proposed by Volder and is applicable only for the Circular coordinates.

This was later extended to the linear and hyperbolic coordinates by Walther in 1971 by adding an additional parameter to decide the coordinates as summarized in Table 1.

$$x_{i+1} = K_i\, [x_i - y_i{}^* m^* d_i{}^* 2^{-i}] \quad (18)$$

$$y_{i+1} = K_i\, [y_i - x_i{}^* d_i{}^* 2^{-i}] \quad (19)$$

$$z_{i+1} = z_i - \tan^{-1}(2^{-i}) \quad (20)$$

$$y_{i+1} = K_i\, [y_i - x_i{}^* d_i{}^* 2^{-i}]$$

Here, the 'm' defines the coordinates.

Table 1. Summary of Welther's CORDIC

m	Rotational mode	Linear mode
1	$x_n = K(x_0 \cos z_0 - y_0 \sin z_0)$ $y_n = K(x_0 \sin z_0 + y_0 \cos z_0)$	$x_n = K\sqrt{(x_0{}^2 + y_0{}^2)}$ $z_n = z_0 + \tan^{-1}(y_0/x_0)$
0	$x_n = x_0$ $y_n = y_0 + x_0 z_0$	$x_n = x_0$ $y_n = x_0 + y_0 / z_0$
−1	$x_n = K(x_0 \cosh z_0 - y_0 \sinh z_0)$ $y_n = K(x_0 \sinh z_0 + y_0 \cosh z_0)$	$x_n = K\sqrt{(x_0{}^2 - y_0{}^2)}$ $z_n = z_0 + \tanh^{-1}(y_0/x_0)$

2.2 Multiplication using CORDIC

With reference to Table 1, it can be observed that the product of the two numbers is obtained when the CORDIC engine operates in rotational mode with linear coordinates. One of the operands is fed as the x coordinate of the vector and the second operand is fed as the desired angle of rotation (z) with zero y coordinate. If number of bits in the operands is 'n', then iterating the CORDIC equations 'n' times yields the product of the desired accuracy. The domain of convergence of the CORDIC is limited as explained in Reference[6]. The reliable product is obtained only when the operands are in the range {–1, 1}. Thus, the application of such a multiplier is very limited. In order to make the multiplier more generic, the pre-processing of the operands and the post-processing of the product is essential. The CORDIC with these additional steps is explained in the next section.

2.3 Division using CORDIC

Referring to Table 1, it can be observed that the quotient of the two numbers is obtained when the CORDIC engine operates in vector mode with the linear coordinates. The dividend is fed as the y coordinate and the divisor is fed as the desired angle (z) with the x coordinate being equal to zero. As explained in Reference[6], the CORDIC engine has its predefined limitations. The accurate quotient is available only when the dividend is less than the divisor or in other words, when (y<z). Thus, in order to make the algorithm generic the pre-scaling of the operands and the post-scaling of the quotient is required. The updated algorithm is described in the Section 3.

2.4 Trigonometric Functions using CORDIC

When the CORDIC engine operates in the rotational mode with the circular coordinates, the sine and cosine of the desired angle is obtained. The x coordinate should be initialized to 1/K (0.607) and the y coordinate should be initialized to zero for obtaining the scale free results. The sine and cosine values of the angles are simultaneously obtained in the x and y coordinates after completing the iterations. The domain of operation for the trigonometric functions is ($z<\pi/2$).

The algorithm can be extended to the entire range (0 to 360^0) by pre-processing the input angle before feeding to the CORDIC engine as explained in the next section.

2.5 Hyperbolic Functions using CORDIC

The hyperbolic functions sinh and cosh can be evaluated by operating the CORDIC engine in the rotation mode for hyperbolic coordinates. The calculation of the hyperbolic functions using the CORDIC is more complex due to the fact that the iteration angle is $z_i=\tanh^{-1}(2^{-i})$ which does not satisfy the convergence criterion. However, with reference to[7] it can be proved that

$$z_{n-1} > z_i - \sum_{j=i+1}^{n-1} zj - z_{3i+1}$$

Hence, the iterations {4, 13, 40….k, 3k+1} have to be repeated to extend the range of convergence. Various algorithms have been proposed to address the convergence issue of the hyperbolic functions. One popular way is using the mathematical identities, such as the Taylor expansion of hyperbolic functions. However, this method is not suitable for the hardware implementation. In this work, the expanded hyperbolic CORDIC algorithm as suggested in references[4] is used.

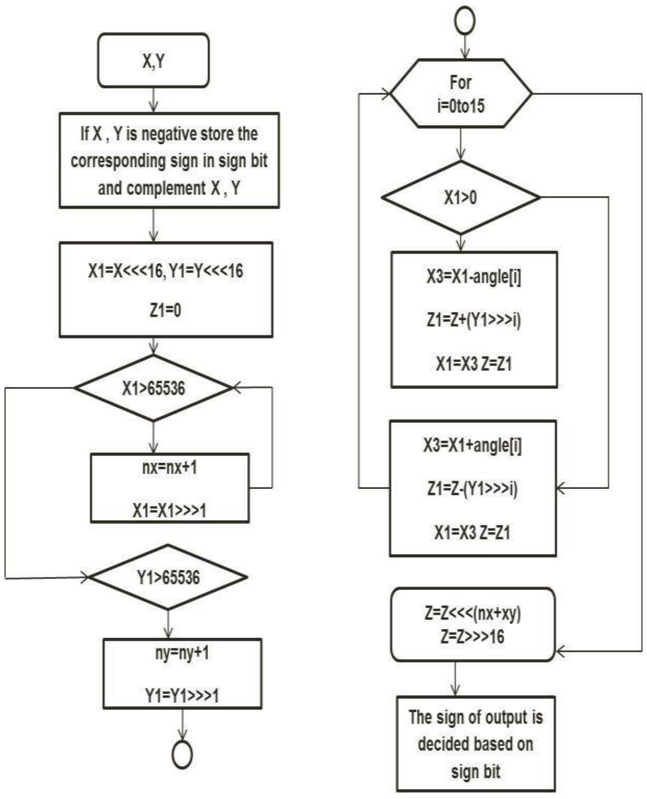


Figure 2. Flow chart of Enhanced CORDIC for Multiplication.

3. Proposed Enhancement to CORDIC

3.1 Enhanced CORDIC for Multiplication

The enhanced CORDIC algorithm for multiplication shown in Figure 2 operates on 16 bit operands. The supported product range is –32768 to +32767. Since the operands are 16 bit wide, 16 iterations are sufficient for producing convincing results. In order to deal with the fractional numbers, logic 1 is assumed to be equivalent to 2^{16} and other fractional numbers are scaled accordingly. The input operands are first shifted towards left by 16 bit position to scale the operands by 2^{16}. Since the CORDIC engine supports operands in the range {–1, +1}, the operands are divided by two until the operands fall in this range, and the number of times the division is carried out is stored in the counters. After this stage, the operands can be fed to the CORDIC engine for the product calculation. The counters are used to post scale the product. The stored sign bits are used in determining the sign of the product.

3.2 Enhanced CORDIC for Division

Figure 3 gives the flowchart of enhanced CORDIC for division. The handling of fractional numbers is the same as in the multiplication. The CORDIC engine supports the quotient less than one. Hence, if the dividend is greater than the divisor, the dividend is divided by two until the dividend is less than one, and the number of times the division is carried out is stored in the counter. After this stage, the operands can be fed to the CORDIC engine for the quotient calculation. The counter is used to post scale the quotient. The stored sign bits are used in determining the sign of the quotient.

3.3 Enhanced CORDIC for Trigonometric Calculation

As demonstrated in flowchart Figure 4. The input angle is 32 bit wide and stored as variable z. The upper two bits of the angle define the quadrants. According to the conventional CORDIC algorithm, in order to extend the region of operation to the entire domain of 0 to 360 degree, a slight modification of the inputs is necessary before applying to the CORDIC engine. If the angle lies in the range 0 to 90 degree or 0 to –90 degree, then no modifications is necessary. The angle is applied as it is, x coordinate is $1/K^{*}2^{15}$ and y coordinate is 0. Here, the logic one is assumed to be 2^{15}. If the angle lies in the range of 90 to 180 degree, where the sine is positive and the cosine is negative, 90 degree is subtracted from the angle. And,

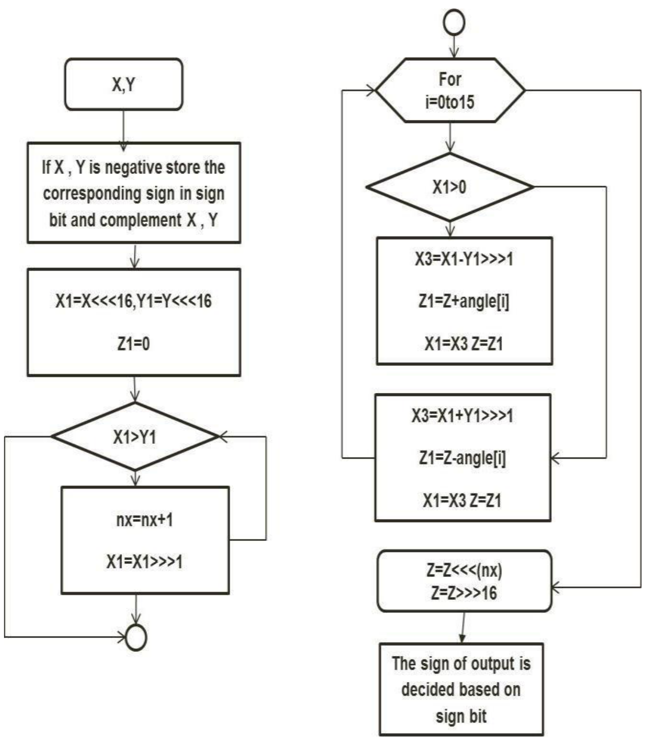


Figure 3. Flow chart of Enhanced CORDIC for Division.

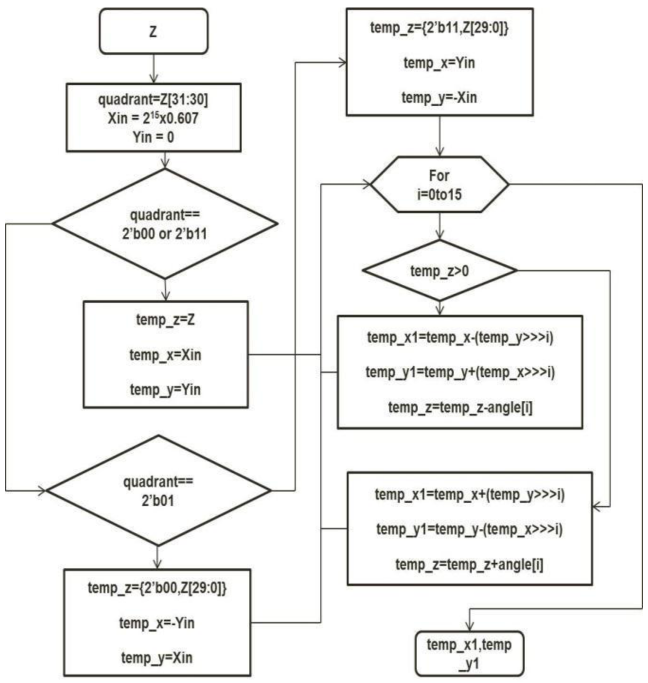


Figure 4. Flow chart of Enhanced CORDIC for trigonometric function.

the y coordinate is $1/K^{*}2^{15}$ and the x coordinate is 0. If the angle lies in the range 180 to 270 degree, where the tan is positive, 90 degree is added to the angle, y coordinate is $-1/k^{*}2^{15}$ and the x coordinate is 0. After sixteen iterations, the sine of the angle is available in the y coordinate and the cosine of the angle is available in the x coordinate simultaneously.

3.4 Expanded CORDIC for Hyperbolic Functions

Theoretically, on operating the CORDIC engine in the rotation mode with hyperbolic coordinates, yields sinh and cosh of the input angle. Unlike the other trigonometric functions, the range of hyperbolic functions is unbound. Thus, the expanded hyperbolic algorithm proposed in References[1,8] is attractive for the hardware implementation of hyperbolic functions. In this approach, a few additional iterations are added to the basic CORDIC for the negative indices of i.

$$\theta_i = \tanh^{-1}(1-2^{i-2}) \text{ for } i<0 \qquad (21)$$

Thus, the set of equations used in this modified CORDIC are as follows.
For i < 0

$$x_{i+1}= K_i\,[x_i + y_i{}^{*}d_i{}^{*}(1-2^{i-2}\,)] \qquad (22)$$

$$y_{i+1}= K_i\,[y_i + x_i{}^{*}d_i{}^{*}(1-2^{i-2}\,)] \qquad (23)$$

$$z_{i+1}=z_i - \tanh^{-1}(1-2^{i-2}\,) \qquad (24)$$

For i>0

$$x_{i+1}= K_i\,[x_i + y_i{}^{*}d_i{}^{*}2^{-i}] \qquad (25)$$

$$y_{i+1}= K_i\,[y_i + x_i{}^{*}d_i{}^{*}2^{-i}] \qquad (26)$$

$$z_{i+1}=z_i - \tanh^{-1}(2^{-i}) \qquad (27)$$

Here, the maximum supported input angle is given by

$$\Theta_{\max} = \sum_{-m}^{0} \tanh^{-1}(1-2^{i-2}) + \tanh^{-1}(2^{-n}) + \sum_{1}^{n} \tanh^{-1}(2^{-1}) \qquad (28)$$

With reference to the References[1,8], for the processor with 32 bit registers, 8 additional iterations are added for the negative indices along with the conventional 16 iterations for positive indices and repeated two iterations corresponding to the indices 4 and 13 for increasing the range of convergence. Thus, in total, 26 iterations are required for evaluating the sinh and the cosh functions. The maximum sinh and cosh value supported by this implementation is of the range given by $\{-\ 4.4^{*}10^{6}$ to $+\ 4.4^{*}10^{6}\}$. The flow is explained in Figure 5.

4. Hardware Implementation

4.1 Basic CORDIC Unit

Figure 6 depicts the basic CORDIC block. It is made of three adders, two shifters and a Look Up Table (LUT) containing the iteration angles. The CORDIC being an iterative algorithm, the same set of operations is carried out repeatedly with the output of the previous stage fed as input to the successive stage. In this project, the parallel and the pipelined CORDIC architectures are implemented and their performance parameters are measured as discussed in the ensuing sections.

4.2 Parallel CORDIC Architecture

As described above, the elementary operations such as the multiplication, division, sine and cosine require sixteen iterations in arriving at the desired output. On the other hand, the hyperbolic functions sinh, cosh and the exponential functions require twenty six iterations. Thus, for the parallel implementation sixteen CORDIC units shown in Figure 6

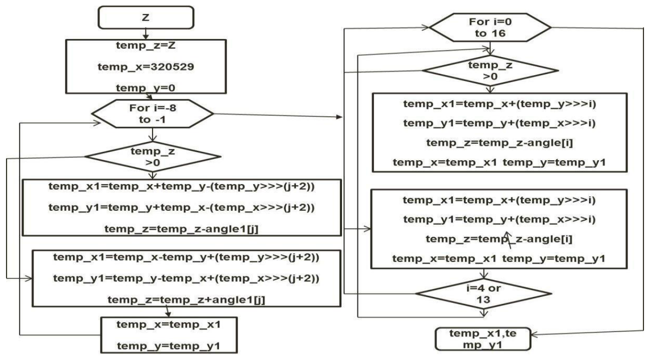

Z
temp_z=Z
temp_x=320529
temp_y=0
For i=-8 to -1
temp_z >0
temp_x1=temp_x+temp_y-(temp_y>>>(j+2))
temp_y1=temp_y+temp_x-(temp_x>>>(j+2))
temp_z=temp_z-angle1[j]
temp_x1=temp_x-temp_y+(temp_y>>>(j+2))
temp_y1=temp_y-temp_x+(temp_x>>>(j+2))
temp_z=temp_z+angle1[j]
temp_x=temp_x1
temp_y=temp_y1
For i=0 to 16
temp_z >0
temp_x1=temp_x+(temp_y>>>i)
temp_y1=temp_y+(temp_x>>>i)
temp_z=temp_z-angle[i]
temp_x=temp_x1 temp_y=temp_y1
temp_x1=temp_x+(temp_y>>>i)
temp_y1=temp_y+(temp_x>>>i)
temp_z=temp_z-angle[i]
temp_x=temp_x1 temp_y=temp_y1
i=4 or 13
temp_x1,temp_y1

Figure 5. Flow chart expanded hyperbolic CORDIC.

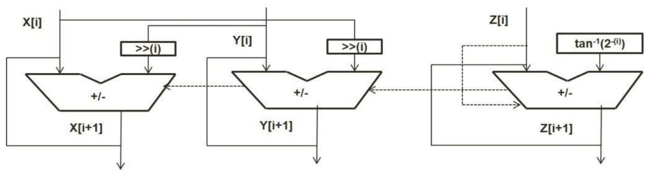


Figure 6. Basic CORDIC unit.

are connected in parallel such that the output of one stage is fed to the subsequent stage as shown in Figure 7.

Advantages

- It is completely combinational and avoids the clock switching power.
- Fixed shift is needed at each stage which can be achieved by wiring.
- Iteration angles for each stage can be hardwired as constants thus avoiding the memory space.
- Registers are needed only at the input and after the final stage for calculating the frequency of operation. The output is obtained in one clock cycle.

Limitations

- The critical path length is very large, which reduces the frequency of operation to a significant extent.

4.3 Pipelined CORDIC Architecture

The parallel CORDIC architecture is converted to the pipelined architecture by inserting the pipelined registers between every stage as seen in Figure 8. The attractive part is that the processing of next sample is started even before the completion of the current sample.

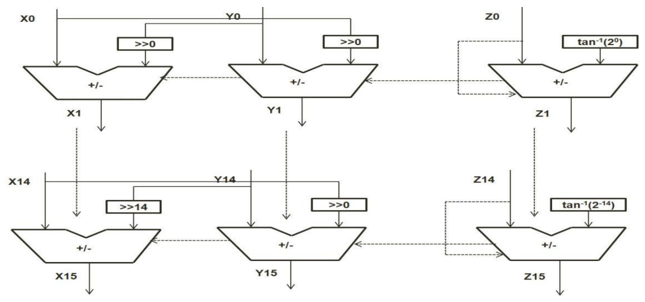


Figure 7. Parallel CORDIC architecture.

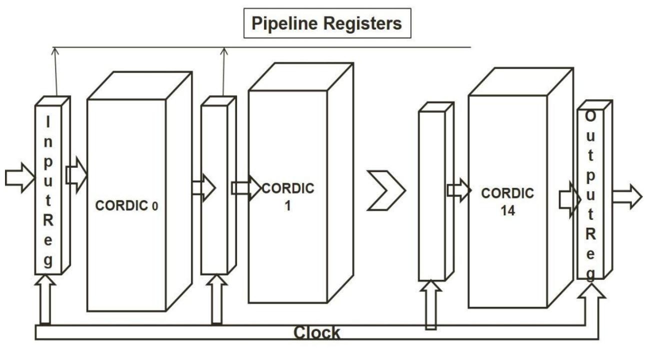


Figure 8. Pipelined CORDIC architecture.

Advantage

- In this implementation, the critical path is the path between the two pipelined registers. Thus, it divides the critical path and in the process, significantly increases the frequency of operation

Limitation

- This kind of implementation suffers from the initial

4.4 CORDIC Processor Architecture

Figure 9 gives the block diagram of the proposed CORDIC processor. The processor is designed for 32 bits of operation. Hence, the input and output busses are of 32 bit width. The program bus is of 10 bit width, thus supporting a program memory of 1K byte.

4.4.1 Datapath

The datapath of the processor comprises of the following.

- CORDIC block for circular and linear coordinates
- CORDIC block for hyperbolic coordinates
- Instruction decoder
- Program counter
- Register set

The CORDIC block designed for operating in the linear and the circular coordinates is used for evaluating the product, the quotient and the trigonometric functions, whereas the CORDIC block operating in the hyperbolic mode is used in evaluating the hyperbolic and the exponential function. The processor is provided with sixteen 32 bit registers for storing the operands.

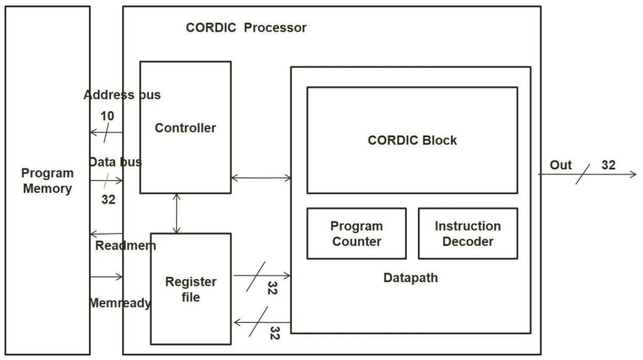


Figure 9. CORDIC Processor block diagram.

4.4.2 Controller

Figure 10 gives the operating states of the processor controller

- **Reset:** All the logic blocks are reset. The program counter is loaded with logic zero and thus the address bus points to the first memory location of the program memory.
- **Fetch:** The instruction is loaded from the memory location pointed to by the address bus into the instruction decoder.
- **Decode:** The instruction is decoded to fetch the opcode and the register numbers. The opcode is used in generating the control signals and register numbers are fed as input to the register set.
- **Execute:** Based on the control signals generated by the controller, the desired functions are carried out by the respective block in the datapath.
- **Load:** This state is activated only with the move instruction. On reception of this instruction, the actual data needs to be fetched from the next memory location.

4.5 Waveform Generation

The pipelined processor which executes the set of the pipelined instructions generating one output for each of the clock cycles can be configured as a waveform generator by using a relatively smaller look up table.

To evaluate the trigonometric functions, the input to the CORDIC engine is fed in the form of angle*2^{32}/360. The pre-evaluated angles with a difference of five degrees in the range 0 to 360 are stored in a lookup table and fed repetitively to the CORDIC engine operating in rotation mode with circular coordinates to generate the sine and cosine waves.

To evaluate the hyperbolic functions, the input to the CORDIC engine is fed in the form angle*2^{31}/360. Unlike the trigonometric functions, the input to the hyperbolic function is unbound. Thus, to handle both the negative and the positive inputs, the angle is scaled by multiplying with 231. The pre-evaluated angles with a difference of one degree in the range –16 to 16 are stored in a lookup table and fed repetitively

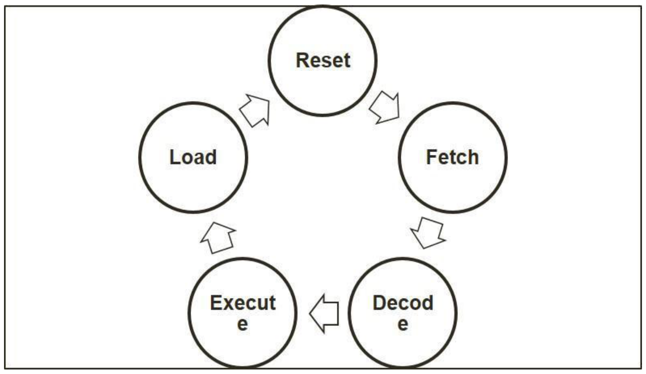


Figure 10. Controller state machine.

5. Results and Discussion

5.1 Synthesis Results

It can be concluded from the results in Table 2 that if the area and power dissipation are of concern, the parallel CORDIC architecture needs to be preferred. On the other hand, if the processor is designed for an application that demands very high frequency of operation, then the pipelined architecture can be selected.

The parallel processor delivers results in one clock cycle, though suffering from a very large critical path delay which drastically reduces the frequency of operation. The frequency of operation is increased ten times in the pipelined implementation relative to the parallel implementation. However, pipelined architecture is realized at the cost of more number of gates and area. In this work there is 28% increase in number of gates and 39.82% increase in area.

5.2 Simulation Results

Figure 11 shows the simulation results on executing the division instruction using the parallel and the pipelined processor. As already discussed, the pipelined processor suffers from an initial delay. On the other hand, the parallel processor delivers results in one clock cycle, though suffering from a very large critical path delay. It thus drastically reduces the frequency of operation.

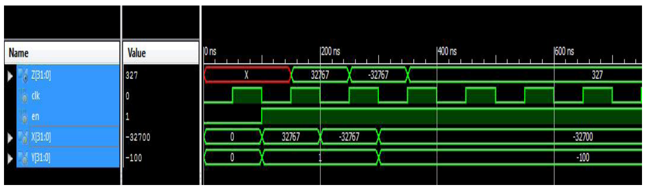
Table 2. Synthesis summary

	Parallel Implementation	Pipelined Implementation	Waveform Generator
Area	47437 μm^2	66328 μm^2	67476 μm^2
Number of gates	22955	29601	30306
Power	4.403mW	6.298mW	10.44mW
Frequency of operation	24.23 MHz	261.36MHz	255.29MHz
Critical Path	41.268ns	3.826ns	3.917ns

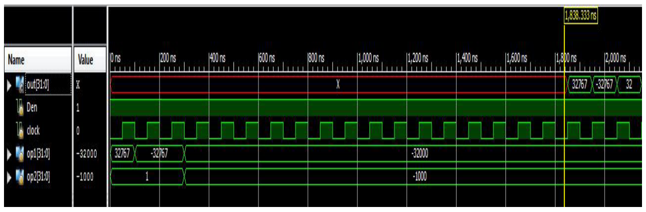
In this implementation, since the CORDIC in linear and the circular coordinates require sixteen iterations, the corresponding results for the first set of inputs is available after 16 clock cycles. On the other hand, the CORDIC for the hyperbolic coordinates requires twenty six iterations. Hence, it has an initial latency of 26 clock cycles.

5.3 Static Power Analysis Results

Static power analysis of the processor is carried out using Cadence® Voltus tool for the typical corner model. The graph in Figure 12 gives the module wise power dissipation and clearly suggests that the parallel implementation of the CORDIC is more of a combinational circuit with very less sequential elements. The change in the output of the previous stage triggers the next stage thus avoiding the clock switching which is one of the major power consuming factors. On the contrary, due to the insertion of the pipelined registers, the pipelined implementation of CORDIC has a large number of sequential elements. It is clear from Figure 13 that a considerable amount of power dissipation is contributed by these sequential elements. In this structure, the output of the previous stage is fed to the successive stage only when triggered by the clock. Thus, there is a large amount of switching power dissipation in the clock network.

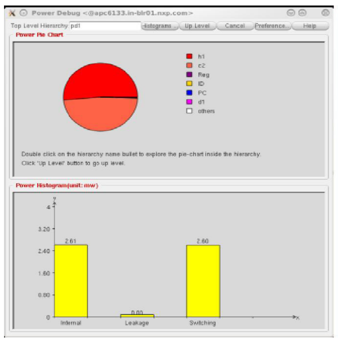


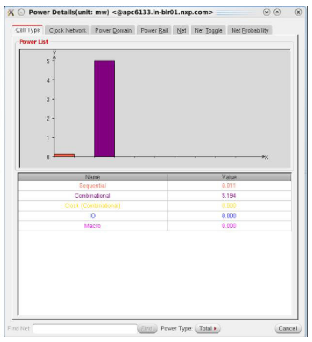
(a)



(b)

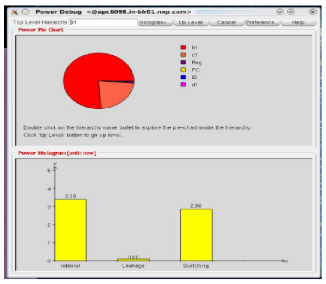
Figure 11. Result of division instruction for **(a)** Parallel CORDIC and **(b)** Pipelined CORDIC.

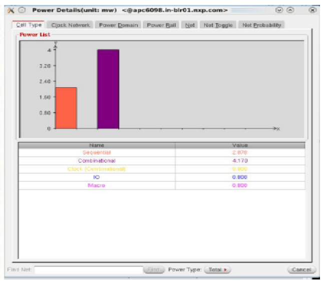




(a) (b)

Figure 12. Parallel CORDIC **(a)** Graphical display of power dissipated by individual module and **(b)** Power contribution by combinational and sequential gates.





(a) (b)

Figure 13. Pipelined CORDIC **(a)** Graphical display of power dissipated by individual module and **(b)** Power contribution by combinational and sequential gates.

Table 3. Comparison with existing CORDIC processors

	Frequency of operation	**No of operand Bits**
DDS using Hybrid	380MHz	13
CORDIC architecture[2]		
Sine/Cosine wave	13.3MHz	32
generator[3]		
Scale free Hyperbolic	56.38 MHz	16
CORDIC processor[4]		
Current Work	261.36MHz	32
[Pipelined]		
Current Work	24.23MHz	32
[Parallel]		

6. Conclusion

In this work, the popular and elementary mathematical operations such as the sine, cosine, sinh, cosh, exponential, multiplication and division, which form the inevitable part of various DSP algorithms, communication systems and common scientific and technical calculations are hardwired into the processor. This reduces the computation time compared to the software implementations using microprocessor. The CORDIC is proved to be the most attractive algorithm due to the reason that with slighter modifications, all of the above operations are implemented through the simple shift and add operations. Both parallel and pipelined CORDIC architectures were realized and their results are compared.

Table 3 gives the performance comparison of designed processor with existing works. The novel concept being that both trigonometric and hyperbolic operations are integrated in a single processor. Processors in[2–4] employ CORDIC to design a DDS (Direct Digital Synthesizer) which is in turn used in arbitrary waveform generator. In this work, the pipelined processor with a small look up table acts as a waveform generator that generates trigonometric and hyperbolic waves. The limitation of this waveform generator is that frequency of generated trigonometric and hyperbolic waves is constant [sin,cos = 3.5MHz sinh, cosh = 7.9MHz]. An additional frequency divider circuit needs to be externally connected to the processor to control the frequency of waves. The comparison as shown in Table 2 clearly demonstrates that, there is ten times increase in frequency of operation from parallel implementation to pipelined implementation. It is concluded that the pipelined approach is ideal for applications demanding very high performance. On the other hand, the parallel approach is more suitable for applications, where reduction of area and power are of primary concern since pipelined processor demands 28% increase in number of gates as shown in Table 2.

7. Acknowledgement

This work would not have been made possible without the constant support of the Logic library team at NXP Semiconductors India PVT LTD, Bangalore. I am deeply indebted to Swanand Kulkarni for permitting me to use the industry standard tools and the resources.

8. References

1. Volder JE. The CORDIC trigonometric computing technique. IRE Trans Electron Computers. 1959; 8:330–4.
2. Caro DD, Petra N, Strollo AGM. A 380 MHz direct digital synthesizer/mixer with hybrid CORDIC architecture in 0.25 um CMOS. IEEE J Solid-State Circuits. 2007; 42:151–60.
3. Hsiao SF, Hu YH, Juang TB. A memory efficient and high speed sine/cosine generator based on parallel CORDIC rotations. IEEE Signal Process Lett. 2004; 11:152–5.
4. Aggarwal S, Meher PK, Khare K. Scale-Free Hyperbolic CORDIC Processor and Its Application to Waveform Generation. IEEE Transactions on Circuits and Systems—I: Regular Papers. 2013; 60(2):314–26.
5. Subha S, Muthaiah R. FPGA Implementation of RA-CORDIC Processor. Indian Journal of Science and Technology. 2013; 6(5):4403–9.
6. Aggarwal S, Meher PK, Khare K. Area-Time Efficient Scaling-Free CORDIC Using Generalized Micro-Rotation Selection. IEEE Transactions on Very Large Scale Integration Systems. 2012; 20(8):1542–46.
7. Walther JS. A unified algorithm for elementary functions. Spring Joint Computer Conf; 1971; USA. p. 379–85.
8. Hu X, Harber RG, Bass SC. Expanding the Range of Convergence of the CORDIC Algorithm. IEEE Transactions on Computers. 1991; 40(1):13–21.